Finitary Fairness*

Rajeev Alur[†] Thomas A. Henzinger[‡]

Department of Electrical Engineering and Computer Science
University of California, Berkeley, CA 94720.
Email: {alur,tah}@eecs.berkeley.edu

Abstract

Fairness is a mathematical abstraction: in a multiprogramming environment, fairness abstracts the details of admissible ("fair") schedulers; in a distributed environment, fairness abstracts the independent processor speeds. 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* $\mathit{fin}(F)$ of any given fairness assumption 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.

1 Introduction

Interleaving semantics provides an elegant and abstract way of modeling concurrent computation. 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.

*A preliminary version of this paper appears in *Proceedings of the Ninth IEEE Symposium on Logic in Computer Science*, pp. 52–61, 1994.

[†]On leave from Bell Laboratories, Lucent Technologies.

[‡]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 ARPA grant NAG2-892, and by the SRC contract 95-DC-324A.

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, [LPS82, Fra86, MP91]), 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 even if one process fails. We quote from their paper [FLP85]:

> 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 (i.e., it is closed under limits). 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 strong fairness, weak fairness, etc. [LPS82, Fra86]. For every choice of F, we obtain its finitary version $\mathit{fin}(F)$ 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 $\mathit{fin}(F)$ is particularly intuitive: while F prohibits a schedule if it postpones a task forever, $\mathit{fin}(F)$ also prohibits a schedule if there is no bound on how many consecutive times a task is postponed. In general, a fairness requirement F is an ω-regular liveness property [AFK88]. We show that the finitary version $\mathit{fin}(F)$, then, is still live, but not 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 [HMP94]. 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 assumption 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

Verification. We address the problem of verifying that a program satisfies a property under a finitary fairness assumption $fin(F)$. Since $fin(F)$ is not ω-regular, it is not 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 by the use of finitary fairness. 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 [LPS82, Fra86]. 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 non-faulty processes of a distributed system agree on a common output value [PSL80]. Although consensus cannot be reached in the asynchronous model if one process fails [FLP85], 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 fixed upper bound on the relative speeds of different components, but this bound is not known a priori [DLS88, AAT97, RW92]. 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 substitute the timing assumptions of the solution of [AAT97].

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:

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

The program P_0 consists of two processes, each with one transition. 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 assumption F_∞, the program P_0 satisfies the property

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

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. 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 (non-probabilistic) real-life scheduler, however, is finite-state 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 assumption F_∞ by the much stronger assumption F_1 that contains only two schedules, $(lr)^\omega$ and $(rl)^\omega$. Under F_1, the program P_0 satisfies the property

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

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 (similar definition is considered in [Jay88]).

Let F_k be the set of k-bounded schedules. The assumption F_k 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 assumption F_∞.

Three observations about F_ω are immediate. First, the finitary version 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 paint itself into a corner [AFK88]: every finite word over $\{l, r\}$ can be extended into a bounded schedule[1].

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

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

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_∞.

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 (non-faulty) 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 assumption 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,

[1] 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.

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 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.

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 [Büc62]. The class of ω-regular languages is very robust with many alternative characterizations (see [Tho90] 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 [GPSS80]. 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 $k \geq 1$.

Safety and Liveness

For an ω-language $\Pi \subseteq \Sigma^\omega$, let $\mathit{pref}(\Pi) \subseteq \Sigma^*$ 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 $\mathit{pref}(\Pi)$ then $\overline{w} \in \Pi$ [ADS86]. Every safety property Π is fully characterized by $\mathit{pref}(\Pi)$. 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 without acceptance conditions. Properties defined by temporal-logic formulas of the form $\Box\, p$, where p is a past formula of PTL, are safe and ω-regular. For every $k \geq 1$, the set F_k (Section 2) of k-bounded schedules is an ω-regular safety property.

The ω-language Π is a *liveness property* iff $\mathit{pref}(\Pi) = \Sigma^*$ [AS85]; that is, every finite word can be extended into a word in Π. The set F_∞ (Section 2) 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 $\overline{w}$ and $\overline{w}'$ 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 $\mathbf{F}_{\sigma\delta} \cap \mathbf{G}_{\delta\sigma}$.[2]

There is also a temporal characterization of the first two-and-a-half levels of the Borel hierarchy [MP90]. 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* $\mathit{fin}(\Pi)$ of an ω-language Π is the (countable) union of all ω-regular safety languages that are contained in Π.

By definition, the finitary restriction of every ω-language is in $\mathbf{F}_\sigma$. Also, by definition, $\mathit{fin}(\Pi) \subseteq \Pi$. The following theorem states some properties of the operator *fin*.

Theorem 1 *Let Π, Π' be ω-languages:*

1. $\mathit{fin}(\mathit{fin}(\Pi)) = \mathit{fin}(\Pi)$.
2. *fin is monotonic: if $\Pi \subset \Pi'$ then $\mathit{fin}(\Pi) \subset \mathit{fin}(\Pi')$.*
3. *fin distributes over intersection: $\mathit{fin}(\Pi \cap \Pi') = \mathit{fin}(\Pi) \cap \mathit{fin}(\Pi')$.*

Proof. The first two follow immediately from the definition of *fin*. Since $\Pi \cap \Pi'$ is contained in Π as well as in Π', from the monotonicity, we have $\mathit{fin}(\Pi \cap \Pi') \subseteq \mathit{fin}(\Pi) \cap \mathit{fin}(\Pi')$. To prove the inclusion $\mathit{fin}(\Pi) \cap \mathit{fin}(\Pi') \subseteq \mathit{fin}(\Pi \cap \Pi')$, consider a word $\overline{w} \in \mathit{fin}(\Pi) \cap \mathit{fin}(\Pi')$. 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 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 $\overline{w} \in \mathit{fin}(\Pi \cap \Pi')$, and (3) follows. ■

The following 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, $F_\omega = \mathit{fin}(F_\infty)$), and F_ω is neither ω-regular nor safe.*

Proof. Recall that $F_\omega = \cup_{k\geq 1} F_k$, and $\mathit{fin}(F_\infty)$ 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, $F_\omega \subseteq \mathit{fin}(F_\infty)$.

Now consider an ω-regular safety property G contained in F_∞. Suppose G is accepted by a Büchi automaton M_G over the alphabet $\{l, r\}$. Without loss of generality, assume that every state of M_G is reachable from some initial state, and every state is an accepting state (since G is a safety property). We wish 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. Thus $\overline{w} = \overline{w}' l^{k+1} \overline{w}''$, for $\overline{w}' \in \{l, r\}^*$ and $\overline{w}'' \in \{l, r\}^\omega$. Since M_G has only k states, it follows

[2] 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.

that there is a state s of M_G such that there is a path from the initial state to s labeled with $\overline{w}'l^i$ 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 the word $\overline{w}'l^\omega$ 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 $l^k \in \mathit{pref}(F_\omega)$. However, $l^\omega \notin F_\omega$, implying that F_ω is not closed under limits, that is, F_ω is not a safety property.

Now we prove that F_ω is not ω-regular. Suppose 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 $lrllrlllrllllr\ldots$). From the properties of ω-regular languages it follows that G contains a word w such that $\overline{w} = \overline{w}'\overline{w}''^\omega$ for two finite words $\overline{w}', \overline{w}'' \in \{l, r\}^*$. Since $\overline{w} \in F_\infty$, $\overline{w}''$ contains at least one l and one r symbol. This means that, for $k = |\overline{w}'| + |\overline{w}''|$, $\overline{w}$ 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. In general, the operator *fin* does not preserve liveness also. That is, it may happen that $\mathit{pref}(\Pi) = \Sigma^*$, but $\mathit{pref}(\mathit{fin}(\Pi)) \neq \Sigma^*$. However, when applied to ω-regular properties, liveness is preserved.

Theorem 3 *If* Π *be an* ω*-regular language then* $\mathit{pref}(\mathit{fin}(\Pi)) = \mathit{pref}(\Pi)$.

Proof. Since $\mathit{fin}(\Pi) \subseteq \Pi$ and *pref* is monotonic, $\mathit{pref}(\mathit{fin}(\Pi)) \subseteq \mathit{pref}(\Pi)$. To prove the inclusion $\mathit{pref}(\Pi) \subseteq \mathit{pref}(\mathit{fin}(\Pi))$, suppose Π is ω-regular language over Σ, and consider $\overline{w} \in \mathit{pref}(\Pi)$. From 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 $\overline{w} \in \mathit{pref}(\mathit{fin}(\Pi))$. ∎

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: $\mathit{pref}(F_\omega) = \{l, r\}^\omega$. This means that when executing a program, the fairness requirement F_ω, just like the original requirement F_∞, can be satisfied after any finite number of steps.

The operator *fin* is illustrated on some typical languages below:

$\mathit{fin}(\Diamond p) = \Diamond p$;
$\mathit{fin}(\Box p) = \Box p$;
$\mathit{fin}(\Diamond\Box p) = \Diamond\Box p$;
$\mathit{fin}(\Box\Diamond p) = \{\overline{w} \mid \exists k$ such that every subsequence of length k has some $p\}$;
$\mathit{fin}(\Box\Diamond p \rightarrow \Box\Diamond q) = \{\overline{w} \mid \exists k$ such that every subsequence with k p's has some $q\}$.

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 [MP91].

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 all $i \geq 0$, 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. 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 step of a computation $\overline{q}$ iff $\tau(q_i)$ is nonempty, and τ is *taken* at the i-th step 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 $\mathit{enabled}(\tau)$ and a boolean variable $\mathit{taken}(\tau)$. Let the *scheduling alphabet* Σ_T be the (finite) set of interpretations of these boolean variables, that is, Σ_T is the power set of the set $\{\mathit{enabled}(\tau), \mathit{taken}(\tau) \mid \tau \in T\}$. 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 allowed computations of the program. In general, F is an ω-regular liveness property [AFK88]. The requirement of liveness ensures that, when executing a program, a fairness requirement can be satisfied after any finite number of steps.

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 integers $i \geq 0$ with $\mathit{taken}(\tau) \in w_i$ or $\mathit{enabled}(\tau) \notin w_i$; that is, no transition is enabled forever without being taken. It is specified by the following temporal-logic formula

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

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 steps $i \geq 0$ with $\mathit{enabled}(\tau) \in w_i$, then there are infinitely many steps $j \geq 0$ with $\mathit{taken}(\tau) \in w_j$; that is, no transition is enabled infinitely often without being taken. It is a stronger requirement than the weak fairness ($\mathit{SF} \subset \mathit{WF}$), and is specified by the formula

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

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 sets are ω-regular, their finitary restrictions $\mathit{fin}(\mathit{WF})$ and $\mathit{fin}(\mathit{SF})$, which belong to $\mathbf{F}_\sigma$, are again live (Corollary 4).

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 nonnegative integer k, iff for all transitions τ of P, τ cannot be enabled for more than k consecutive steps without being taken, that is, for all integers $i \geq 0$, there is an integer j, $i \leq j \leq i + k$, with $\mathit{taken}(\tau) \in w_j$ or $\mathit{enabled}(\tau) \notin w_j$. A schedule is *weakly-bounded* if it is weakly-k-bounded for some $k \geq 0$. 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 integers $0 \leq i_1 < i_2 < \cdots < i_k$, if $\mathit{enabled}(\tau) \in w_{i_j}$ for all $1 \leq j \leq k$, then $\mathit{taken}(\tau) \in w_{j'}$ for some $i_1 \leq j' \leq i_k$. A schedule is *strongly-bounded* if it is strongly-k-bounded for some $k \geq 0$.

Theorem 5 *Let P be 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 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 G is accepted by a Büchi automaton M_G over the alphabet Σ_T. Without loss of generality, assume that every state of M_G is reachable from its initial state, and every path in M_G is an accepting path. 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 $enabled(\tau)$ and does not contain $taken(\tau)$. From 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, $T = \{l, r\}$, 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

Corollary 6 *If P is a transition system with at least two transitions, then 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 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 assumption 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 [HMP94, MMT91]. 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 been long realized that the timing-based model with unknown delays is different from, and often more appropriate than, the asynchronous model (with standard fairness) [DLS88, AAT97, RW92]. 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 [HMP94]. 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* $\overline{t}$ is an infinite nondecreasing and unbounded sequence of real numbers. For $\overline{t}$ to be consistent with a given computation $\overline{q}$ 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. Note that if a transition τ is enabled in all states q_n, for $i \leq n \leq j$, and τ is not taken in all states q_n, for $i \leq n < j$, then it has been continuously enabled for $t_{j+1} - t_i \leq u_\tau$ (recall that $taken(\tau) \in q_j$ iff $q_{j+1} \in \tau(q_j)$). Then, the time sequence $\overline{t}$ is *consistent* with the computation $\overline{q}$ iff for every transition $\tau \in T$,

[*Lower bound*] if $taken(\tau) \in q_j$, then for all steps i with $0 \leq i \leq j$ 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 steps k with $i \leq k \leq j$, and $taken(\tau) \notin q_k$ for all steps k with $i \leq k < j$, then $t_{j+1} - t_i \leq u_\tau$.

A *timed computation* $(\overline{q}, \overline{t})$ of the timed transition system $P_{\ell,u}$ consists of a computation $\overline{q}$ of P together with a consistent time sequence $\overline{t}$. 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 $\Pi(P)$, $\Pi_{\ell,u}(P)$ is also ω-regular [AD94] (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 $\cup_{\ell,u}\Pi_{\ell,u}(P)$ 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 $\cup_{\ell,u}\Pi_{\ell,u}(P)$ is same as the set $\Pi_{\mathit{fin}(WF)}(P)$.

Theorem 7 *Let P be a transition system with the set T of transitions, let WF be the weak fairness requirement for P, and let $\overline{q}$ be a computation of P. Then $\overline{q} \in \Pi_{\mathit{fin}(WF)}(P)$ iff for some function ℓ and u from T to $\mathbb{Q}_{>0}$, $\overline{q} \in \Pi_{\ell,u}(P)$.*

Proof. Consider a weakly-bounded computation $\overline{q} \in \Pi_{\mathit{fin}(WF)}(P)$. From Theorem 5, there is a nonnegative 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. $t_i = i$). Since time increases by 1 at every step, 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 steps (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 $\overline{q} \in \Pi_{\ell,u}(P)$ 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 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 the number of transitions in T be n. Let k be an integer such that $k > \frac{nu^*}{\ell^*}$. We claim that the schedule corresponding to $\overline{q}$ is weakly-k-bounded. Suppose not. Then there is a transition τ and $i \geq 0$ such that τ is enabled but not taken at all states $q_i, q_{i+1}, \ldots q_{i+k}$. At every step of $\overline{q}$, $taken(\tau')$ holds for some transition τ'. Since $k > \frac{nu^*}{\ell^*}$, it follows that, there is a transition τ' such that $taken(\tau')$ holds at more than $\frac{u^*}{\ell^*}$ distinct states between q_i and q_{i+k}. Since $\ell(\tau') \geq \ell^*$, from the assumption that $\overline{t}$ satisfies the lower bound requirement, we have $t_{i+k} - t_i > u^*$, and hence, $t_{i+k} - t_i > u(\tau)$. 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 [HMP94]. 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.

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 recursive safety properties that are contained in Π (an ω-language is *recursive* iff it is the language of a 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 $w_n w_{n+1} \ldots w_{n+k \cdot f(n)}$, for all $n \geq 0$. 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*. Automated tools for model checking have been successfully used to check the correctness of digital hardware and communication protocols [CK96]. Here we examine the effects of finitary fairness on the algorithms that underlie these tools.

Untimed systems

Consider a finite-state transition system P with the 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 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 (Corollary 6). The following theorem shows that finite-state verification for finitary fairness can be reduced to verification under standard fairness.

Theorem 8 *For all ω-regular languages Π_1 and Π_2, $\Pi_1 \cap \Pi_2$ 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 also 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 *For a finite-state program P with set Q of states, an ω-regular fairness requirement $F \subseteq Q^\omega$, and an ω-regular specification $\Phi \subseteq Q^\omega$, P satisfies Φ under the fairness assumption F iff P satisfies Φ under the fairness assumption $\mathit{fin}(F)$.*

Proof. We want to show that

$$\Pi(P) \cap \mathit{fin}(F) \subseteq \Phi \text{ iff } \Pi(P) \cap F \subseteq \Phi.$$

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 finite-state timed transition system $P_{\ell,u}$ with set Q of states. Suppose the specification does not refer to time at all, and is given as an ω-regular specification $\Phi \subseteq Q^\omega$. To verify that $P_{\ell,u}$ satisfies Φ, we want to check the inclusion $\Pi_{\ell,u}(P) \subseteq \Phi$. This problem is solved by constructing a Büchi automaton that recognizes the language $\Pi_{\ell,u}(P)$ [AD94]. This method is applicable only when the bound functions ℓ and u are fully specified. In the parametric verification problem, the bound maps are not fully specified [AHV93]. The bounds $\{\ell_\tau, u_\tau \mid \tau \in T\}$ are viewed as parameters: values of these parameters are not known, but are required to satisfy certain (linear) constraints (such as $\ell_{\tau_1} = u_{\tau_1}$, $u_{\tau_2} < \ell_{\tau_3}$). The parametric verification problem, then, is specified by

1. a finite-state transition system $P = (Q, \tau, Q_0)$,

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

Corollary 10 *If P is a finite-state transition system, Φ is ω-regular, and $LU = [T \mapsto \mathbb{Q}_{>0}]^2$, then the parametric verification problem of checking the inclusion $\Pi_{LU}(P) \subseteq \Phi$ is decidable.*

The parametric verification problem, in general, is undecidable if the class LU constrains the allowed choices of the bound maps [AHV93]. For instance, if LU requires that $\ell_\tau = u_\tau$ for each $\tau \in T$ (i.e., $LU = \{(\ell, \ell) \mid \ell \in [T \mapsto \mathbb{Q}_{>0}]\}$), then the parametric 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 [LPS82].

> 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 $\rho(q') \prec \rho(q)$.

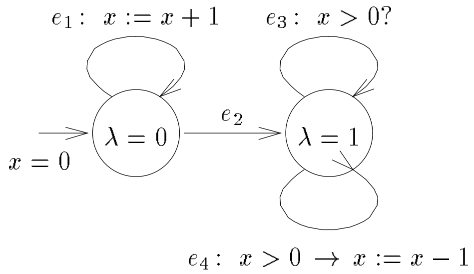

Figure 1: Program P_1

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 [LPS82]. 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 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 [LPS82].

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 finitary 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

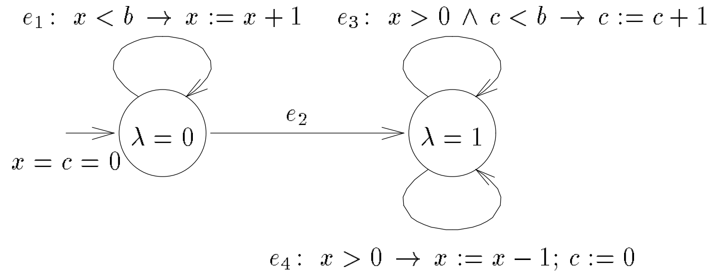

Figure 2: Transformed program *fin*(P_1)

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 R_{e_1} and R_{e_3} to be the empty sets; a state (λ, x) belongs 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 before P_1 terminates. This *unbounded nondeterminism* is what makes the mathematical treatment of fairness difficult.

Proving the termination of P_1 under finitary weak fairness—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 5). 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 nonnegative 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 assumption *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). That is, 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 $x \leq b$; $(1, x, c, b) \in R$ 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 nondeterminism. 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.

Finitary transformation of a program

Let us consider a general transformation for a given transition system $P = (Q, T, Q_0)$. Let T consist of m transitions $\tau_1, \tau_2, \ldots \tau_m$. The *finitary transformation* $\mathit{fin}(P)$ of the transition system P is obtained by introducing a new integer variable b, and for each transition τ_i, $1 \leq i \leq m$, a new integer variable c_i. Thus, the state space of $\mathit{fin}(P)$ is $Q \times \mathbb{N}^{m+1}$. The initial value of b is arbitrary; the initial value of each c_i is 0. Thus, the set of initial states of $\mathit{fin}(P)$ is $Q_0 \times \mathbb{N} \times \{0\}^m$. For every transition $\tau \in T$, the transition system $\mathit{fin}(P)$ contains a transition $\mathit{fin}(\tau)$ such that $(q', b', c'_1, \ldots c'_m) \in \mathit{fin}(\tau)(q, b, c_1, \ldots c_m)$ iff

1. $q' \in \tau(q)$;
2. $b' = b$;
3. for $1 \leq i \leq m$, if $\tau_i(q)$ is empty or $q' \in \tau_i(q)$ then $c'_i = 0$, else $c_i < k$ and $c'_i = c_i + 1$.

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 *A transition system P terminates under finitary weak fairness iff all computations of the transition system $\mathit{fin}(P)$ terminate.*

Proof. Suppose the program $\mathit{fin}(P)$ 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 $\mathit{fin}(P)$, it is clear that $\overline{q}'$ is also a computation of P. The value of the bound variable b stays unchanged throughout $\overline{q}$, let it be k. Furthermore, $c_i \leq k$, is an invariant over the computation $\overline{q}$ for all $1 \leq i \leq m$. Since for each transition τ_i, c_i is incremented each time τ_i 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 5, $\overline{q}$ is weakly-k-bounded for some k. Define the sequence $\overline{q}'$ over the state-space of $\mathit{fin}(P)$ as follows. For all $i \geq 0$, $q'_i = (q_i, k, c^i_1, \ldots c^i_m)$, where c^i_j is the maximum nonnegative integer n such that the transition τ_j is enabled but not taken in all states $q_{n'}$ for $i - n \leq n' < i$. It is easy to check that, since $\overline{q}$ is weakly-k-bounded, if the transition τ_j is enabled, but not taken, in state q_i then $c^i_j < k$. Consequently, $\overline{q}'$ is a (nonterminating) computation of $\mathit{fin}(P)$. ∎

Thus, the language $\Pi_{\mathit{fin}(WF)}(P)$ of finitary weak-fair computations of the transition system P is the projection of the language $\Pi(\mathit{fin}(P))$ of the transformed program $\mathit{fin}(P)$. It is known that given a transition system P and a fairness requirement F, there exists a transition system P' such that $\Pi_F(P)$ equals $\Pi(P')$ [Var87]. However, this transformation requires uncountably many states (the transformed program 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, $1 \leq i \leq n$, 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 $\mathit{decide}(v)$. To model failures, we introduce a special transition fail_i for each process. The transition fail_i is enabled only if the

```
Shared registers: initially: out =⊥, y[*] =⊥, x[*,*] = 0.
Local registers: initially: r_i = 1, v_i = in_i.
1. while out =⊥ do
2.        x[r_i, v_i] := 1;
3.        if y[r_i] =⊥ then y[r_i] := v_i fi;
4.        if x[r_i, ¬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).
```

Figure 3: Consensus, assuming finitary weak-fairness (program for process P_i with input in_i)

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 requirements, 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 requirements under the weak-fairness assumption WF [FLP85, LA87].

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 [AAT97]. We show that the consensus algorithm of [AAT97] 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 $\perp$, 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 $\perp$. 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 $x[r, \bar{v}]$ 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 $\bar{v}$, and every process that reaches round $(r + 1)$ has the preference v for that round. This ensures agreement (see [AAT97] for more details of proofs). It is easy to check that if all processes have the same initial input v, then no process will ever decide on $\bar{v}$; implying the requirement of validity.

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 $y[r] =\perp$ first (line 3), and one of them proceeds and chooses its preference for the next round (line 7) before the other one finishes the assignment to $y[r]$. The finitary fairness requirement ensures that this behavior cannot be repeated in every round. In every finitarily 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 requirements of agreement, validity, and wait-freedom under the finitary fairness assumption* $fin(WF)$.

By contrast, the program does not satisfy wait-freedom under the standard fairness assumption WF. Also observe that the algorithm uses potentially unbounded space, and therefore is not a finite-state program. The results of Section 4 imply 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 requirements of agreement, validity, and wait-freedom under the finitary fairness assumption* $fin(WF)$ *(or equivalently, in the unknown-delay model).*

The unknown-delay model of [DLS88] 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 protocol under various fault models. These bounds can be established using the finitary weak-fairness. Similar observation applies to the results on the session problem for the unknown-delay model [RW92].

Acknowledgments. Notions that are similar to k-bounded fairness, for a fixed k, have been defined in several places [Jay88]; 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.

References

[AAT97] R. Alur, H. Attiya, and G. Taubenfeld. Time-adaptive algorithms for synchronization. *SIAM Journal on Computing*, 26(2), 1997.

[AD94] R. Alur and D.L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126:183–235, 1994.

[ADS86] B. Alpern, A.J. Demers, and F.B. Schneider. Safety without stuttering. *Information Processing Letters*, 23:177–180, 1986.

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

[AHV93] R. Alur, T.A. Henzinger, and M.Y. Vardi. Parametric real-time reasoning. In *Proceedings of the 25th ACM Symposium on Theory of Computing*, pages 592–601, 1993.

[AS85] B. Alpern and F.B. Schneider. Defining liveness. *Information Processing Letters*, 21:181–185, 1985.

[Büc62] J.R. Büchi. On a decision method in restricted second-order arithmetic. In *Proceedings of the International Congress on Logic, Methodology, and Philosophy of Science 1960*, pages 1–12. Stanford University Press, 1962.

[CK96] E. Clarke and R. Kurshan. Computer-aided verification. *IEEE Spectrum*, 33(6):61–67, 1996.

[DLS88] C. Dwork, N. Lynch, and L. Stockmeyer. Consensus in the presence of partial synchrony. *Journal of the ACM*, 35(2):288–323, 1988.

[FLP85] M. Fischer, N. Lynch, and M. Paterson. Impossibility of distributed consensus with one faulty process. *Journal of the ACM*, 32(2):374–382, 1985.

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

[GPSS80] D. Gabbay, A. Pnueli, S. Shelah, and J. Stavi. On the temporal analysis of fairness. In *Proceedings of the Seventh ACM Symposium on Principles of Programming Languages*, 1980.

[HMP94] T.A. Henzinger, Z. Manna, and A. Pnueli. Temporal proof methodologies for timed transition systems. *Information and Computation*, 112(2):273–337, 1994.

[Jay88] D.N. Jayasimha. *Communication and Synchronization in Parallel Computation*. PhD thesis, Univ. of Illinois at Urbana-Champaign, 1988.

[LA87] M. Loui and H. Abu-Amara. Memory requirements for agreement among unreliable asynchronous processes. *Advances in Computing Research*, 4:163–183, 1987.

[LPS82] D. Lehman, A. Pnueli, and J. Stavi. Impartiality, justice, and fairness: The ethics of concurrent termination. In *Automata, Languages and Programming: Proceedings of the Ninth ICALP*, LNCS 115, pages 264–277. Springer-Verlag, 1982.

[MMT91] M. Merritt, F. Modugno, and M. Tuttle. Time constrained automata. In *Proceedings of Workshop on Theories of Concurrency: CONCUR'91*, 1991.

[MP90] Z. Manna and A. Pnueli. A hierarchy of temporal properties. In *Proceedings of the Ninth ACM Symposium on Principles of Distributed Computing*, pages 377–408, 1990.

[MP91] Z. Manna and A. Pnueli. *The temporal logic of reactive and concurrent systems*. Springer-verlag, 1991.

[PSL80] M. Pease, R. Shostak, and L. Lamport. Reaching agreement in the presence of faults. *Journal of the ACM*, 27(2):228–234, 1980.

[RW92] I. Rhee and J. Welch. The impact of time on the session problem. In *Proceedings of the 11th ACM Symposium on Principles of Distributed Computing*, pages 191–201, 1992.

[Tho90] W. Thomas. Automata on infinite objects. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B, pages 133–191. Elsevier Science Publishers, 1990.

[Var87] M.Y. Vardi. Verification of concurrent programs: the automata-theoretic framework. In *Proceedings of the Second IEEE Symposium on Logic in Computer Science*, pages 167–176, 1987.