

Bounded Fairness

Nachum Dershowitz^{1,*}, D. N. Jayasimha^{2,**}, and Seungjoon Park²

¹ School of Computer Science, Tel Aviv University
Ramat Aviv, Tel Aviv 69978, Israel
email: nachum.dershowitz@cs.tau.ac.il

² Platform Architecture Research, Enterprise Platforms Group
Intel Corporation
Santa Clara, CA 95052, USA
email: {jay.jayasimha,seungjoon.park}@intel.com

*Dedicated with boundless gratitude to
Zohar Manna
in honor of his 2⁶-th birthday.*

Abstract. *Bounded fairness* is a stronger notion than ordinary eventuality-based fairness, one that guarantees occurrence of an event within a fixed number of occurrences of another event. We formalize bounded fairness by introducing a new family of temporal modal operators. The resultant logic is equivalent to temporal logic with the *until* modality, but more succinct.

This logic can be used to specify bounded fairness requirements in a more natural manner than possible with *until*. It can, for example, relate the frequency of shared resource access of a particular process to other processes that access the resource with mutual exclusion. As applications of bounded fairness, we verify requirements for some standard concurrent programming problems and for a new cache protocol. We also show that Dekker's mutual exclusion algorithm is fair in the conventional sense, but not bounded fair.

Was euer booke containing such vile matter
So fairely bound?

*The most excellent and lamentable tragedie,
of Romeo and Iuliet*
—William Shakespeare (1599)

* Research supported in part by the Israel Science Foundation (grant no. 254/01).

** Most of this author's work was performed while at the University of Illinois and at The Ohio State University.

1 Introduction

Fairness means that every process gets an opportunity to make progress, regardless of what other processes may do. The distinguishing feature of a large class of fairness notions is *eventuality*, that is, fairness is defined as a restriction on some infinite behaviors, relating to the eventual occurrence of some events [9]. Temporal logic, with modal operators \Box and \Diamond , has been commonly used as a tool to specify and analyze such fairness properties, in large measure due to the pioneering work of Pnueli and Manna (see [21, 24, 25]). Gabbay et al. [10] introduced the \mathcal{U} (*until*) operator to formalize aspects of responsiveness (for example, the absence of unsolicited response) and fairness (for example, “strict” fairness).

For many applications (including real-time applications), the weak guarantee of eventual occurrence may be insufficient. Instead, systems such as flight control and process control systems require that time bounds on their reactive behavior be met, which makes it necessary to specify and reason explicitly about the frequency of occurrences of events. Many explicit-time logics have been proposed for such applications (see [11, 22, 26]). Whereas eventual occurrence is too weak a commitment, explicit mention of time is constrictive and unpleasant in many situations. For example, the following is a plausible requirement: any process p that requests a resource (such as a critical region) be granted the resource within the next k times it is granted to a process arriving after p . This requirement relates the frequency of shared resource access of a particular process with that of other processes (which access the resource with mutual exclusion). It is stronger than the eventuality requirement, but, nonetheless, does not require explicit mention of time.

This parameterized notion of *k-fairness* allows one to express a variety of fairness requirements elegantly, ranging from $k = 1$, corresponding to the (absolutely fair) “first come first served” discipline, to $k = \infty$, corresponding to the (totally unrestricted) eventuality concept. Bounded fairness, though suited to real-time applications, makes no assumption about the relative progress of processes; that is, a process’s execution rate on a processor is independent of the execution rate of another process. Such rate assumptions would make solutions more restrictive and time-dependent.

Example 1 (Family Status). Consider a system comprising three variables: z and m are Boolean; n is a natural number. Suppose three processes are running concurrently:

- Z can set or reset z , or idle with z unchanged;
- M can set or reset m , or idle with m unchanged;
- N can increment n , or idle with n unchanged.

The specification

$$z \diamond_3 m$$

demands that m hold true within three occurrences of z . The requirement

$$m \wedge y = n \leq 4 \rightarrow z \diamond_8 n > y$$

insists that n grow within eight occurrences, provided m holds and n does not exceed four.³

In the literature, bounded fairness has been mentioned in specific contexts or en passant. For example, Fagin and Williams [8] define an intuitively similar notion in the context of a carpool scheduling algorithm. Manna and Pnueli [20] mention “bounded overtaking,” which is the kind of fairness we have in mind and suggested in [13, 15]. Here, we provide practical motivation for using this concept, and illustrate its application in rigorous temporal-logic analyses.

In the next section, we describe bounded fairness and show how to specify it for some standard concurrent programming problems. In Section 3, we incorporate the family of binary modalities \diamond_k into linear temporal logic, to formally capture our intuitive idea of bounded fairness. Section 4 lists some properties of k TL, and proves that this logic is precisely as powerful as temporal logic with the *until* operator \mathcal{U} . We also demonstrate the succinctness made possible with the new operator. In Section 5, we illustrate several applications of bounded fairness. We show that Dekker’s solution to the (two process) mutual exclusion problem is *fair* in the conventional sense, but *not* k -fair for any fixed value of k . We also show that a shared-memory distributed synchronizer and a starvation-free mechanism for a retry-based cache-coherence algorithm are both bounded fair.

2 k -Fairness

We work in the context of the concurrent processing of several asynchronous processes. Though we will define fairness using the state transition model of Burns et al. [4], it will be obvious that bounded fairness could just as well be defined for other abstract models of concurrent computation (for example, the one described in [6]).

Our process model comprises a set of states with transition functions. Each process p_i is a triple $\langle V, X_i, \tau_i \rangle$, where V is a shared set of values and X_i is a (possibly countably infinite) set of states partitioned into disjoint sets R_i , T_i , C_i , and E_i , corresponding to the *remainder*, *trying*, *critical*, and *exit* regions of process p_i , respectively. The remainder set R_i is non-empty; the other partitions can be empty. The state transition function $\tau_i: V \times X_i \rightsquigarrow V \times X_i$ has the following restricted behavior:

$$\begin{aligned} \tau_i &: V \times (R_i \cup T_i) \rightsquigarrow V \times (T_i \cup C_i) \\ \tau_i &: V \times (C_i \cup E_i) \rightsquigarrow V \times (R_i \cup E_i) . \end{aligned}$$

³ Legend: z is true if Zohar called; m , if one is in the married state; n is the number of one’s children.

For convenience, we refer to a process by the region to which it currently belongs.

We assume a linearly ordered sequence of time instants. The usual fairness requirement expressed in linear temporal logic [24], and using the above process model, is

$$p_i \in T_i \rightarrow \diamond p_i \in C_i, \quad (1)$$

stating that process p_i , currently in its trying region, eventually (at some subsequent instant) enters its critical region.

For some applications, one needs a stronger assertion than (1), namely that the entry of any process into its critical section is k -fair. The parameter k is a fixed positive integer referred to informally as the *bound*. A formula $z \diamondstar m$ may be read “ m is k -bounded to z ” and is true at a particular state if and only if m is true at one of the next k instants that z is true. Define the following proposition:

q_i : The scheduler allows a process p_i , or another process arriving after p_i , into the respective critical region.

Then, k -fairness for p_i is expressed as follows:

$$p_i \in T_i \rightarrow q_i \diamondstar p_i \in C_i, \quad (2)$$

meaning that any process p_i wanting to enter its critical region is guaranteed to do so at one of the next k times that a process is scheduled.

The following examples illustrate how bounded fairness requirements may be specified for standard concurrent programming situations using the \diamondstar operator in a more natural manner than possible with either *until* or *atnext* [18].

Example 2 (Monitor). A *monitor* is a language construct for process synchronization. A process must wait on a condition variable, say B , if it finds that it should not be granted access to a particular section of the program. The operations defined on B are **wait** (delay the process waiting on B) and **signal** (awake the process waiting on B) [12]. Let ℓ be the label at which a process p waits. Define the following:

- $at \ell$: Control flow is at the beginning of statement ℓ .
- $after \ell$: Control has just completed execution of ℓ .
- r : The process waiting on B is awoken.
- q : Operation **signal**(B) awakens process p waiting on B , or wakes a process waiting on B that arrived after p .

The standard fairness property for p is

$$\square \diamond (at \ell \wedge r) \rightarrow \diamond after \ell,$$

where an assertion $\square \diamond z$ means that z is true infinitely often. The stronger k -fairness property (2) is

$$\square \left[at \ell \rightarrow \left(q \diamondstar after \ell \right) \right].$$

Example 3 (Five Dining Philosophers). Consider the familiar Dining Philosophers Problem, as originally formulated by Dijkstra [5]. Suppose we do not require a strict precedence in granting forks to a philosopher according to the order of the request made, since such a requirement holds up resources (forks). Instead, allow philosopher i 's neighbors to hold the forks at most once out of turn after i expresses the wish to use them. Let T_i^{forks} and C_i^{eat} be the trying and critical regions, respectively, of the i th philosopher's process, φ_i . Define:

q_i : The scheduler allows φ_i , or its neighboring processes φ_{i+1} and φ_{i-1} (+ and $-$ are modulo 5) arriving after φ_i , to enter the respective critical region.

The specification for this bounded requirement is as follows:

$$\bigwedge_{i=1}^5 \square \left[\varphi_i \in T_i^{\text{forks}} \rightarrow \left(q_i \diamond_3 \varphi_i \in C_i^{\text{eat}} \right) \right].$$

3 Semantics of k TL

We define now the syntax and semantics of k TL, our linear-time logic with temporal operators \diamond_k . For convenience, we also use other well known temporal operators; later (Proposition 3), we will show how these operators can also be expressed in terms of \diamond_k . Consider a language with atomic propositions z_0, z_1, \dots , the usual propositional connectives, \neg , \wedge , and \vee (and other connectives as abbreviations for combinations of these), the truth constants, T and F , the temporal connectives, \diamond , \square , \bigcirc , \mathcal{U} , *atnext*, and the new connectives \diamond_k for each positive integer k . We refer to well-formed formulæ in this language as " k TL formulæ."

A *run (model)* σ is given by:

- A (countably infinite) sequence of states, $\sigma_0, \sigma_1, \dots$, indexed by "instants."
- A truth assignment $\sigma, i \models z_n$ for each atomic proposition z_n in each individual state σ_i .

The semantics of the temporal operators are given by inductively extending the truth assignment to all temporal formulæ. The meanings of propositional connectives and the standard temporal operators, \diamond , \square , and \bigcirc , are as usual [21]. In addition, we define:

$$\begin{aligned} \sigma, j \models m \mathcal{U} z \text{ iff} \\ & \sigma, k \models z \text{ for some } k > j \text{ and } \sigma, i \models m \text{ for all } i, j < i < k. \\ \sigma, j \models m \text{ atnext } z \text{ iff} \\ & \sigma, k \models m \wedge z \text{ for some } k > j \text{ and } \sigma, i \not\models z \text{ for all } i, j < i < k. \\ \sigma, j \models z \diamond_k m \text{ (} k > 0 \text{) iff} \\ & \text{there are } j < t_1 < \dots < t_\ell, 1 \leq \ell \leq k, \text{ such that } \sigma, t_\ell \models m, \\ & \sigma, t_i \models z \text{ for all } i, 1 \leq i \leq \ell, \text{ and} \\ & \sigma, t \not\models z \text{ for all } t, t_{i-1} < t < t_i, 1 \leq i \leq \ell. \end{aligned} \quad (3)$$

By convention, temporal operators do *not* include the present state. The definitions of \mathcal{U} and $atnext$ are “strong” in the sense that assertion z must hold in the future for the formulæ to be true.

For completeness, we ought to also define 0-fairness:

$$z \diamond_0 m \Leftrightarrow m.$$

A k TL formula ϕ is *satisfiable* if there is a run σ such that $\sigma, 0 \models \phi$; it is *valid* if $\sigma, 0 \models \phi$ for all σ .

4 Expressiveness of k TL

Any formula $z \diamond_k m$ can be unfolded into k nested *untils*, k occurrences of \diamond_1 , and linearly many duplications of z and m :

Proposition 1. *The following equivalence is valid for all positive integers k :*

$$z \diamond_{k+1} m \Leftrightarrow \left(z \diamond_1 m \right) \vee \left[\neg z \mathcal{U} \left(z \diamond_k m \right) \right].$$

Proof. Consider what it means (by 3) for the left side to be true at instant i for run σ , namely, that there are ℓ , $1 \leq \ell \leq k + 1$, subsequent instants $i < i_1 < i_2 < \dots < i_\ell$ at which z holds, but between which z is false, and, furthermore, m is true at the last instant, i_ℓ . The meaning of the second disjunct on the right is that z is false until some instant i_1 (when it may or may not be true), true at exactly the instants $i_2, \dots, i_{\ell'}$ (for some $\ell' \leq k + 1$), while m is true at $i_{\ell'}$. Comparing the two, we see that the left side is also true when m and z are both true at i_1 , which is exactly covered by the disjunct $z \diamond_1 m$.

If, on the other hand, the first disjunct of the right side is true, then so is the left side. Otherwise, for the right side to be true, m must be false when z is first true, but true one of the next k times it is, which also makes the left side true. \square

Alternatively:

Proposition 2. *The following equivalence is valid for all natural numbers k :*

$$z \diamond_{k+1} m \Leftrightarrow z \diamond_1 \left[m \vee \left(z \diamond_k m \right) \right].$$

For convenience, we may define a dual notion to that of bounded eventuality, namely *bounded contemporaneity*:

$$z \square_k m \Leftrightarrow \neg \left(z \diamond_k \neg m \right),$$

under which m must be true *each* of the next k instants z is.

The other temporal connectives can be expressed succinctly in terms of \diamond_1 :

Proposition 3. *The following translations are valid:*

$$\begin{aligned} \diamond m &\Leftrightarrow m \diamond_1 T \\ \bigcirc m &\Leftrightarrow T \diamond_1 m \\ \square m &\Leftrightarrow \neg \left(\neg m \diamond_1 T \right) \\ m \text{ atnext } z &\Leftrightarrow z \diamond_1 m. \end{aligned}$$

Kamp [16] showed that $\mathcal{L}(\mathcal{U})$, the language with \mathcal{U} as the only temporal connective (plus atomic sentences and the usual logical connectives), is as expressive as the first-order theory of linear order (given by the set of natural numbers with $=$ and $<$ and a set of monadic predicates). Based on this, $\mathcal{L}(\mathcal{U})$ is said to be *expressively complete*.

Theorem 1. *kTL is of the same expressive power as $\mathcal{L}(\mathcal{U})$.*

Proof. The temporal connective \mathcal{U} can be expressed in terms of \diamond_1 , as follows:

$$z \mathcal{U} m \Leftrightarrow (z \rightarrow m) \diamond_1 m. \quad (4)$$

Consider the left side: it holds in state σ_i if m is true in some state σ_j , $j > i$, and z and $\neg m$ hold in each state of the (possibly empty) sequence $\sigma_{i+1}, \dots, \sigma_{j-1}$. Similarly, the right side holds in σ_i when m is true at some state σ_j , $j > i$, at which time $z \rightarrow m$ must also be true; at all other states in the sequence $(\sigma_{i+1}, \dots, \sigma_{j-1})$, z is true and m is false.

For the other direction, it is fairly obvious that \diamond_1 can be expressed in terms of \mathcal{U} in the following manner:

$$z \diamond_1 m \Leftrightarrow \neg z \mathcal{U} (m \wedge z).$$

Thus, we have that $\mathcal{L}(\diamond_1)$, the language with just \diamond_1 , is of equal expressiveness as $\mathcal{L}(\mathcal{U})$.

By the preceding two propositions, the temporal connectives of k TL may all be expressed in terms of \diamond_1 (or, for that matter, \mathcal{U}). So $\mathcal{L}(\mathcal{U})$ and $\mathcal{L}(\diamond_1)$ are as expressive as all of k TL:

$$\mathcal{L}(\diamond, \square, \bigcirc, \mathcal{U}, \text{atnext}, \diamond_1, \diamond_2, \dots). \quad \square$$

As a corollary, since $\mathcal{L}(\mathcal{U})$ is more powerful than $\mathcal{L}(\diamond, \square)$ [10], our logic k TL is also more powerful.

There is an inter-dependence between the iterated *atnext* operator of [18] and our \diamond_k , first proposed in [15]. Informally, $m \text{ atnext}^k z$ means, “ m holds at the k th next state at which z holds,” but may hold earlier, too. Let’s symbolize atnext^k by an infix circle operator \odot_k . It is straightforward to see that:

Theorem 2. *The following equivalences are valid for all positive integers k :*

$$\begin{aligned} z \diamond_{k+1} m &\Leftrightarrow \bigvee_{i=1}^{k+1} \left[m \circlearrowleft_i z \right] \\ z \circlearrowleft_{k+1} m &\Leftrightarrow z \circlearrowleft_1 \left(z \circlearrowleft_k m \right). \end{aligned}$$

Furthermore, one can show by induction that:

Corollary 1. *The following expansions are valid for all positive integers j and k :*

$$\begin{aligned} z \diamond_{j+k} m &\Leftrightarrow z \diamond_j \left[m \vee \left(z \diamond_k m \right) \right] \\ z \circlearrowleft_{j+k} m &\Leftrightarrow z \circlearrowleft_j \left(z \circlearrowleft_k m \right). \end{aligned}$$

Timed (metric) temporal operators (as in [17], for example) are also easily expressible. For example, the formula $\diamond_{\leq k} m$, meaning that m holds within the next k instants, is simply $T \diamond_k m$.

Finally, we point out that the operator \diamond_k contributes to the succinctness of expressions that would otherwise require multiple occurrences of \mathcal{U} :

Theorem 3. *Formulae of k TL can be much more succinct than those of $\mathcal{L}(\mathcal{U})$.*

Proof. We have seen (4) how to transform any occurrence of \mathcal{U} into one instance of \diamond_k . On the other hand, expressing \diamond_k requires nested \mathcal{U} 's. Specifically, the formula STAIR_{k+1} of [7], which expresses the claim that there is a subrun with $k+1$ occurrences of event a but no b 's, can be expressed using (the dual of) \diamond_k , as follows:

$$\diamond \left\{ a \square_k [\neg b \wedge (\neg b \mathcal{U} a)] \right\}.$$

But, as shown in [7, Thm. 4.1], expressing STAIR_{k+1} in $\mathcal{L}(\mathcal{U})$ requires k nested \mathcal{U} 's. \square

Since propositional temporal logic with *until* is decidable, it follows from Theorem 1 that k TL is likewise decidable, for example by designing a semantic tableau (cf. [27]).

5 Applications of k TL

This section considers the analysis of three programs for fairness, running the gamut from linear-bounded fairness to unbounded fairness.

Example 4 (Distributed Synchronizer). A synchronization primitive, particularly suited for implementation on large shared memory multiprocessors, and called the *Distributed Synchronizer*, was introduced in [14]. It was shown there that for a multiprocessor with n processing elements, the *DSP* and *DSV* operations (the usual *P* and *V* operations implemented with the distributed synchronizer) are $(2n - \lfloor \log_2 n \rfloor - 1)$ -fair.

Our next example appeals to the following:

Lemma 1. *For all positive integers k and ℓ , the following implication is valid:*

$$\square \left(p \diamond_k q \right) \wedge \left((p \wedge q) \diamond_\ell r \right) \Rightarrow p \diamond_{k \cdot \ell} r .$$

Proof. Consider a run σ such that $\sigma, 0 \models \square (p \diamond_k q) \wedge [(p \wedge q) \diamond_\ell r]$. By hypothesis, $\sigma, 1 \models p \diamond_k q$, which (by 3) implies that there should be instants $1 < t_1 < \dots < t_{k_1}$, $0 < k_1 \leq k$, such that $\sigma, t_{k_1} \models p \wedge q$ and p false at all intervenient instants. Furthermore, $\sigma, t_{k_1} \models p \diamond_k q$, which implies that there are $t_{k_1} < t_{k_1+1} < \dots < t_{k_2}$, $k_1 < k_2 \leq k_1 + k$, for which $\sigma, t_{k_2} \models p \wedge q$. Consider the endless sequence $t_{k_1} < t_{k_2} < \dots$, constructed in this manner, with $\sigma, t_{k_i} \models p \wedge q$, for all i , and $\sigma, j \models \neg p$, for all $t_{k_i} < j < t_{k_{i+1}}$. The hypothesis $\sigma, 0 \models (p \wedge q) \diamond_\ell r$ implies that there is an instant $t_{k_{\ell'}}$, $\ell' \leq \ell$, in this sequence that satisfies $\sigma, t_{k_{\ell'}} \models r$. Now, we can construct a sequence $i < t_1 < \dots < t_{k_1} < t_{k_1+1} < \dots < t_{k_2} < t_{k_2+1} < \dots < t_{k_{\ell'}}$, of length at most $k\ell$, at each instant of which p holds (but not in between), because $\ell' \leq \ell$ and $k_i < k_{i+1} \leq k_i + k$, for all i ($k_0 = 0$). The run σ therefore satisfies the intended meaning (3) of the consequent of the formula. That is,

$$\sigma, 0 \models p \diamond_{k \cdot \ell} r . \quad \square$$

Example 5 (Scalability Port). An application of bounded fairness arises in *retry-based* multiprocessor cache-coherence protocols in which source agents make caching requests to home agents associated with cache lines. In such protocols, requests may need to be retried when there are multiple requests to the same cache line—referred to as an *address conflict*, or when there are no available resources to grant at the home agent—referred to as a *resource conflict*. In either case, it is important that a particular source agent does not starve because it encounters one of the two conflicts each time its request arrives at a home agent. Intel’s Scalability Port specification [2] is an example of a retry-based protocol.

A total of N caching agents are competing for shared resources to process transactions to shared memory. The resources are managed by a starvation-prevention algorithm that enforces fairness among the multiple agents, while prioritizing retried transactions resulting from address conflicts. Home agents prioritize retried caching agents over other serviced agents for the shared resource in the future.

Priorities assigned on account of resource conflict and address conflict are tracked separately by each home agent using two vectors, R (priority vector due to resource conflict) and A (priority vector due to address conflict), indexed by caching agents. Vector entries can be in a *Normal*, *Prioritized*, or *Serviced* state.

A request may be serviced if it is from a *Prioritized* agent and the resource is available. Whenever a request is serviced, the corresponding entry is set to *Serviced*. Whenever a request is retried, the corresponding entry is set to *Prioritized* unless it is already *Serviced*. When there are no *Prioritized* agents, the vector reverts to *Normal*. When a retry results from an address conflict, the resource

is, in addition, marked by a conflict flag and reserved for the specific cache line. This flag is reset when the entries in A transition back to *Normal*.

Define the following propositions:

p : A new request is serviced.

q : The conflict flag is reset and vector A reverts to *Normal*.

r : The resource is acquired by a request from a particular caching agent.

Once a conflict flag is set for a resource, there can be at most N requests, one from each caching agent, that are serviced by that resource before the home agent's A vector gets back to *Normal* and the flag is cleared. Therefore, the algorithm satisfies $\square(p \diamond q)$. Once both p and q events occur, only *Prioritized* agents in R get serviced, because there cannot be a resource with its address conflict flag set. Therefore, caching agents are guaranteed to get serviced before such an event recurs N times. Consequently, the algorithm satisfies $\square((p \wedge q) \diamond r)$.

Applying the lemma, with $k = \ell = N$, and p , q , and r , as above, we have:

$$\square \left(p \diamond_N q \right) \wedge \left((p \wedge q) \diamond_N r \right) \Rightarrow p \diamond_{N^2} r,$$

and conclude that every caching agent's request will get serviced before N^2 subsequent requests are.

Example 6 (Dekker's Algorithm). One can show that Dekker's solution (see Fig. 1) to the two-process mutual-exclusion problem [5, 19] is not k -fair, for *any* fixed value of k . The trying regions in this case are $T_1 = \{z_1, \dots, z_6\}$ and $T_2 = \{m_1, \dots, m_6\}$; the critical regions are $C_1 = \{z_7, z_8\}$ and $C_2 = \{m_7, m_8\}$; the exit regions are $E_1 = \{z_9\}$ and $E_2 = \{m_9\}$; and the remainder regions are $R_1 = \{z_0\}$ and $R_2 = \{m_0\}$. For process p_1 , the k -fairness requirements are:

$$\begin{aligned} p_1 \in T_1 \wedge p_2 \in T_2 &\rightarrow \text{after } z_2 \vee \left(\text{after } m_2 \diamond_{k+1} \text{after } z_2 \right) \\ p_1 \in T_1 \wedge p_2 \notin T_2 &\rightarrow \text{after } z_2 \vee \left(\text{after } m_2 \diamond_k \text{after } z_2 \right). \end{aligned}$$

Assume that the processor executing Process 1 is at z_5 and that it takes a long time to execute z_5 (making no assumptions about the relative speeds of processes). Meanwhile, the processor executing Process 2 executes m_7, \dots, m_1 , and is at m_2 . It still finds that $y_1 = \text{false}$, since z_5 and z_6 have not completed. Thus, it enters its critical region though $p_1 \in T_1$. This, in fact, can happen any number of times. Hence, the algorithm is not k -fair for *any* fixed value of k . It is, however, fair in the unbounded sense, since Process 1 will eventually be able to enter its critical region. (Each instruction takes only a finite amount of time to execute.)

I: $t := 1$; $y_1 := false$; $y_2 := false$;

Process 1	Process 2
z_0 : <i>non-critical</i>	m_0 : <i>non-critical</i>
z_1 : $y_1 := true$	m_1 : $y_2 := true$
z_2 : if $\neg y_2$ then goto z_7	m_2 : if $\neg y_1$ then goto m_7
z_3 : if $t = 1$ then goto z_2	m_3 : if $t = 2$ then goto m_2
z_4 : $y_1 := false$	m_4 : $y_2 := false$
z_5 : await $t = 1$	m_5 : await $t = 2$
z_6 : goto z_1	m_6 : goto m_1
z_7 : $t := 2$	m_7 : $t := 1$
z_8 : $y_1 := false$	m_8 : $y_2 := false$
z_9 : goto z_0	m_9 : goto m_0

Fig. 1. Dekker’s algorithm

6 Discussion

Most fairness properties involve the temporal concept, “eventually.” Eventuality, however, is a relatively weak concept with which to specify desirable properties of many concurrent programs. Instead, we have proposed a stronger notion, called *k-fairness*, and introduced a new set of binary temporal modalities, \diamond_k , into linear temporal logic. The definition is elegant in that time is not explicit. As we have seen, one is able to express a variety of fairness notions, ranging from strict first-come-first-served fairness to eventuality, and to analyze standard concurrent programming problems and mutual exclusion algorithms for bounded fairness.

Specifically, we have seen that:

1. There is a linear bound, in the number of processors, for the Distributed Synchronizer.
2. The bound for the retry-based Scalability Port algorithm is quadratic in the number of agents.
3. There is no fairness bound for Dekker’s algorithm.

One can similarly analyze other concurrent algorithms for bounded fairness. In particular:

4. The solution to the Dining Philosophers Problem utilizing monitors (presented in [3]) is 1-fair.
5. Peterson’s two process Mutual Exclusion Algorithm [23] is 2-fair, since it permits a process executing in its trying region to be overtaken at most once.

The concept of bounded fairness developed here lends itself to a range of quantitative variants. Consider the assertion, “ m is true the k th time z is, *but not before*,” which we might denote

$$z \nabla^k m .$$

This temporal connective can be expressed in terms of our \diamond_k operator as follows:

$$\begin{aligned} z \nabla^0 m &\Leftrightarrow m \\ z \nabla^{k+1} m &\Leftrightarrow \neg \left(z \diamond_k m \right) \wedge \left(z \diamond_{k+1} m \right) . \end{aligned}$$

All in all, we now have a full spectrum of related temporal modalities:

$$z \diamond_k m \quad z \bigcirc_k m \quad z \nabla^k m \quad z \square_k m .$$

Each refers to the next k instants when event z is true. The first asks only that m hold at one of them; the second demands that it be true precisely at the last occurrence; the third adds that it not hold before; the last requires m to be true at all of them.

Alur and Henzinger [1] have promoted an idea called “finitary fairness,” which imposes bounds on the relative frequency in scheduling of a set of events, and have developed the formal semantics of their F_∞ modality. While that notion applies to a set of events uniformly, k -fairness can specify fairness of one *particular* event relative to another. One is tempted to suggest a locally-finitary operator \diamond_* , meaning \diamond_k for some k . Unlimited use of unbounded modalities $\exists k. z \diamond_k m$, however, gives an undecidable logic.

We have only described bounded modalities for the future fragment of time. It would be natural to also refer to the past by allowing negative k in k TL. Though this would not add expressive power to the temporal logic of discrete linear time with a *since* operator, it would perhaps allow simpler specifications and simpler proofs of past program behavior.

Acknowledgements

We thank Tim Carlson, Ching-Tsun Chou, Ganesh Gopalakrishnan, and Neelam Soundararajan for useful discussions, Amir Pnueli for his encouragement, Alex Rabinovich for his comments, and Lenore Zuck for her help.

And thank you, Zohar!

References

1. R. Alur and T. Henzinger. Finitary fairness. *ACM Transactions on Programming Languages and Systems*, 20:1171–1194, 1998.
2. M. Azimi, F. Briggs, M. Cekleov, M. Khare, A. Kumar, and L. P. Looi. Scalability port: A coherent interface for shared memory multiprocessors. In *Hot Interconnects 10 — A Symposium on High Performance Interconnects*, pages 65–70, August 2002.
3. M. Ben-Ari. *Principles of Concurrent Programming*. Prentice-Hall International, 1982.
4. J. E. Burns, M. J. Fischer, P. Jackson, N. A. Lynch, and G. L. Peterson. Shared data requirements for implementation of mutual exclusion using a test-and-set primitive. In *Proc. of the International Conference on Parallel Processing*, pages 79–87, 1977.
5. E. W. Dijkstra. Cooperating sequential processes. In F. Genuys, editor, *Programming Languages*, pages 43–112. Academic Press, 1968.
6. E. A. Emerson. Temporal and modal logic. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, pages 996–1072. Elsevier Science, 1990.
7. K. Etessami and Th. Wilke. An until hierarchy and other applications of an Ehrenfeucht-Fraïssé game for temporal logic. *Information and Computation*, 160(1/2):88–108, July 2000.
8. R. Fagin and J. H. Williams. A fair carpool scheduling algorithm. *IBM J. of Research & Development*, 27(2):133–139, 1983.
9. N. Francez. *Fairness*. Texts and Monographs in Computer Science. Springer-Verlag, 1986.
10. D. Gabbay, A. Pnueli, S. Shelah, and J. Stavi. On the temporal analysis of fairness. In *Proc. Seventh Annual ACM Symp. on Prin. of Prog. Languages*, pages 163–173, 1980.
11. H. Hansson and B. Jonsson. A framework for reasoning about time and reliability. In *Proc. IEEE Real-Time Systems Symp.*, pages 102–111, 1989.
12. C. A. R. Hoare. Monitors: An operating system structuring concept. *Communications of the ACM*, 17(10):549–557, 1974.
13. D. N. Jayasimha. *Communication and Synchronization in Parallel Computation*. PhD thesis, Department of Computer Science, University of Illinois, Urbana, IL, June 1988.
14. D. N. Jayasimha. Distributed synchronizers. In *Proc. 17th Intl. Conf. on Parallel Processing*, pages 23–27., 1988.
15. D. N. Jayasimha and N. Dershowitz. Bounded fairness. CSRD Report 615, Center for Supercomputing Research and Development, Univ. of Illinois, Urbana, IL, 1986.
16. J. A. W. Kamp. *Tense Logic and the Theory of Linear Order*. PhD thesis, University of California, Los Angeles, CA, 1968.
17. R. Koymans. Specifying real-time properties with metric temporal logic. *Journal of Real-Time Systems*, 2:255–299, 1990.
18. F. Kröger. *Temporal Logic of Programs*, volume 8 of *EATCS Monographs on Theoretical Computer Science*. Springer-Verlag, Berlin, October 1987.
19. Z. Manna and A. Pnueli. Verification of concurrent programs: The temporal framework. In R. S. Boyer and J. S. Moore, editors, *The Correctness Problem in Computer Science*. Academic Press, New York, NY, 1981.
20. Z. Manna and A. Pnueli. Proving temporal properties: The temporal way. In *Proc. 10th Colloq. on Automata, Languages, and Programming*, pages 491–512, 1983.

21. Z. Manna and A. Pnueli. *The Temporal Logic of Reactive and Concurrent Systems: Specification*. Springer-Verlag, New York, 1991.
22. J. S. Ostroff. *Temporal Logic for Real Time Systems*. Research Studies Press, 1989.
23. G. L. Peterson. Myths about the mutual exclusion problem. *Information Processing Letters*, 12(3):115–116, 1981.
24. A. Pnueli. The temporal logic of programs. In *Proc. 19th Annual IEEE Symp. on Foundations of Computer Science*, pages 46–57, 1977.
25. A. Pnueli. The temporal semantics of concurrent programs. In *Symp. on the Semantics of Concurrent Computations*, volume 70 of *Lecture Notes in Computer Science*, pages 1–20, Berlin, 1979. Springer-Verlag.
26. A. Pnueli and E. Harel. Applications of temporal logic to the specification of real time systems. In *Symp. on Formal Techniques in Real-Time and Fault-Tolerant Systems*, volume 331 of *Lecture Notes in Computer Science*, pages 84–98, Berlin, 1988. Springer-Verlag.
27. P. Wolper. The tableau method for temporal logic: An overview. *Logique et Analyse*, 28:119–152, 1985.