Bounded Fairness

Nachum Dershowitz[1]
Department of Computer Science
University of Illinois
Urbana, IL 61801
nachum@cs.uiuc.edu

D. N. Jayasimha[2]
Department of Computer and Information Science
The Ohio State University
Columbus, OH 43210
jayasim@cis.ohio-state.edu

February 1993

[1]Research supported in part by the U. S. National Science Foundation under Grants CCR-90-07195 and CCR-90-24271 and by a Lady Davis fellowship at the Hebrew University of Jerusalem.

[2]Research supported in part by the U. S. National Science Foundation under Grant CCR-89-09189.

Abstract

Bounded fairness, a stronger notion than the usual fairness based on eventuality, can be used, for example, to relate the frequency of shared resource access of a particular process with regard to other processes that access the resource with mutual exclusion. We formalize bounded fairness by introducing a new binary operator into temporal logic. One main difference between this logic and explicit-time logics, one which we consider to be an advantage in many cases, is that time does not appear explicitly as a parameter.

The syntax and semantics for this new logic, kTL, are given. This logic is shown to be more powerful than temporal logic with the eventuality operator and as powerful as the logic with the *until* operator. We argue that kTL can be used to specify bounded fairness requirements in a more natural manner than is possible with *until*; in particular, we show properties that can be expressed more succinctly in kTL. We also give a procedure for testing satisfiability of kTL formulas.

As applications of bounded fairness, we specify requirements for some standard concurrent programming problems, and show, for example, that Dekker's mutual exclusion algorithm is *fair* in the conventional sense, but *not* bounded fair. We also give examples of bounded fair algorithms.

1 Introduction

Fairness means that every process gets a chance to make progress, regardless of what other processes do. The distinguishing feature of a large class of fairness notions is *eventuality*, that is, fairness is defined as a restriction on some infinite behavior according to the eventual occurrence of some events [7]. Temporal logic (with the modal operators, $\Box$ and $\Diamond$) has been used as a tool to specify and analyze such fairness properties [19, 20]. Gabbay, *et al.* [8], added the $\mathcal{U}$ (*until*) operator to formalize aspects of responsiveness (for example, the absence of unsolicited response) and fairness (for example, strict fairness).

In many applications (including real-time applications) the weak commitment of eventual occurrence may not be sufficient. Instead, for some systems, such as flight control and process control systems, it is required that time bounds on their behavior be met. It is then necessary to specify and reason about time explicitly. Many explicit-time logics have been proposed for such applications (see [21, 9, 17]). While eventual occurrence is a weak commitment, explicit mention of time is restrictive and undesirable in many situations. For example, the following is a plausible requirement: any process p_i 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_i. 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). This is stronger than the eventuality requirement, but, nonetheless, does not need the explicit mention of time. This notion of k-*bounded fairness* allows one to express a variety of fairness notions elegantly: from k=1, corresponding to the *f*irst *c*ome *f*irst *s*erved (FCFS) discipline (which may be too restrictive) to $k = \infty$, corresponding to the (totally unrestricted) eventuality concept. Note that bounded fairness, though suited to real-time applications, makes no assumption about the relative progress of processes; that is, a process' 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.

In the literature, bounded fairness has been mentioned in specific contexts or *en passant*. For example, Fagin and Williams [6] define fairness in the context of a carpool scheduling algorithm which is intuitively similar to our idea of bounded fairness. Manna and Pnueli [16] mention "bounded overtaking" which is the same as our idea of fairness. In this paper, we provide practical motivation for using this concept, and show how a rigorous temporal logic analysis can be done.

The layout is as follows: In Section 2, we give a semi-formal description of bounded fairness through the process model and show how to specify bounded fairness requirements for standard concurrent programming problems. In Section 3, we extend linear temporal logic [19] to include a new binary operator $\langle k \rangle$, which formally captures our intuitive idea of bounded fairness. The syntax and semantics for this logic, kTL, are given. In Section 4, we give some properties of kTL. In particular, we prove that this logic is more powerful than the temporal logic with the modal operators $\Box$ and $\Diamond$ and precisely as powerful as temporal logic with the *until* operator $\mathcal{U}$, and give an example showing the succinctness made possible with the new operator. In Section 5, we describe the construction of a semantic tableau for kTL which gives a decision procedure for satisfiability. In Section 6, we give applications of bounded fairness. In particular, we show that Dekker's solution to the (two process) mutual exclusion problem is *fair* in the conventional sense, but *not* k-bounded fair for any fixed value of k. In the last section of the paper, we discuss some possible extensions to this research.

2 Bounded Fairness

We work in the context of the concurrent processing of several asynchronous processes. To make our ideas concrete, we define fairness using the state transition model of Burns, *et al.* [3]. It will become obvious to the reader that bounded fairness could be defined for other abstract models of concurrent computation (for example, the model described in [5]). Our process model, then, is a set of states with a transition function. More formally, a process P_i is a triple $\langle V, X_i, p_i \rangle$ where V is a 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, T_i, C_i, and E_i, can be empty. The state transition function p_i: $V \times X_i \rightarrow V \times X_i$ has the following properties:

1. $x \in R_i,\ v \in V$ imply $p_i(v, x) \in V \times (T_i \cup C_i)$;
2. $x \in T_i,\ v \in V$ imply $p_i(v, x) \in V \times (T_i \cup C_i)$;
3. $x \in C_i,\ v \in V$ imply $p_i(v, x) \in V \times (E_i \cup R_i)$;
4. $x \in E_i,\ v \in V$ imply $p_i(v, x) \in V \times (E_i \cup R_i)$.

Note that, for convenience, we refer to a process by the region to which it currently belongs instead of referring to it by its associated triple.

The usual fairness requirement expressed in linear temporal logic [19], using the above process model, is

$$p_i \in T_i \;\supset\; \Diamond(p_i \in C_i) \tag{1}$$

This assertion states that the process, which is currently in its trying region, eventually enters its critical region.

For some applications we would want a stronger assertion than (1), namely that the entry of any process into its critical section is k-bounded fair. The parameter k is a fixed positive integer referred to informally as the *bound*. The formula $q \langle k \rangle p$ is read "p is k-bounded to q" and is true at a particular state if and only if p is true at one of the next k instances that q is true. (Note the assumption of a linearly ordered time sequence.) Define the proposition

> q: a scheduler allows a process p_i or another process arriving after p_i into the respective critical region.

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

$$p_i \in T_i \;\supset\; q \langle k \rangle (p_i \in C_i) \tag{2}$$

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

The following two examples illustrate how bounded fairness requirements may be specified for standard concurrent programming situations using the $\langle k \rangle$ operator in a more natural manner than with either *until* or *atnext* [14].

(Five) Dining Philosophers Problem: We assume that the reader is familiar with the dining philosophers problem, as originally formulated by Dijkstra [4]. 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, we allow philosopher i's neighbors to get the forks at most twice out of turn after i wishes to use the forks. Let $\textit{try-forks}_i$ and C_i be the trying and the critical regions, respectively, of the ith philosopher's process $\textit{phil}_i$. Define

q_i: the scheduler allows $phil_i$ or its neighboring processes ($phil_{(i+1)\ mod\ 5}$, $phil_{(i+4)\ mod\ 5}$) arriving after $phil_i$ to enter the respective critical region.

The specification for this bounded requirement would be

$$\forall i(1 \leq i \leq 5)\ \Box\ \Big((phil_i \in \textit{try-forks}_i)\ \supset\ q_i \diamond_3 (phil_i \in C_i)\Big)$$

A Monitor Example: A *monitor* is a language construct for process synchronization. A process has to wait on a condition variable, say B, if it finds that it should not be granted access to a particular section of the program. Let L be the label at which a process p waits. The operations defined on B are *wait* and *signal* [10]. The assertion *at* x means that the control flow is at the beginning of statement x; *after* x means that control has just completed execution of x. Define the propositions:

wait(B): the associated process waiting on the condition variable B delays.

signal(B): wake up a process waiting on the condition variable B.

The usual fairness property for p is

$$fair(L : wait(B)) :$$
$$\Box\Diamond(at\ L\ \wedge\ signal(B))\ \supset\ \Diamond(after\ L)$$

where the assertion $\Box\Diamond x$ means that x is true infinitely often.

Define the proposition

signal(p,B): signal (B) wakes process p waiting on B or wakes a process waiting on B which arrived after p.

The bounded fairness property for p is

$$\textit{bounded-fair}_k(L : wait(B)) :$$
$$\Box\Big(at\ L\ \supset\ \Big(signal(p, B) \diamond_k\ (after\ L)\Big)\Big)$$

3 Formal Definition of $\langle k \rangle$

We define now the syntax and semantics of kTL, the logic with the temporal operator $\langle k \rangle$. For convenience, we use other well known temporal operators also. Later we show that these operators can be expressed in terms of $\langle k \rangle$. Consider a language $\mathcal{L}$ with atomic sentences q_0, q_1, $\ldots$, the usual connectives $\neg$, $\wedge$, $\vee$, the truth constants T and F, and the temporal connectives $\Diamond$, $\Box$, $\bigcirc$ (the next instant operator), $\mathcal{U}$, *atnext*, and $\langle k \rangle$. A Kripke structure for $\mathcal{L}$ is given by

i) a countably infinite sequence, $W = \{\eta_0, \eta_1, \ldots\}$ of states (η_0 is the initial state).

ii) a mapping $\nu(q_i, \eta_j) \in \{\mathit{true},\ \mathit{false}\}$ with every atomic formula q_i of $\mathcal{L}$ and every $\eta_j \in W$.

The mapping ν is inductively extended to all formulas as follows:

1. $\nu(\neg q,\ \eta_j) \ = \ \mathit{true}$ iff $\nu(q,\ \eta_j) \ = \ \mathit{false}$.
2. $\nu(p \ \supset \ q,\ \eta_j) \ = \ \mathit{true}$ iff $\nu(p,\ \eta_j) \ = \ \mathit{false}$ or $\nu(q,\ \eta_j) \ = \ \mathit{true}$.
3. $\nu(p \ \vee \ q,\ \eta_j) \ = \ \mathit{true}$ iff $\nu(p,\ \eta_j) \ = \ \mathit{true}$ or $\nu(q,\ \eta_j) \ = \ \mathit{true}$.
4. $\nu(p \ \wedge \ q,\ \eta_j) \ = \ \mathit{true}$ iff $\nu(p,\ \eta_j) \ = \ \mathit{true}$ and $\nu(q,\ \eta_j) \ = \ \mathit{true}$.
5. $\nu(\Box q,\ \eta_j) \ = \ \mathit{true}$ iff $\nu(q,\ \eta_j) \ = \ \mathit{true}$ for every $i \ > \ j$.
6. $\nu(\Diamond q,\ \eta_j) \ = \ \mathit{true}$ iff $\nu(q,\ \eta_j) \ = \ \mathit{true}$ for some $i \ > \ j$.
7. $\nu(\bigcirc q,\ \eta_j) \ = \ \mathit{true}$ iff $\nu(q,\ \eta_{j+1}) \ = \ \mathit{true}$.
8. $\nu(p \ \mathcal{U} \ q,\ \eta_j) \ = \ \mathit{true}$ iff for some $k > j$
   $\nu(q,\ \eta_k) \ = \mathit{true}$ and $\nu(p,\ \eta_i) \ = \ \mathit{true}$ for all $i,\ j < i < k$.
9. $\nu(p \ \mathit{atnext} \ q,\ \eta_j) \ = \ \mathit{true}$ iff for some $k > j$
   $\nu(p \wedge q,\ \eta_k) \ = \mathit{true}$ and $\nu(q,\ \eta_i) \ = \ \mathit{false}$ for all $i,\ j < i < k$.
10. $\nu(q \langle k \rangle p,\ \eta_{m_0}) \ = \ \mathit{true}$ (where $k \in \{1, 2, 3, \ldots\}$),
    iff there is $l \leq k$ and $m_0 < m_1 < \ \cdots \ < m_l$ such that
    for all $i,\ 1 \leq i \leq l,\ \nu(q,\ \eta_{m_i}) \ = \ \mathit{true}$,
    $\nu(q,\ \eta_j) \ = \ \mathit{false}$ for all $i, j,\ 1 \leq i \leq l,\ m_{i-1} < j < m_i$,
    and $\nu(p,\ \eta_{m_l}) \ = \ \mathit{true}$.
11. $\nu(T,\ \eta_j) \ = \ \mathit{true}$.
12. $\nu(F,\ \eta_j) \ = \ \mathit{false}$.

We will sometimes refer to well-formed formulae (wffs) in the above system as kTL formulas. We will abuse notation by saying "P is true (false)" when we really mean $\nu(P) = \mathit{true}$ (*false*), for wff P.

Observation 1: By convention, the temporal operators are chosen so as *not* to include the present state. Our definitions of $\mathcal{U}$ and *atnext* are "strong" in the sense that we require the assertion q to hold in the future for the associated formulas to be true.

Observation 2: A linearly ordered time structure is implied by the assumption of a countably infinite sequence of states. Hence our structure is valid for an ω-model.

Observation 3: There is a natural correspondence between states of our process model and W. In fact, a possible sequence of computations chosen from the processes, called an *admissible computation*, forms a one-to-one correspondence with W. The set of all admissible computations is the interleaved model of parallel computation.

4 Expressiveness

Kamp [13] has shown that $\mathcal{L}(\mathcal{U})$, the language with $\mathcal{U}$ as the only temporal connective,[1] is as expressive as the first order theory of linear order. The latter is given by the set of natural numbers with equality, the binary relation $<$, and a set of unary predicates. Based on this result of expressiveness, $\mathcal{L}(\mathcal{U})$ is said to be *expressively complete.*

We now show that $\mathcal{L}(\langle k \rangle)$ is also expressively complete by showing that

- Every wff P in $\mathcal{L}(\mathcal{U})$ can be rewritten as a wff Q in $\mathcal{L}(\langle k \rangle)$ such that, for every structure , and for every state in , , Q is true at state η_i in , if and only if P is true at η_i in , .
- Every wff P in $\mathcal{L}(\langle k \rangle)$ can be rewritten as a wff Q in $\mathcal{L}(\mathcal{U})$ such that, for every structure , and for every state in , , Q is true at state η_i in , if and only if P is true at η_i in , .

Theorem 1 *$\mathcal{L}(\langle k \rangle)$ is at least as powerful as $\mathcal{L}(\mathcal{U})$.*

[1] It is understood that the language has atomic sentences and the usual non-temporal connectives. All languages mentioned in this paper are interpreted in Kripke structures mentioned in the previous section.

Proof: This follows from the fact that the temporal connective $\mathcal{U}$ can be expressed in terms of $\langle k \rangle$, as follows:

$$p\,\mathcal{U} q \;\equiv\; (p \supset q) \langle 1 \rangle q \tag{3}$$

Consider the left-hand side of this identity: its truth in state η_i means that q is true in some state η_j $(j > i)$, and p and $\neg q$ hold in each state of the (possibly empty) sequence $\eta_{i+1}, \ldots, \eta_{j-1}$. Similarly, the right-hand side, $(p \supset q) \langle 1 \rangle q$, is true in state η_i when q is true at some state η_j $(j > i)$ at which time $p \supset q$ must also be true; at all other states in the sequence $\eta_{i+1}, \ldots, \eta_{j-1}$, $p \supset q$ is false, that is, p is true and q is false. □

Other temporal connectives can be expressed succinctly in terms of $\langle k \rangle$. We state the following results without proof. (Their proofs are straightforward using the mappings given in the previous section.)

$$\begin{aligned}
\Diamond p &\equiv p \langle 1 \rangle T \\
\Box p &\equiv \neg \left(\neg p \langle 1 \rangle T \right) \\
\bigcirc p &\equiv T \langle 1 \rangle p \\
p \ \mathit{atnext}\ q &\equiv q \langle 1 \rangle p
\end{aligned}$$

Since $\mathcal{L}(\mathcal{U})$ is more powerful than $\mathcal{L}(\Diamond)$ or $\mathcal{L}(\Box))$ [8], we conclude from Theorem 1 that $\mathcal{L}(\langle k \rangle)$ is more powerful than $\mathcal{L}(\Box)$.

Theorem 2 *$\mathcal{L}(\mathcal{U})$ is at least as powerful as $\mathcal{L}(\langle k \rangle)$.*

Proof: We show that $\langle k \rangle$ can be expressed in terms of $\mathcal{U}$ in the following inductive manner:

$$q \langle 1 \rangle p \;\equiv\; \neg q \ \mathcal{U}\ (p \wedge q) \tag{4}$$

$$q \langle k \rangle p \;\equiv\; q \langle 1 \rangle p \vee \left(\neg q \ \mathcal{U}\ q \langle k-1 \rangle p \right) \ \text{for } k > 1, \tag{5}$$

The correctness of the base case, $k = 1$, is fairly obvious (refer to the mappings in Section 4). For the inductive case, consider first what it means for $q \langle k \rangle p$ to be true at state m_0: There are $1 \le l \le k$ subsequent states $m_1 < m_2 < \cdots < m_l$ at which q is true, but between which (for all times x,

$m_{i-1} \leq x \leq m_i$, $1 \leq i \leq l$) q is false, and, furthermore, p is true at the last one. Comparing this with the meaning of $\neg q\ \mathcal{U}(q \langle k{-}1 \rangle p))$, namely that q is false until some state m_1 (when it may or may not be true), and is true at each of $m_2, \ldots, m_{l'}$ (for some l'), but false in between, while p is true at $m_{l'}$, we see that the left-hand side is true when p and q are true at m_1, which is covered by the disjunct $q \langle 1 \rangle p$.

On the other hand, if the first disjunct of the right-hand side is true, then so is the left-hand side. For the right-hand side to be true, when $\neg(q \langle 1 \rangle p)$, p must be false when q is first true, but true one of the next $k-1$ times it is, which also makes the left-hand side true. □

Corollary 1 *$\mathcal{L}(\langle k \rangle)$ has the same power as $\mathcal{L}(\mathcal{U})$.*

There is some similarity between the iterated *atnext* operator and $\langle k \rangle$. (The two were proposed independently in [14] and [12], respectively.) Informally, p $atnext^k$ q means, "p holds at the kth next state at which q holds" (but may hold earlier, too). It is straightforward to show that

$$q \langle k \rangle p \quad \equiv \quad \bigvee_{i=1}^{k} p \ atnext^i \ q \tag{6}$$

and

$$p \ atnext^k \ q \quad \equiv \quad q \langle 1 \rangle (p \ atnext^{k-1} \ q) \text{ for } k > 1 \tag{7}$$

Succinctness of a formalism is its ability to express properties in short formulas. The operator $\langle k \rangle$ contributes to the succinctness of expressions that would otherwise require multiple occurrences of U. In fact, a formula

$$(q_1 \wedge \cdots \wedge q_m) \langle k \rangle (p_1 \wedge \cdots \wedge p_n),$$

where the q_i and p_j are distinct propositional variables, requires k uses of U, *each* referring to all of the qs and ps.

5 Satisfiability

Since propositional logic with the *until* operator is decidable, it follows from Theorems 1 and 2 that kTL is also decidable. All the same, in this section, we outline a semantic tableau method (closely related to the analytic tableau method of Smullyan [22]) to obtain a decision procedure for satisfiability that has some interest in its own right.

A kTL formula is *elementary* if it is a propositional variable, its negation, or a next time formula (one with $\bigcirc$ as the main, "outermost" connective). Any other formula is non-elementary. Using the rules of Table 1, algorithm BUILD shown in Figure 1 decomposes a formula f to construct a tableau (graph) that comprises a systematic search for an interpretation (model) of f. The number of nodes in the resultant graph for a finite length formula is finite. (F_n is the set of formulas labeling node n.)

1.	$\neg\neg p$	$\rightarrow$	$\{p\}$
2.	$p \wedge q$	$\rightarrow$	$\{p, q\}$
3.	$\neg(p \vee q)$	$\rightarrow$	$\{\neg p, \neg q\}$
4.	$\neg \bigcirc p$	$\rightarrow$	$\{\bigcirc \neg p\}$
5.	$\Box p$	$\rightarrow$	$\{\bigcirc p, \bigcirc \Box p\}$
6.	$\neg \Diamond p$	$\rightarrow$	$\{\bigcirc \neg p, \bigcirc \neg \Diamond p\}$
7.	$p \vee q$	$\rightarrow$	$\{p\}, \{q\}$
8.	$\neg p \wedge q$	$\rightarrow$	$\{\neg p\}, \{\neg q\}$
9.	$\Diamond p$	$\rightarrow$	$\{\bigcirc p\}, \{\bigcirc \Diamond p\}$
10.	$\neg \Box p$	$\rightarrow$	$\{\bigcirc \neg p\}, \{\bigcirc \neg \Box p\}$
11.	$p \ \mathcal{U}\ q$	$\rightarrow$	$\{\bigcirc q\}, \{\bigcirc p, \bigcirc(p \ \mathcal{U}\ q)\}$
12.	$\neg(p \ \mathcal{U}\ q)$	$\rightarrow$	$\{\bigcirc \neg p, \bigcirc \neg q\}, \{\bigcirc \neg q, \bigcirc p, \bigcirc \neg (p\, \mathcal{U} q)\}$
13.	$q \Diamond_{1} p$	$\rightarrow$	$\{\bigcirc p, \bigcirc q\}, \{\bigcirc \neg q, \bigcirc \left(q \Diamond_{1} p\right)\}$
14.	$\neg \left(q \Diamond_{1} p\right)$	$\rightarrow$	$\{\bigcirc \neg p, \bigcirc q\}, \{\bigcirc \neg q, \bigcirc \neg \left(q \Diamond_{1} p\right)\}$
15.	$q \Diamond_{k} p$	$\rightarrow$	$\{\bigcirc p, \bigcirc q\}, \{\bigcirc \neg p, \bigcirc q, \bigcirc \left(q \Diamond_{k-1} p\right)\},$ $\{\bigcirc \neg q, \bigcirc \left(q \Diamond_{k} p\right)\}$
16.	$\neg \left(q \Diamond_{k} p\right)$	$\rightarrow$	$\{\bigcirc \neg p, \bigcirc q, \bigcirc \neg \left(q \Diamond_{k-1} p\right)\},$ $\{\bigcirc \neg q, \bigcirc \neg \left(q \Diamond_{k} p\right)\}$

Table 1: Tableau rules

Algorithm BUILD

1. Let $F_r = \{f\}$, where r is the initial node.
2. A decomposed formula f is marked f^* to avoid its repeated decomposition. The next two steps are repeatedly applied.
3. If F_n contains an unmarked, non-elementary formula g whose decomposition rule yields S_1, S_2, $\ldots$, S_m, then, for every S_i, form $n_i = (F_n - \{g\}) \cup \{S_1, \ldots, S_m\} \cup \{g^*\}$. If the graph already has a node labeled n_i, form an edge from n to this node. Otherwise, create a new node labeled n_i. Thus, every non-elementary formula that has not been decomposed is expanded using the decomposition rules.
4. Otherwise (when F_n has elementary or marked formulas only), remove the outermost $\bigcirc$ from *all* the nexttime formulas and create edges to existing nodes whose outermost $\bigcirc$ has been removed, or else (if such a node does not exist) create a new node labeled by only those formulas whose outermost $\bigcirc$ has been removed.

Figure 1: Algorithm to construct the semantic tableau for a kTL formula.

Algorithm REMOVE of Figure 2 checks for satisfiability, using the graph produced by BUILD. It removes nodes that are unsatisfiable from the tableau. Following Ben-Ari, *et al.* [2], we call a node containing only elementary or marked formulas a *state*. An *eventuality formula* is a temporal formula whose tableau decomposition yields two or more subformulas, one of which contains itself as a subformula (rules 9–16 in Table 1). In a graph containing these formulas, it is possible that the eventualities are never satisfied by some paths which indefinitely postponed its evaluation. In rules 9–12 of Table 1, let its *finite* subformula be the first set in the decomposition. To check satisfiability of eventuality formulas, we define *finite-term node(s)* for each eventuality formula. An eventuality formula given in rules 9–12 is said to have a finite-term node if there exists a path from the node labeled by this formula to a state labeled by its finite subformula. For formulas involving $\langle k \rangle$ (rules 13–16), the finite term node definitions are more involved. A formula $q \langle k \rangle p$ is said to have finite-term nodes if there exists a path from a node labeled by this formula that includes i $(1 \leq i \leq k)$

Algorithm REMOVE

1. Remove any node containing a formula and its negation.

2. Remove any node, all successors of which have been removed.

3. Remove the child of any state, or the initial state, if it contains an eventuality formula not having finite-term nodes.

Figure 2: Algorithm to remove unsatisfiable nodes

---

distinct states, each of which is labeled $\{\bigcirc p, \bigcirc q\}$. (The need for this seemingly complicated definition can be understood by building a tableau for the unsatisfiable formula $\bigcirc(\neg p \wedge q) \wedge (q \langle 1 \rangle p)$.) A formula $\neg(q \langle k \rangle p)$ is said to have finite-term nodes if there exists a path from a node labeled by this formula that includes exactly k *distinct* states, each of which is labeled $\{\bigcirc(\neg p), \bigcirc q\}$.

Theorem 3 (Satisfiability) *Let G be the graph resulting from applying algorithms BUILD and REMOVE to a formula f. Then f is satisfiable if and only if the initial node is in G.*

This can be proved using the techniques suggested by Smullyan [22] and Wolper [23].

6 Applications

In [11], the second author introduced a new synchronization primitive called the *Distributed Synchronizer*, which is particularly suited for implementation on large shared memory multiprocessors. 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 $(2n - (1 + \log_2 n))$-bounded fair.

In this section, we show that Dekker's solution to the two process mutual exclusion problem [4, 15] is not k-bounded fair for *any* fixed value of k. A rigorous proof is tedious; an informal proof follows the algorithm:

I: $t := 1$; $y_1 := y_2 := \mathit{false}$;

Process 1	Process 2
l_0: execute	m_0: execute
l_1: $y_1 := \mathit{true}$	m_1: $y_2 := \mathit{true}$
l_2: if $y_2 = \mathit{false}$ then goto l_7	m_2: if $y_1 = \mathit{false}$ then goto m_7
l_3: if $t = 1$ then goto l_2	m_3: if $t = 2$ then goto m_2
l_4: $y_1 := \mathit{false}$	m_4: $y_2 := \mathit{false}$
l_5: loop until $t = 1$	m_5: loop until $t = 2$
l_6: goto l_1	m_6: goto m_1
l_7: $t := 2$	m_7: $t := 1$
l_8: $y_1 := \mathit{false}$	m_8: $y_2 := \mathit{false}$
l_9: goto l_0	m_9: goto m_0

According to the process model of Section 2, $T_i = \{x_1, \ldots, x_6\}$; $C_i = \{x_7, x_8\}$; $E_i = \{x_9\}$; $R_1 = \{x_0\}$. For $i = 1$, x is l, and x is m for $i = 2$.

For process p_1, the k-bounded fairness requirements are:

$$a)\ ((p_1 \in T_1) \wedge (p_2 \in T_2)) \supset \Big((\mathit{after}\ l_2) \vee (\mathit{after}\ m_2) \ \Diamond_{k+1}\ \mathit{after}\ l_2\Big)$$

$$b)\ ((p_1 \in T_1) \wedge (p_2 \notin T_2)) \supset \Big((\mathit{after}\ l_2) \vee (\mathit{after}\ m_2) \ \Diamond_{k}\ \mathit{after}\ l_2\Big)$$

Assume that the processor executing Process 1 is at l_5 and that it takes a long time to execute l_5 (we make no assumptions about the relative speeds of processes). Meanwhile, the processor executing Process 2 executes m_7, $\ldots$, m_1, and is at m_2. It still finds that $y_1 = \mathit{false}$, since l_5 and l_6 are not complete. 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-bounded fair for *any* fixed value of k. It is fair, since Process 1 will eventually be able to enter its critical region (an instruction takes only a finite amount of time to execute).

In the examples that we have chosen, the bound is $(2n - (1 + \log_2 n))$ for *DSP* and *DSV*, and is unbounded for Dekker's algorithm. We have investigated other concurrent algorithms for bounded fairness. Peterson's two process mutual exclusion algorithm [18] is 2-bounded fair since it permits a process executing in its trying region to be overtaken at most once. The solution to the dining philosophers problem utilizing monitors presented in Ben-Ari [1] is 1-bounded fair.

7 Discussion

Most fairness properties involve the temporal concept "eventually." Eventuality, however, is a weak concept with which to specify and prove properties of many real-time concurrent programs. We have introduced a stronger notion of fairness called *k-bounded fairness*. The definition is elegant because time is not explicit, though the idea of bounded fairness would seem to require it. We have formalized this notion by introducing a new binary operator $\langle k \rangle$ into the linear temporal logic with the operators $\Box$ and $\Diamond$. With the new operator we are able to express a variety of fairness notions from strict *fifo* fairness to eventuality. We have shown that the extended temporal logic is quite powerful. We have used it to specify a few standard concurrent programming problems and to study mutual exclusion algorithms for bounded fairness.

This work can be extended in various ways:

- A number of related operators merit investigation:
  - There is a dual notion,

    $$q \boxed{k} p \quad \equiv \quad \neg(q \langle k \rangle \neg p)$$

    (p is true each of the next k instants q is).
  - The assertion, "p is true the kth time q is, *but not before*," denoted $q \langle =k \rangle p$, can be expressed in terms of the $\langle k \rangle$ operator as follows:

    $$q \langle =1 \rangle p \;\equiv\; q \langle 1 \rangle p$$
    $$q \langle =k \rangle p \;\equiv\; \neg\left(q \langle k-1 \rangle p\right) \wedge \left(q \langle k \rangle p\right) \text{ for } k > 1.$$

  We have now a spectrum of related operators: $q \langle k \rangle p$, p $atnext^k$ q, $q \langle =k \rangle p$, and $q \boxed{k} p$. Each refers to the next k times q is true. The first asks only that p be true at one of them; the second demands that it be true at the kth occurrence specifically; the third adds that it not be true before; the last requires p to be true at all of them. It might be interesting to see what the natural applications for these various operators are.

- kTL incorporates only the future fragment of time. There is a natural extension to the past by allowing k to be a negative integer. Though inclusion of the past would not add expressive power to the logic, perhaps the extended logic would allow simpler specifications, simpler proofs of past program behavior, or the like.

- Another possible extension is to allow quantification of k. We might consider a restricted logic in which k is quantified but the non-temporal variables are not. Is the restricted logic, with k quantified, decidable?

Acknowledgments

We thank Tim Carlson and Neelam Soundararajan for useful discussions on this topic and Lenore Zuck for helping with the proof of Theorem 2.

References

[1] M. Ben-Ari. *Principles of Concurrent Programming.* Prentice-Hall International, 1982.

[2] M. Ben-Ari, A. Pnueli, and Z. Manna. The temporal logic of branching time. *Acta Informatica*, 20:207–226, 1983.

[3] 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.

[4] E. W. Dijkstra. Cooperating sequential processes. In F. Genuys, editor, *Programming Languages*, pages 43–112. Academic Press, 1968.

[5] E. A. Emerson. Temporal and modal logic. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, pages 996–1072. Elsevier Science, 1990.

[6] R. Fagin and J. H. Williams. A fair carpool scheduling algorithm. *IBM J. of Research & Development*, 27(2):133–139, 1983.

[7] N. Francez. *Fairness.* Texts and Monographs in Computer Science. Springer-Verlag, 1986.

[8] 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.

[9] H. Hansson and B. Jonsson. A framework for reasoning about time and reliability. In *Proc. IEEE Real-Time Systems Symp.*, pages 102–111, 1989.

[10] C. A. R. Hoare. Monitors: An operating system structuring concept. *Communications of the ACM*, 17(10):549–557, 1974.

[11] D. N. Jayasimha. Distributed synchronizers. In *Proc. 17th Intl. Conf. on Parallel Processing*, pages 23–27., 1988.

[12] D. N. Jayasimha and N. Dershowitz. Bounded fairness. CSRD Rpt. 615, Ctr. for Supercomputing Res. and Dev., Univ. of Illinois, Urbana, IL, 1986.

[13] J. A. W. Kamp. *Tense Logic and the Theory of Linear Order*. PhD thesis, University of California, Los Angeles, CA, 1968.

[14] F. Kroger. *Temporal Logic of Programs*. Springer Verlag, 1987.

[15] 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.

[16] Z. Manna and A. Pnueli. Proving temporal properties: The temporal way. In *Proc. 10th Colloq. on Automata, Languages, and Programming*, pages 491–512, 1983.

[17] J. S. Ostroff. *Temporal Logic for Real Time Systems*. Research Studies Press, 1989.

[18] G. L. Peterson. Myths about the mutual exclusion problem. *Information Processing Letters*, 12(3):115–116, 1981.

[19] A. Pnueli. The temporal logic of programs. In *Proc. 19th Annual IEEE Symp. on Foundations of Computer Science*, pages 46–57, 1977.

[20] A. Pnueli. The temporal semantics of concurrent programs. *Symp. on the Semantics of Concurrent Computations*, pages 1–20, 1979. LNCS Vol. 70, Springer Verlag, Berlin.

[21] 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*, pages 84–98, 1988. LNCS Vol. 331, Springer Verlag, Berlin.

[22] R. Smullyan. *First Order Logic*. Springer Verlag, 1971.

[23] P. Wolper. Temporal logic can be more expressive. *Information and Control*, 56(1–2):72–98, 1983.

Footnotes

1. It is implicitly understood that the language has atomic sentences and the usual non-temporal connectives. All languages mentioned in this paper are interpreted in Kripke structures mentioned in the previous section.