PROBLEMS CONCERNING FAIRNESS AND TEMPORAL LOGIC FOR CONFLICT-FREE PETRI NETS

Rodney R. Howell and Louis E. Rosier

Department of Computer Sciences
The University of Texas at Austin
Austin, Texas 78712-1188

TR-87-19 May 1987

Problems Concerning Fairness and Temporal Logic for Conflict-Free Petri Nets[1]

Rodney R. Howell and Louis E. Rosier

Department of Computer Sciences
University of Texas at Austin
Austin, Texas 78712-1188

Abstract

In this paper, we examine the complexity of the fair nontermination problem for conflict-free Petri nets under several definitions of fairness. For each definition of fairness, we are able to show the problem to be complete for either NP, PTIME, or NLOGSPACE. We then address the question of whether these results extend to the more general model checking problem with respect to the temporal logic for Petri nets introduced by Suzuki. Since many of the model checking problems concerning finite state systems can be reduced to a version of the fair nontermination problem, it would seem plausible that the model checking problem for conflict-free Petri nets would be decidable. However, it turns out that unless the logic is severely restricted, model checking is undecidable for conflict-free Petri nets. In particular, the problem is undecidable even when formulas are of the form **Gf** ("invariantly f") where f contains no temporal logic operators. On the other hand, we show that model checking for conflict-free Petri nets is NP-complete for $\widetilde{L}(\mathbf{F},\mathbf{X})$ -- the logic restricted to the operators **F** (eventually), **X** (next time), $\wedge$, and $\vee$, with negations allowed only on the predicates.

1. Introduction

For some time now, temporal logic has been considered an appropriate formalism for reasoning about systems of concurrent programs [28, 36]. Research in this area seems to emphasize two directions. The first concentrates on the proof-theoretic paradigm of manual program verification [33], while the second concerns itself with algorithmically solving special cases, such as the case where the system is finite state [9, 12, 26, 37, 38, 39, 40, 44]. The latter strategy usually involves viewing the global state transition graph of a finite state concurrent system as a finite structure over which temporal logic formulas are interpreted. For many temporal logics there exist efficient "model checking" algorithms for determining if a given structure defines a model of a correctness specification expressed in the temporal logic. (See, e.g., [9, 26, 40, 44].) An automata-theoretic viewpoint is often used. There, the model checking problem for finite state programs is recast in terms of testing emptiness for an ω-automaton; i.e., the global state transition graph is viewed as a finite state automaton which accepts an infinite string iff it corresponds to a computation of the concurrent system. To check that some computation of the system meets the specification f, one checks that the automaton with acceptance condition f accepts some input. This

[1]This work was supported in part by U.S. Office of Naval Research Grant No. N00014-86-0763. A summary of the results will be presented at the *8th European Workshop on Applications and Theory of Petri Nets*.

problem, which we refer to as model checking, is actually a restricted version of the model checking problem examined in [9], and is referred to as "determination of truth in a structure" in [40].

A problem closely related to model checking is the fair nontermination problem. For a concurrent system a fairness constraint is a property that is either true or false of an infinite computation. For a given fairness constraint, the fair nontermination problem is to determine if an infinite computation exists which satisfies the fairness constraint. For many of the model checking problems concerning finite state systems, model checking can be reduced to a version of the fair nontermination problem. See, e.g., [9, 26, 44].

An obvious limitation of the above strategy is that it only applies to finite state systems. One reason for this is that model checking is, in general, undecidable for infinite state systems [1, 3, 5, 41]. The work of [7, 35], however, seems to suggest that there are cases where model checking can be done for certain types of infinite state systems.

Petri nets constitute a powerful automata theoretic formalism that is often employed to model concurrent systems [34]. Although the formalism can not be utilized to model (arbitrary) Turing machines, it can readily model many infinite state systems. Furthermore, many decision problems with respect to Petri nets are known to be decidable. See, e.g., [20, 22, 29]. As a result, one might hope that a reasonably expressive temporal logic could be designed around this formalism such that model checking would be decidable. Now certain versions of fairness were defined (or adapted) for Petri nets in [2, 3, 5, 6, 37]. Decidability issues were considered in [3, 5, 43]. Results here include the fact that for two definitions of fairness, the fair nontermination problem is decidable. This might seem to suggest that model checking with respect to Petri nets is decidable for some reasonably expressive temporal logic. However, Carstensen [5] shows that for a stronger definition of fairness, the fair nontermination problem is undecidable. This latter result virtually assures us that even for very restricted temporal logics, model checking with respect to general Petri nets is destined to be undecidable. For example, a rather modest temporal logic for Petri nets was described by Suzuki in [41] for which model checking is shown to be undecidable.

For model checking to be decidable with respect to Petri nets, we must lower our expectations. The best one could hope for would be for model checking to be decidable for very powerful subclasses of Petri nets. During the last decade or so, many subclasses of Petri nets have been studied. In many cases, decision problems can be solved more efficiently for the restricted classes than for arbitrary Petri nets (see, e.g., [8, 10, 11, 13, 14, 15, 16, 20, 24, 30, 32, 42, 45]). One of the simplest such classes is that of conflict-free Petri nets [8, 16, 15, 24]. (Conflict-free Petri nets are equivalent to the controls of decision-free flow-chart schemata studied in [20].)

In this paper, we examine various fair nontermination problems for conflict-free Petri nets. For the notions of fairness considered in [2, 3, 5, 6, 23, 25, 37, 43], we are able to show that the nontermination problems are all decidable. In fact, we provide a much stronger result by establishing the nontermination problem for each of these definitions of fairness to be complete for NLOGSPACE, PTIME, or NP. Now since a number of these problems have efficient solutions, one might conjecture efficient solutions also exist for the more general model checking problem. So, we start by defining a simple version of the linear-time temporal logic utilized by Suzuki in [41]. The temporal operators include **F** (eventually), **G** (invariantly), **X** (next time), and **U** (until). We then examine the model checking problem for the various logics that result from restricting the use of temporal operators in a similar fashion as was done by Sistla and Clarke in [40]. Although we expected each of these problems to be decidable, we found that most were not. Specifically, we found that for formulas of the form **G**f, where f contains no temporal logic operators, the problem is undecidable. This result immediately implies that the problem is undecidable for arbitrary formulas using only the **F** temporal logic operator. (This is analogous to the subset of temporal logic called L(**F**) in [40].) Furthermore, it is not hard to extend our proof to safety properties or any of three types of liveness properties defined in [27, 46]. The major difficulty in model-checking seems to be that the simplest temporal logics are so expressive that unleashing their power on even the most simple infinite state computing structures enables one to construct temporal logic formulas that can only be satisfied by the structure emulating computations of much more powerful automata. However, we are able to show that for $\widetilde{L}$(**F**,**X**), the logic comprised of only the operators **F**, **X**, $\wedge$, and $\vee$, with negations allowed only on the predicates (see also [40]), the problem is NP-complete. Thus, the main contribution of this paper is to show that unless a very restricted subset of temporal logic is used, model checking is undecidable even for conflict-free Petri nets, one of the simplest classes of Petri nets.

The remaining portion of the paper is organized as follows. In Section 2, we define the formalisms we will be using. Throughout the paper, we will use the notion of a vector replacement system, which is simply a succinct notational variant of the Petri net formalism [21]. In Section 3, we give completeness results for the liveness problem and the various fair nontermination problems. Finally, in Section 4, we give our results concerning model checking.

2. Definitions

Let Z (N, R) denote the set of integers (nonnegative integers, rational numbers, respectively), and let Z^k (N^k, R^k) be the set of vectors of k integers (nonnegative integers, rational numbers), and $Z^{k\times m}$ ($N^{k\times m}$, $R^{k\times m}$) be the set of $k\times m$ matrices of integers (nonnegative integers, rational numbers). For a vector $v \in Z^k$, let v(i), $1 \leq i \leq k$, denote the i^{th} component of v. For a matrix $V \in Z^{k\times m}$, let V(i,j), $1 \leq i \leq k$, $1 \leq j \leq m$, denote the element in the i^{th} row and j^{th} column of V, and let v_j denote the j^{th} column of V. For a given value of k, let **0** in Z^k denote the vector of k zeros (i.e., **0**(i)=0 for i=1,...,k). Now given vectors u,v, and w in Z^k we say:

- $v=w$ iff $v(i)=w(i)$ for $i=1,...,k$;
- $v \geq w$ iff $v(i) \geq w(i)$ for $i=1,...,k$;
- $v>w$ iff $v \geq w$ and $v \neq w$; and
- $u=v+w$ iff $u(i)=v(i)+w(i)$ for $i=1,...,k$.

A $k \times m$ *vector replacement system* (VRS), is a triple (v_0,U,V), where $v_0 \in N^k$, $U \in N^{k \times m}$, and $V \in Z^{k \times m}$, such that for any i,j, $1 \leq i \leq k$, $1 \leq j \leq m$, $U(i,j)+V(i,j) \geq 0$. v_0 is known as the *start vector*, U is known as the *check matrix*, and V is known as the *addition matrix*. A column u_j of U is called a *check vector*, and a column v_j of V is called an *addition rule*. For any $x \in N^k$, we say addition rule v_j is *enabled* at x iff $x \geq u_j$. A sequence $<y_1,...,y_n>$ of rules in V is *enabled* at a vector x iff for each j, $1 \leq j \leq n$, y_j is enabled at $x+y_1+ \cdots +y_{j-1}$. If a sequence θ is enabled at v_0, then we say that θ is a *valid sequence* in (v_0,U,V). The *reachability set* of the VRS $\gamma=(v_0,U,V)$, denoted by $R(v_0,U,V)$ (or $R(\gamma)$), is the set of all vectors z, such that $z=v_0+y_1+ \cdots +y_n$ for some $n \geq 0$, where each y_j $(1 \leq j \leq n)$ is a column of V, and $<y_1,...,y_n>$ is enabled at v_0. Let $\sigma=<w_0,...,w_t>$ be a sequence of vectors in N^k. If $w_0=v_0$, and for every r, $1 \leq r \leq t$, there is a j such that $w_r=w_{r-1}+v_j$ and $w_{r-1} \geq u_j$, then we say $<w_0,...,w_t>$ is a *path* in (v_0,U,V). If there exist r and s, $0 \leq r < s \leq t$, such that $w_r \leq w_s$, then we say that $\pi=<w_r,...,w_s>$ is a *loop*. Let Ψ denote the *Parikh mapping*, such that if θ is a sequence of rules in V, then $\Psi(\theta) \in N^m$, and $\Psi(\theta)(j)$ is the number of occurrences of v_j in θ. Let $\delta(\theta)$ denote the displacement of θ.

A VRS (v_0,U,V) is said to be *conflict-free* iff (1) no number in U is greater than 1; (2) no number in V is less than -1; (3) no row in V has more than one -1; and (4) if $V(i,j)=-1$, then $U(i,j) = 1$, and all other elements in row i of U are 0. Conflict-freedom guarantees that whenever any two rules v_j and $v_{j'}$ are enabled at a vector v, v_j is also enabled at $v+v_{j'}$. (Note that this must hold even when v is not in $R(v_0,U,V)$.) For a given $k \times m$ addition matrix V, the *minimal check matrix* is a $k \times m$ matrix U in which $U(i,j)=1$ if $V(i,j)=-1$, and $U(i,j)=0$ otherwise. It is easy to see that the set of $k \times m$ conflict-free VRSs with minimal check matrices is equivalent to the set of $k \times m$ conflict-free VASs (see [8]). Furthermore, there is an obvious translation from a conflict-free Petri net (see [24]) with k places and m transitions to a $k \times m$ conflict-free VRS whose addition rules and check vectors have no elements larger than 1. Thus, our definition is general enough to include both previous definitions. In addition, all lower bounds shown in this paper are shown using VRSs having minimal check matrices and no elements larger than 1. Thus, all of our completeness and undecidability results hold for conflict-free VRSs, conflict-free VASs, and conflict-free Petri nets.

An addition rule $v_j \in V$ is said to be *live* in (v_0,U,V) if for any $w \in R(v_0,U,V)$, there is a path σ in (w,U,V) that enables v_j. The *transition liveness problem* for VRSs is to determine, for a given VRS (v_0,U,V) and an addition rule $v_j \in V$, whether v_j is live in (v_0,U,V). The VRS (v_0,U,V) is said to be *live* if

every transition $v_j \in V$ is live in (v_0,U,V). The *liveness problem* for VRSs is to determine whether a given VRS is live. A VRS is said to be *bounded* (*unbounded*) if its reachability set is finite (infinite). The *boundedness problem* for VRSs is to determine whether a given VRS is bounded.

Several of the problems studied in this paper have to do with various notions of fairness. The first three types of fairness we consider were introduced by Lehman, Pnueli, and Stavi [25]. Let σ be an infinite path in (v_0,U,V). σ is said to be *impartial* if every addition rule $v_j \in V$ is executed infinitely often. σ is said to be *just* if every addition rule $v_j \in V$ that is enabled continuously after some point in σ is executed infinitely often. σ is said to be *fair* if every addition rule $v_j \in V$ that is enabled infinitely often in σ is executed infinitely often. (Note that in a conflict-free VRS, an enabled rule remains enabled until it is executed; hence, justice and fairness are equivalent for conflict-free VRSs.) The remaining definitions of fairness come from Landweber [23] and Carstensen and Valk [6]. These definitions concern reachable markings rather than addition rules, and are interesting because they yield different complexity results than the definitions concerned with addition rules. Let $\mathcal{A}$ be a finite set of finite nonempty subsets of N^k. σ is said to be

- *1-fair* for $\mathcal{A}$ if there is an $A \in \mathcal{A}$ such that some vector reached by σ is in A.
- *1'-fair* for $\mathcal{A}$ if there is an $A \in \mathcal{A}$ such that every vector reached by σ is in A.
- *2-fair* for $\mathcal{A}$ if there is an $A \in \mathcal{A}$ such that some vector reached infinitely often by σ is in A.
- *2'-fair* for $\mathcal{A}$ if there is an $A \in \mathcal{A}$ such that every vector reached infinitely often by σ is in A.
- *3-fair* for $\mathcal{A}$ if the set of vectors reached infinitely often by σ is an element of $\mathcal{A}$.
- *3'-fair* for $\mathcal{A}$ if there is an $A \in \mathcal{A}$ such that every vector in A is reached infinitely often by σ.

We refer to these six types of fairness collectively as *i-fairness*, where i is understood to be an element of $\{1,1',2,2',3,3'\}$. The *impartial (just, fair, i-fair) nontermination problem* is the problem of determining whether there is an infinite impartial (just, fair, i-fair, respectively) path in a given VRS for a given set $\mathcal{A}$ (if applicable). (Many other definitions of fairness have been proposed. See, e.g., [2, 5, 6, 37, 43]. Although we do not formally discuss all of these types, we do, in the text, mention how our results can be extended to encompass them.)

Other problems that we examine in this paper have to do with temporal logic. Formulas in temporal logic are formed from *predicates*, *Boolean connectives*, and *temporal operators*. The predicates we use are ge(i,c), en(j), and fi(j), where $c,i,j \in N$. Intuitively, if θ is a given sequence of rules enabled at a given vector w, ge(i,c) means that $w(i) \geq c$, en(j) means that v_j is enabled at w, and fi(j) means that v_j is the first rule used in θ. One might wonder why we need ge(i,c) in addition to en(j). The reason for this is that although for arbitrary VRSs we can express ge(i,c) using en(j), doing so may destroy conflict-freedom. For example, we cannot simulate ge(i,c) for $c>1$ using en(j), because the check matrix in a

conflict-free VRS can contain no numbers larger than 1. The Boolean connectives we use are $\neg$, $\wedge$, $\vee$, and $\supset$, and the temporal operators are **X**, **U**, **F**, and **G**. A *well-formed formula* is either a predicate or of the form $\neg$f, f$\wedge$g, **X**f, or f **U** g where f and g are well-formed formulas. We also use the following abbreviations:

- f$\vee$g $\equiv$ $\neg(\neg f \wedge \neg g)$;
- f$\supset$g $\equiv$ $\neg f \vee g$;
- **F**f $\equiv$ true **U** f; and
- **G**f $\equiv$ $\neg$**F**$\neg$f.

Let (v_0,U,V) be a VRS, and let $\theta=<w_1,w_2,...>$ be a finite or infinite valid sequence in (v_0,U,V). The following define the semantics of our logic:

- $<\mathcal{V},\theta,n> \models ge(i,c)$ iff $v_0+\sum_{j=1}^{n} w_j(i) \geq c$;
- $<\mathcal{V},\theta,n> \models en(j)$ iff $v_0+\sum_{j=1}^{n} w_j \geq u_j$;
- $<\mathcal{V},\theta,n> \models fi(j)$ iff $w_{n+1}=v_j$;
- $<\mathcal{V},\theta,n> \models f\wedge g$ iff $<\mathcal{V},\theta,n> \models f$ and $<\mathcal{V},\theta,n> \models g$;
- $<\mathcal{V},\theta,n> \models \neg f$ iff not $(<\mathcal{V},\theta,n> \models f)$;
- $<\mathcal{V},\theta,n> \models$ **X**f iff $<\mathcal{V},\theta,n+1> \models f$;
- $<\mathcal{V},\theta,n> \models f$ **U** g iff $\exists s>n$ such that $<\mathcal{V},\theta,s> \models g$ and $\forall t$, $n<t<s$, $<\mathcal{V},\theta,t> \models f$.

Note that the above semantics defines a *linear-time* temporal logic. Also note that the present is not considered to be a part of the future. We define the *length* of a temporal logic (TL) formula as the sum of the number of predicates, Boolean connectives, and temporal operators in the formula. We say that a TL formula f is *satisfiable* if there is an infinite valid sequence $\theta=<w_1,w_2,...>$ in some VRS $\mathcal{V}=(v_0,U,V)$ such that $<\mathcal{V},\theta,0> \models f$. We then say that $\mathcal{V}$ is a *model* for f. The *model checking problem* is the problem of determining whether a given VRS $\mathcal{V}$ is a model for a given TL formula f.

3. Liveness and Fairness

The first problems we would like to examine are the liveness and transition liveness problems for conflict-free VRSs. In [16] we gave an $O(n^{1.5})$ algorithm for determining boundedness for a conflict-free VRS, where n is the number of bits needed to encode the VRS. One portion of this algorithm was devoted to determining which addition rules could be used infinitely often. Call the set of these rules I. In what follows, we will show that the set L of all live rules is identical to I. This fact yields an $O(n^{1.5})$ upper bound for the liveness problem--the first polynomial upper bound for the problem. From this

result, we will be able to show the liveness and transition liveness problems to be PTIME-complete. We first reproduce the following lemma from [16]:

Lemma 3.1: (From [16].) For any k×m conflict-free VRS $\mathcal{V}=(v_0,U,V)$ that is described by n bits, we can construct in time $O(n^{1.5})$ a path σ in which no rule in V is used more than once, such that if some rule v_r is not used in σ, then there is no path in which v_r is used.

Proof: We construct σ as follows. First, we execute all rules enabled at v_0. Then we repeatedly cycle through U (or V), executing all those rules which are enabled but have not yet been executed. We continue until a complete pass is made through U, during which no position increases in value. (Note that this is a sufficient condition to conclude that no new rules are enabled.) Clearly, no more than m+1 passes are made through U. On each pass except the last, there is at least one rule (say v_j) enabled that was not enabled the previous pass; i.e., some position (say p) which was zero the previous pass is now positive. Furthermore, since $\mathcal{V}$ is conflict-free, if some rule subtracts from position p, that rule must be v_j. Therefore, position p must have never previously been positive. Thus, on each pass except the last some position becomes positive for the first time, so the number of passes is no more than $\min(k,m)+1=O(n^{0.5})$. Therefore, the entire procedure operates in time $O(n^{1.5})$.

Now suppose there is a path σ' using rules not in σ. Let v_r be the first such rule executed in σ'. Then all rules used before v_r in σ' are used in σ. Since v_r is not executed in σ, no position from which v_r subtracts ever decreases in value in σ; hence, these positions are at least as large as they are at the point at which v_r is executed in σ'. Then v_r is enabled by σ, a contradiction. Therefore, if v_r is not used in σ, then there is no path in which v_r is used. □

Lemma 3.2: For any k×m conflict-free VRS (v_0, U, V), the set $I = \{v_j : v_j$ is used infinitely often in some path $\sigma\}$ is the same as the set L of live rules.

Proof: Clearly $L \subseteq I$. Assume $I \not\subseteq L$, and let $w \in N^k$ be such that in some path, w is the last point from which there exists for each rule in I, a path that enables that rule. Thus, there are rules $v_1 \in V$ ($u_1 \leq w$), $v_2 \in I$, and a path σ in (w,U,V) such that v_2 is used in σ, but v_2 can never be used after $w+v_1$ is reached. From Lemma 3.1, there is a path σ' enabled at w that uses v_1 and all rules in I. Note from the proof of Lemma 3.1 that without loss of generality, we can assume that any arbitrary rule enabled at w is used first in σ'; thus, we may assume v_1 is used first in σ'. Since $v_2 \in I$ but v_2 cannot be used after $w+v_1$ is reached, $v_2=v_1$. Now since $v_2 \in I$, for every position from which v_2 subtracts, there must be a rule in I that adds to that position. Since every rule in I is executed in σ', v_2 is enabled by σ'--a contradiction. Hence, $L=I$. □

Theorem 3.1: The liveness and transition liveness problems for conflict-free VRSs are PTIME-complete. Furthermore, there is an $O(n^{1.5})$ algorithm to decide each problem.

Proof: Since the set of live rules can be computed in time $O(n^{1.5})$ (see [16]), we can clearly decide both problems in time $O(n^{1.5})$. The PTIME-hard proofs are similar to Lemma 4.1 in [16]. The details are left to the reader. □

We now consider the various fair nontermination problems defined in Section 2. It is often the case that problems in verification of parallel computations can be phrased as fair nontermination problems concerning various formal models of computation. Most work of this sort to date has been concerned with finite state models (see, e.g., [9, 12, 26, 38, 39, 44]). Hence, the reason that we now examine fair nontermination problems for conflict-free VRSs is that perhaps some of the problems in verification of parallel computations can be modelled as a type of fair nontermination problem for conflict-free VRSs. It turns out that some of the fair nontermination problems we examine have efficient solutions. Therefore, it may be the case that the algorithms presented here will be useful in the verification of parallel systems. However, we will show in the next section that these results do not extend to general temporal logic formulas, as one might hope. The first three problems we will consider are the impartial, just, and fair nontermination problems.

Theorem 3.2: The impartial, just, and fair nontermination problems for conflict-free VRSs are all PTIME-complete. Furthermore, there is an $O(n^{1.5})$ algorithm to decide each problem.

Proof: We first claim that there is an infinite impartial path iff the VRS is live. If there is an infinite impartial path, the VRS is clearly live. Conversely, if the VRS is live, from Lemma 3.1, there is a path from any reachable marking which uses one occurrence of each rule; hence, there is an infinite impartial path. Thus, from Theorem 3.1, the impartial nontermination problem for conflict-free VRSs is PTIME-complete, and there is an $O(n^{1.5})$ algorithm to solve the problem.

Next, we claim that there is an infinite just (fair) path iff there is an infinite path. To see this, note that from any reachable marking, we can execute a path using every live rule. Now from Lemma 3.2, the set of live rules is exactly the set of rules which can be enabled infinitely often. Thus, from [16], there is an $O(n^{1.5})$ algorithm to decide just (fair) nontermination. The VRS constructed in Lemma 4.1 of [16] to show the boundedness problem to be PTIME-hard has an infinite path iff it is unbounded; thus, the just (fair) nontermination problem is PTIME-hard. □

Remark: Prof. Vidal-Naquet has pointed out to us yet another definition of fairness, namely, an infinite path σ is fair in (v_0,U,V) if for every rule $v_j \in \{v_j : v_j$ is live in (w,U,V) for all $w \in \sigma\}$, v_j is executed

infinitely often. See also Best [2] and Queille and Sifakis [37]. Clearly, the above result holds for this type of fairness as well. In addition, the fair nontermination problem has been shown to be decidable in [5, 43] for general Petri nets with respect to two other definitions of fairness. The first is defined in [43] as requiring a certain transition to appear infinitely often. The second is defined in [5] as requiring a group of transitions to have the finite delay property; that is, infinitely often all transitions in the group are simultaneously not enabled or some of them appear infinitely often. These results are in contrast to another result in [5] concerning our definition of fairness, but applied to general Petri nets. The corresponding fair nontermination problem was shown in [5] to be undecidable; however, it follows from the above proof that for all three of these definitions of fairness, there is an $O(n^{1.5})$ algorithm to decide fair nontermination for conflict-free VRSs. Other types of fairness were defined in [2]. The above proof works for these types as well. Finally, if we define i-fairness on addition rules rather than on reachable vectors, we have the transitional i-behavior introduced by Carstensen and Valk [6]. It also follows that these types of fair nontermination can be decided in $O(n^{1.5})$ time.

Now of the six remaining fair nontermination problems, four are NP-complete, one is PTIME-complete, and one is NLOGSPACE-complete.

Theorem 3.3: The i-fair nontermination problem for conflict-free VRSs is NP-complete for $i\in\{1,2,3,3'\}$.

Proof: The *reachability problem* for conflict-free VRSs (i.e., the problem of determining for a given $w\in N^k$ and a given VRS $\mathcal{V}$, whether $w\in R(\mathcal{V})$) was shown in [19] to be NP-hard. We therefore show that all four of the problems are NP-hard by reducing reachability to each of them. Let (v_0,U,V) be an arbitrary $k\times m$ conflict-free VRS, and let w be an arbitrary vector in N^k. Let V' (U') be V (U) with an additional column of zeros, and let $\mathcal{A}=\{\{w\}\}$. It is now easy to see that for each $i\in\{1,2,3,3'\}$, there is an i-fair path for $\mathcal{A}$ in (v_0,U',V') iff $w\in R(v_0,U,V)$. Thus, the four problems are NP-hard.

We now describe an NP algorithm for each problem. In each of these algorithms, we use a result shown in [15] that the reachability problem for conflict-free VRSs is NP-complete. Let (v_0,U,V) be a $k\times m$ conflict-free VRS, and let $\mathcal{A}$ be a finite set of finite nonempty subsets of N^k.

- 1-fair: Guess a vector w in some set $A\in\mathcal{A}$; verify that $w\in R(v_0,U,V)$, and that there is some live rule in (w,U,V).
- 2-fair: Guess a vector w in some set $A\in\mathcal{A}$, verify that $w\in R(v_0,U,V)$, then verify that for some w' reachable in one step from w, $w\in R(w',U,V)$.
- 3-fair: Guess some set $A\in\mathcal{A}$, verify that some element w of A is reachable, then guess a sequence θ of $|A|$ rules. Verify that the set of vectors reached in the execution of θ from w is A, and that $\delta(\theta)=0$.
- 3'-fair: Guess some set $A\in\mathcal{A}$, verify that the first element of A is reachable, verify that each

successive element of A is reachable from the previous element of A, and verify that the first element of A is reachable from the last element of A.

Clearly, all of the above algorithms operate in NP; therefore, all four problems are NP-complete. □

For each of the four problems discussed above, it was necessary to determine whether some given vector was reachable; thus each problem was shown to be NP-hard. On the surface, it appears that the 2′-fair nontermination problem should be NP-hard for the same reason. However, it turns out that any unbounded conflict-free VRS has an infinite 2′-fair path for any $\mathcal{A}$. Thus, to decide the 2′-fair nontermination problem, we need to be able to decide reachability with respect to bounded systems only. We will now show that this can be done in PTIME.

Lemma 3.3: The reachability problem for bounded conflict-free VRSs is in PTIME.

Proof: Let $\mathcal{V}=(v_0,U,V)$ be a bounded $k\times m$ conflict-free VRS, and let w be an arbitrary vector in N^k. We will describe an algorithm to decide whether $w\in R(\mathcal{V})$. The algorithm consists of two phases, each corresponding to finding a portion of a path to w. The initial portion of the path will use only rules that can be used at most finitely many times in any path, and the final portion of the path will use only rules that can be used infinitely many times in some path.

In the first phase, the algorithm will attempt to construct sets A and B such that set A will contain all the rules used in an initial portion of a path σ to w, and B will contain all the rules used in the remainder of σ (and possibly others). Furthermore, A and B will be disjoint. If the algorithm is successful in this phase, it will also compute a vector v which, if $w\in R(\mathcal{V})$, will represent the vector produced by the initial portion of σ.

Phase 1 proceeds as follows. First, assign to B the set of all rules executable in $\mathcal{V}$. From Lemma 3.1, this can be done in PTIME. Also, assign v_0 to v, and initialize A to the empty set. A will be a set of rules v_j that have the property that for some $n_j>0$, all paths from v_0 to w use v_j exactly n_j times. v will record $v_0+\sum_{v_j\in A} n_j*v_j$. Any rules placed in A will be removed from B. Now suppose there is some $v_j\in B$ and some i, $1\le i\le m$, such that $V(i,j)=-1$ and for all $j'\neq j$, if $v_{j'}\in B$, then $V(i,j')=0$. Clearly, if there is a path to w, $w(i)\le v(i)$, and v_j must be executed exactly $v(i)-w(i)$ times in that path. Therefore, we remove v_j from B, and if $w(i)<v(i)$, include v_j in A and add $[v(i)-w(i)]*v_j$ to v. We can now continue iterating this process as long as there is some $v_j\in B$ satisfying the condition above. If at any time there is a $v_j\in B$ and some i, $1\le i\le m$, such that $w(i)>v(i)$, $V(i,j)=-1$, and for all $j'\neq j$, if $v_{j'}\in B$, then $V(i,j')=0$, the algorithm rejects.

If the above loop terminates successfully, then for any $v_j\in B$, if $V(i,j)=-1$, there must be a $v_{j'}\in B$ such that $V(i,j')>0$. From Lemma 3.2 in [16], there is some path σ in $\mathcal{V}$ containing a loop consisting of exactly

one occurrence of each rule in B. Since $\mathcal{V}$ is bounded, this loop must have a displacement of **0**. We can therefore conclude that for any $v_j \in B$, if $V(i,j)>0$, then

1. $V(i,j)=1$;

2. there is a $v_{j'} \in B$ such that $V(i,j')=-1$; and

3. there is no $v_{j''} \in B$ such that $j'' \neq j$ and $V(i,j'')>0$.

Thus, there is no rule in B that adds to a position from which some rule in A subtracts. We can therefore delay execution of all rules in B until all rules in A have been executed. Hence, if w is reachable, there must be a path σ that first executes exactly those rules in A (possibly more than once) until it reaches v, then executes only rules from B.

We are now ready to begin Phase 2. We wish to construct a subset $B' \subseteq B$ that contains exactly those rules used in the second portion of σ. We first initialize B′ to the empty set. As we collect rules in B′, we will determine a minimum number of times they must be executed in σ; v will continue to accumulate the effect of these executions. We now note that if $w>v$ and w is reachable, then there must be some sequence of rules in B with a positive displacement. However, it then follows from the proof of Theorem 3.1 in [16] that $\mathcal{V}$ is unbounded -- a contradiction. Hence, if $w>v$, w is not reachable. Suppose there is some i such that $w(i)<v(i)$. We then begin the outer loop of two nested loops. Let $v_j \in B$ be such that $V(i,j)=-1$. (If there is no such $v_j \in B$, then w clearly cannot be reached, so reject.) We wish to collect a set C of rules $v_{j'}$ such that for each additional execution of v_j (beyond what has already been accumulated in v) $v_{j'}$ must be executed an additional time. We therefore initialize C to $\{v_j\}$. We also initialize v′, which will represent the sum of the rules in C, to v_j. Suppose there is some i′ such that adding v′ to v would increase $|v(i')-w(i')|$. This marks the beginning of the inner loop. There must be a $v_{j'} \in B \setminus C$ such that $V(i',j')=-v'(i')$ (see the preceding paragraph concerning the properties of B). $v_{j'}$ must clearly be executed as many additional times as some rule in C is yet to be executed, and all rules in C must be executed at least as many additional times as v_j is yet to be executed. We therefore add $v_{j'}$ to v′ and include $v_{j'}$ in C. We continue iterating this inner loop until there is no i′ such that adding v′ to v would increase $|v(i')-w(i')|$. At the conclusion of this loop, if $v'(i)=0$, this means that the rule $v_{j'}$ such that $V(i,j')=1$ is in C; hence $v_{j'}$ must be executed as many additional times as v_j is yet to be executed. But if this is the case, w can clearly never be reached, since $w(i)<v(i)$. Therefore, if $v'(i)=0$, we reject. Otherwise, let $n=\min_{v'(i')\neq 0}\{|v(i')-w(i')|\}$ (i.e., n is the minimum number of times v′ must be added to v to bring $v(i')=w(i')$ for some i′). We are now ready to accumulate into v n of the executions of the rules in C. We therefore add $n*v'$ to v, and insert the rules in C into B′. This does not increase $|v(i')-w(i')|$ for any i′, and brings it to 0 for some i′. We therefore iterate the outer loop until $w(i)\geq v(i)$ for all i. Again, if $w>v$, w is not reachable, so reject. Otherwise, $w=v$ when the outer loop terminates. The algorithm has therefore demonstrated the existence of a (not necessarily valid) sequence of rules θ containing exactly those rules in $A \cup B'$ such that $v_0+\delta(\theta)=w$.

The final step of the algorithm is to verify that there is some path in $\mathcal{V}$ using exactly those rules in A∪B′; from Lemma 3.1, this can be done in PTIME. We claim that w is reachable iff there is some path in $\mathcal{V}$ using exactly those rules in A∪B′. First, suppose that w is reachable. We have already shown that there is a path from v_0 to w having an initial segment using exactly those rules in A. From Lemma 3.1, there is a path σ' from v_0 consisting of exactly one occurrence of each rule in A. Suppose there is a proper subset B″ of the rules in B′ that can be executed exactly once (in some sequence) after σ', leaving none of the rules in B′\B″ enabled. Call the resulting vector x. Since A∪B is the set of all executable rules in $\mathcal{V}$, from the proof of Lemma 3.1, there must be an i, $1 \leq i \leq m$, a $v_j \in B \backslash B'$, and a $v_{j'} \in B' \backslash B''$ such that x(i)=0, U(i,j′)=1, and V(i,j)=1. Since $v_j \in B$ and V(i,j)=1, V(i,j′)=-1 (see the properties of B enumerated earlier). Since $v_{j'}$ has not yet been used and x(i)=0, $v_0(i)=0$. But this implies that w(i)<0 (otherwise, v_j would have been included in C with $v_{j'}$) -- a contradiction. Therefore, there must be a path from v_0 containing only those rules in A∪B′. Now suppose there is a path from v_0 containing only those rules in A∪B′. Since the algorithm demonstrates that there is a (not necessarily valid) sequence θ containing exactly those rules in A∪B′ such that $\delta(\theta)+v_0=w$, by Lemma 3.2 in [15], these rules can be arranged into a valid sequence.

We have shown that our algorithm correctly decides reachability for bounded conflict-free VRSs. Furthermore, it is easy to see that the algorithm runs in polynomial time. This completes the proof. □

Theorem 3.4: The 2′-fair nontermination problem for conflict-free VRSs is PTIME-complete.

Proof: Let $\mathcal{V}=(v_0,U,V)$ be a $k \times m$ conflict-free VRS. If $\mathcal{V}$ is unbounded, then there is clearly an infinite path which reaches no vector infinitely often; thus, the path is 2′-fair for any $\mathcal{A}$. The VRS constructed in Lemma 4.1 of [16] to show the boundedness problem to be PTIME-hard has an infinite path iff it is unbounded; therefore, the same proof can be used to show that this problem is PTIME-hard. We will now show the problem to be in PTIME. Let $\mathcal{A}$ be a finite set of finite nonempty subsets of N^k. We first determine whether $\mathcal{V}$ is bounded; if not, accept. If $\mathcal{V}$ is bounded, then there is an infinite 2′-fair path iff there is an $A \in \mathcal{A}$ that contains a $w \in R(\mathcal{V})$, and there is an infinite 1′-fair path for $\mathcal{A}$ in (w,U,V). Therefore, for all $A \in \mathcal{A}$ we check all $w \in A$ to determine whether both of these conditions hold for some w. From Lemma 3.3 and Theorem 3.5 below, this can be done in PTIME. □

Theorem 3.5: The 1′-fair nontermination problem for conflict-free VRSs is NLOGSPACE-complete.

Proof: We will first show the problem to be in NLOGSPACE. We first guess a set $A \in \mathcal{A}$. Next, we verify that the start vector is in A. We will maintain a pointer to what we will call the "current vector" in A; initially, the current vector will be the start vector. We repeatedly guess a rule v_j and a vector $w \in A$, and verify that the execution of v_j at the current vector produces w. w then becomes the current

vector. If this process can be continued for more iterations than there are rules in A, there is an infinite 1′-fair path. Clearly, this nondeterministic algorithm can be implemented using only logarithmic space.

We will now show the problem to be NLOGSPACE-hard. We will use a reduction from the graph accessibility problem, which is well known to be NLOGSPACE-complete [18]. Let $G=(Q,E)$ be a directed graph in which $Q=\{q_1,\ldots q_n\}$, and $q_1,q_n\in Q$ are the start vertex and final vertex, respectively. We first construct a graph $G'=(Q',E')$, where $Q'=\{p_{i,j} : q_i\in Q \text{ and } 1\leq j\leq n\}$ and $E'=\{(p_{i,j},p_{i',j+1}) : (q_i,q_{i'})\in E \text{ or } q_i=q_{i'}=q_n,\ 1\leq j\leq n-1\} \cup \{(p_{n,n},p_{n,n})\}$. Clearly, this construction can be done in deterministic logspace, and there is an infinite path from $p_{1,1}$ in G' iff there is a path from q_1 to q_n in G. We will now construct a $(2n^2)\times(3n^2)$ conflict-free VRS $\mathcal{V}$ and a set $A\subseteq N^{2n^2}$ such that there is an infinite path in $\mathcal{V}$ that is 1′-fair for $\{A\}$ iff there is a path from q_1 to q_n in G. For ease of expression, we will use variables to denote the positions in the VRS; the addition rules will then consist of assignments to these variables. The variables we will use are $\{x_{i,j},y_{i,j} : p_{i,j}\in Q'\}$. A 1′-fair path in $\mathcal{V}$ will represent a traversal of G'. The values of all variables will be restricted to $\{0,1\}$ in A. Intuitively, a value of 1 for $x_{i,j}$ means we are entering vertex $p_{i,j}$, and a value of 1 for $y_{i,j}$ means we are leaving vertex $p_{i,j}$. Thus, in the start vector, $x_{1,1}=1$, and all other positions are 0. $\mathcal{V}$ will have the following rules:

- $v^1_{i,j}$, where $p_{i,j}\in Q'$ (prepare to leave $p_{i,j}$):

  $x_{i,j} \leftarrow x_{i,j}-1$
  $y_{i,j} \leftarrow y_{i,j}+1$

- $v^2_{i,j}$, where $p_{i,j}\in Q'$ (pick $p_{i,j}$ as next vertex):

  $x_{i,j} \leftarrow x_{i,j}+1$

- $v^3_{i,j}$, where $p_{i,j}\in Q'$ (go to next vertex):

  $y_{i,j} \leftarrow y_{i,j}-1$

All rules superscripted with i will be called *type i* rules. The set A will contain all 0-1 vectors in N^{2n^2} that contain exactly one 1, as well as all 0-1 vectors containing exactly two 1's such that $y_{i,j}=x_{i',j'}=1$ iff $(p_{i,j},p_{i',j'})\in E'$. This will guarantee that any 1′-fair path in $\mathcal{V}$ represents a traversal of G'. Clearly, $\mathcal{V}$ is conflict-free, and the construction can be done in deterministic logspace. We will now show that there is an infinite path in $\mathcal{V}$ that is 1′-fair for $\{A\}$ iff there is an infinite path from $p_{1,1}$ in G'.

($\Rightarrow$): Let σ be an infinite path in $\mathcal{V}$ that is 1′-fair for $\{A\}$. Then every vector reached by σ is in A. We associate with σ a sequence of vertices in Q' as follows: with each point reached by σ in which one x variable is 1 and all other variables are 0, associate the vertex $p_{i,j}$ such that $x_{i,j}=1$. We will show by induction that for every $h>0$, (1) there are at least h vertices in the sequence, and (2) the first h vertices in the sequence form a path from $p_{1,1}$ in G'. Clearly, this holds for $h=1$. Let $h>1$, and assume the claim holds for h-1. Suppose the last of the h-1 points has $x_{i,j}=1$. Now since $\mathbf{0}\notin A$ and all vectors in A with

more than one 1 have a y variable equal to 1, $v^1_{i,j}$ must be executed next in σ. Now since executing $v^3_{i,j}$ would produce $\mathbf{0}\notin A$, some type 2 rule must be executed. This produces a vector with $x_{i',j'}=y_{i,j}=1$. Since this vector must be in A, $(p_{i,j},p_{i',j'})\in E'$. Now clearly, the only rule that will produce a vector in A is $v^3_{i,j}$. This produces a vector in which $x_{i',j'}$ is the only position equal to 1. Since $(p_{i,j},p_{i',j'})\in E'$ and there is a path of h-1 vertices from $p_{1,1}$ to $p_{i,j}$, the claim holds for h. Thus, it is clear that there is an infinite path from $p_{1,1}$ in G'.

($\Leftarrow$): Let σ be an infinite path from $p_{1,1}$ in G'. Associate with σ a sequence of vectors such that with each vertex $p_{i,j}$ reached by σ is associated the vector with $x_{i,j}=1$ and all other variables equal to 0. We will show by induction that for any $h>0$ there is a path in $\mathcal{V}$ that remains within A and passes in order through the first h points in the sequence associated with σ. Clearly, this holds for h=1. Let $h>1$, and assume the claim for h-1. Suppose the $(h-1)^{st}$ vertex reached by σ is $p_{i,j}$. Let $p_{i',j'}$ be the next vertex reached by σ. Then $\mathcal{V}$ can clearly execute $v^1_{i,j}$, $v^2_{i',j'}$, and $v^3_{i,j}$, producing vectors in A, the last of which has $x_{i',j'}=1$ and all other variables equal to 0. Thus, the claim holds for h. It is now clear that there is an infinite path in $\mathcal{V}$ that is 1′-fair for {A}. This completes the proof. □

4. The Model Checking Problem

We now consider a generalization of the problems considered in the previous section. Each of these problems can be expressed as an instance of model checking. For example, fairness can be defined for a k×m VRS $\mathcal{V}$ using the TL formula $f=\wedge_{j=1}^{m}(\mathbf{GF}en(j)\supset\mathbf{GF}fi(j))$. Thus, $\mathcal{V}$ has an infinite fair path iff $\mathcal{V}$ is a model for f. Since all of the problems considered in the previous section were decidable, some very efficiently, one might expect model checking in general for conflict-free VRSs to be decidable, with perhaps some nontrivial subset that is efficiently decidable. However, the following theorem puts to rest the first of these expectations, and strongly suggests that the second cannot be met, either.

Theorem 4.1: The model checking problem for conflict-free VRSs is undecidable, even when the TL formula is of the form **G**f, where f contains no TL operators.

Proof: We use a reduction from the halting problem for 2-counter machines (2CMs) on input ϵ; since this problem is undecidable [31], the result follows. Let M be a 2CM with state set $Q=\{q_1,...,q_s\}$, start state q_1, and transition relation δ with r transition rules $\{t_1,...,t_r\}$. Each transition rule is of the form $t_j=(<q_{i_j},z_{1,j},z_{2,j}>, <q_{i'_j},a_{1,j},a_{2,j}>)$, where $z_{i,j}\in\{\text{"}C_i=0\text{"}, \text{"}C_i\neq 0\text{"}\}$ and $a_{i,j}\in\{-1,0,1\}$. t_j may be executed only if M is in state q_{i_j} and both $z_{i,j}$ and $z_{2,j}$ are true of M's counters, C_1 and C_2. Upon execution of t_j, M enters state $q_{i'_j}$, and $a_{1,j}$ and $a_{2,j}$ are added to C_1 and C_2, respectively. We construct a $(2s+6)\times(r+2s+4)$ conflict-free VRS $\mathcal{V}$ as follows. We will represent the 2s+6 rows using the following variables: x_1, ..., x_s, $\bar{x}_1$, ..., $\bar{x}_s$, c_1, c_2, $\bar{c}_{1,-1}$, $\bar{c}_{1,1}$, $\bar{c}_{2,-1}$, $\bar{c}_{2,1}$. Each addition rule will then act on these variables. We

subsequently present addition rules and a TL formula that will cause the variables to simulate a computation of M on ϵ as follows. Informally, x_i=1 will indicate that M is in state q_i, $\bar{x}_i$=1 will indicate that M is entering state q_i, $\bar{c}_{i,a}$=1 will indicate that a must be added to counter C_i, and c_i will contain the value of C_i. Each move of M will be simulated by a series of addition rules. At the beginning of each series, there will exist an i such that x_i=1 and $x_{i'}$=0 for all $i' \neq i$. This will indicate that M is in state q_i. Furthermore, all $\bar{x}$ and $\bar{c}$ variables will be zero. The first addition rule in the series will correspond to choosing some transition t_j such that i_j=i. This rule will increment $\bar{x}_{i'_j}$ and $\bar{c}_{n,a_{n,j}}$ if $a_{n,j} \neq 0$, for n=1,2. Next, for each $\bar{c}_{n,a_{n,j}}$ that was incremented by the first addition rule in the series, $a_{n,j}$ is added to c_n, and $\bar{c}_{n,a_{n,j}}$ is decremented. After this, x_i is decremented, and finally, $x_{i'}$ is incremented and $\bar{x}_{i'}$ is decremented. Note that at the end of the series, the variables have proper values to begin a new series if some transition is enabled.

We now formally define the addition rules (again, all rules superscripted with i will be called type i rules):

- v_j^1, $1 \leq j \leq r$ (initiate transition t_j):

  $\bar{x}_{i'_j} \leftarrow \bar{x}_{i'_j} + 1$

  $\bar{c}_{1,a_{i,j}} \leftarrow \bar{c}_{1,a_{i,j}} + 1$, if $a_{i,j} \neq 0$, for i=1,2

- v_j^2, $1 \leq j \leq s$ (leave state q_j):

  $x_j \leftarrow x_j - 1$

- v_j^3, $1 \leq j \leq s$ (enter state q_j):

  $\bar{x}_j \leftarrow \bar{x}_j - 1$

  $x_j \leftarrow x_j + 1$

- v_j^4, $1 \leq j \leq 2$ (decrement counter C_j):

  $c_j \leftarrow c_j - 1$

  $\bar{c}_{j,-1} \leftarrow \bar{c}_{j,-1} - 1$

- v_j^5, $1 \leq j \leq 2$ (increment counter C_j):

  $c_j \leftarrow c_j + 1$

  $\bar{c}_{j,1} \leftarrow \bar{c}_{j,1} - 1$

We now construct a TL formula f that will guarantee that $\mathcal{V}$ faithfully simulates M. Let $p_{i,j}$ be defined as $ge(c_i,1)$ if $z_{i,j}$="$C_i \neq 0$", or $\neg ge(c_i,1)$ if $z_{i,j}$="$C_i = 0$", for i=1,2, $1 \leq j \leq r$. Now let f=G(A$\wedge$B$\wedge$C$\wedge$D$\wedge$E$\wedge$F$\wedge$G), where

$$A = [\wedge_{j=1}^{s} \neg ge(\bar{x}_j,1)] \supset [\vee_{j=1}^{r} ge(x_{i_j},1) \wedge p_{1,j} \wedge p_{2,j} \wedge fi(v_j^1)];$$

$$B = en(v_1^4) \supset fi(v_1^4);$$

$C = en(v_1^5) \supset fi(v_1^5);$

$D = [en(v_2^4) \wedge \neg en(v_1^4) \wedge \neg en(v_1^5)] \supset fi(v_2^4);$

$E = [en(v_2^5) \wedge \neg en(v_1^4) \wedge \neg en(v_1^5)] \supset fi(v_2^5);$

$F = \wedge_{j,j'=1}^{s} [(en(v_j^2) \wedge \neg en(v_1^4) \wedge \neg en(v_2^4) \wedge \neg en(v_1^5) \wedge \neg en(v_2^5) \wedge ge(\bar{x}_{j',1}) \supset fi(v_j^2)];$

$G = [\wedge_{j=1}^{s} \neg ge(x_j,1)] \supset [\vee_{j=1}^{s} fi(v_j^3)].$

Now let the start vector have x_1=1 and all other variables 0. Clearly, $\mathcal{V}$ is conflict-free. Clause A in f guarantees that a valid transition in M is chosen and that type 1 addition rules are used as the first rule in each series (i.e., when all $\bar{x}$ variables are 0). Clauses B, C, D, and E guarantee that the proper counters are modified accordingly, first C_1, then C_2. Finally, Clauses F and G guarantee that the state is changed properly. It should therefore be reasonably clear that there is an infinite valid sequence θ such that $<\mathcal{V},\theta,0> \models$ f iff M does not halt. □

The above proof is actually quite powerful. Since $\mathbf{G}f \equiv \neg\mathbf{F}\neg f$, the undecidability result immediately follows for arbitrary TL formulas using only the **F** TL operator. This set of formulas corresponds to L(**F**), defined in [40] with respect to finite state structures. In addition, the proof can be readily extended to safety and liveness properties, as defined in [27, 46]. Informally, a *safety property* is of the form **G**f, where f is a formula in a logic that can refer only to events in the past. Clearly, Theorem 4.1 shows model checking to be undecidable for safety properties. Also informally, a *simple liveness property* is either of the form **F**f, **GF**f, or **FG**f, where again f is a past formula. By constructing f to assert that M is in a final state and that the past represents a valid computation of M, model checking for each of these forms can be shown to be undecidable; the proof is left to the reader. For formal definitions of safety and liveness properties, see [27, 46].

In view of Theorem 4.1, it might be interesting to consider model checking for formulas of the form **F**f, **FG**f, or **GF**f, where f contains no TL operators. The latter two forms are interesting because in [9], fairness constraints were characterized by the canonical form $\vee_{i=1}^{n} \wedge_{j=1}^{n_i} (\mathbf{FG}f_{ij} \vee \mathbf{GF}g_{ij})$. Although we are unable to give any results for formulas of the form **GF**f, we will now show that model checking is undecidable for formulas of the form **FG**f; it will follow from a later theorem that for formulas of the form **F**f, model checking for conflict free VRSs is NP-complete.

Theorem 4.2: The model checking problem for conflict-free VRSs is undecidable for TL formulas of the form **FG**f, where f contains no TL operators.

Proof: We again use a reduction from the halting problem for 2CMs on input ϵ. Let M be an arbitrary

2CM. We first construct a 4-counter machine M' that simulates successively longer and longer computations of M on ϵ. M' will use two counters to record the contents of M's counters. M' uses the other two counters -- say C_1 and C_2 -- as a "clock" to stop the simulation of M after a certain number of moves. Thus, if C_1 contains the number of moves to be simulated, then after each simulated move of M, M' decrements C_1 and increments C_2; when $C_1=0$, M' transfers C_2 to C_1, adds 1, and restarts the simulation. The construction of M' from M is straightforward and is left to the reader. From M' we then construct a conflict-free VRS $\mathcal{V}$ and a TL formula **Gf** to simulate M', ala Theorem 4.1. Now suppose θ is a valid sequence such that $<\mathcal{V},\theta,0> \models \mathbf{Gf}$. All vectors reachable in executing θ meet certain syntactical requirements; for example, in the proof of Theorem 4.1, x_i and x_j cannot both be 1 if $i \neq j$. These syntax requirements can clearly be described in a formula g with no TL operators. Consider the TL formula $\mathbf{FG}(f \wedge g)$. Suppose θ is an infinite valid sequence such that $<\mathcal{V},\theta,0> \models \mathbf{FG}(f \wedge g)$. Then for some $n>0$, $<\mathcal{V},\theta,n> \models \mathbf{G}(f \wedge g)$. This means that after the first n rules in θ have been executed, the resulting vector is a syntactically valid description of a configuration of M', and the remainder of θ faithfully simulates an infinite computation of M' from this configuration. Suppose that in this configuration, C_1 contains the value p. Then M' simulates M (from some configuration) for p steps, then restarts the simulation of M on ϵ. From this point on, M' simulates M on ϵ for successively longer and longer computations. Thus, it is clear that $\mathbf{FG}(f \wedge g)$ is a model for $\mathcal{V}$ iff M has an infinite computation. □

The major difficulty in model-checking seems to be that the simplest temporal logics are so expressive that unleashing their power on even the most simple infinite state computing structures enables one to construct temporal logic formulas that can only be satisfied by the structure emulating computations of much more powerful automata. However, a subset of temporal logic was defined in [40] with respect to finite state structures that we can adapt to conflict-free VRSs to produce a logic for which model checking is NP-complete. Let $\widetilde{L}(\mathbf{F},\mathbf{X})$ be the subset of temporal logic that uses the Boolean connectives $\wedge$ and $\vee$ and the temporal operators **F** and **X**, with negations allowed only on the predicates. We will refer to the predicates and their negations as *literals*. Before we can describe our model checking algorithm for $\widetilde{L}(\mathbf{F},\mathbf{X})$, we must make the following definition. A formula in $\widetilde{L}(\mathbf{F},\mathbf{X})$ is said to be *regular* if it is a literal or of the form $\alpha \wedge f$, **F**f, or **X**f, where α is a literal and f is regular.

Our algorithm consists of two parts. Given a formula f in $\widetilde{L}(\mathbf{F},\mathbf{X})$, the first part will nondeterministically generate a regular formula f_r such that

1. for any f_r generated by the algorithm, if θ is an infinite valid sequence in a VRS $\mathcal{V}$ such that $<\mathcal{V},\theta,0> \models f_r$, then $<\mathcal{V},\theta,0> \models f$;

2. if some VRS $\mathcal{V}$ is a model for f, then $\mathcal{V}$ is a model for some f_r generated by the algorithm.

For this purpose we present Algorithm 1:

```
function reg(f)
    case 1    f is a literal: return(f)
```

case 2 f=**F**g: **return**(**F**reg(g))
case 3 f=**X**g: **return**(**X**reg(g))
case 4 f=g∨h: **return**(reg(g) **or** reg(h))
case 5 f=g∧h: **return**(reg2(reg(g),reg(h)))
end

function reg2(g_r,h_r)
case 1 g_r is a literal: **return**($g_r \wedge h_r$)
case 2 $g_r = \alpha \wedge g'_r$, α a literal: **return**($\alpha \wedge$reg2(g'_r,h_r))
case 3 $g_r = \mathbf{F}g'_r$ **and** $h_r = \mathbf{F}h'_r$:
return(**F**reg2(g'_r,**F**h'_r) **or** **F**reg2(h'_r,**F**g'_r) **or** **F**reg2(g'_r,h'_r))
case 4 $g_r = \mathbf{F}g'_r$ **and** $h_r = \mathbf{X}h'_r$: **return**(**X**reg2(g'_r,h'_r) **or** **X**reg2(h'_r,**F**g'_r))
case 5 $g_r = \mathbf{X}g'_r$ **and** $h_r = \mathbf{F}h'_r$: **return**(**X**reg2(g'_r,h'_r) **or** **X**reg2(g'_r,**F**h'_r))
case 6 $g_r = \mathbf{X}g'_r$ **and** $h_r = \mathbf{X}h'_r$: **return**(**X**(reg2(g'_r,h'_r))
end

Algorithm 1

We claim that the function reg in Algorithm 1 generates formulas satisfying conditions (1) and (2) above. The purpose of reg2 is to generate from two regular formulas g_r and h_r formulas f_r satisfying conditions (1) and (2), where $f = g_r \wedge h_r$. It is easily seen that Algorithm 1 operates in NP. Hence, we will now verify first reg2, then reg. In order to do this, we give the following lemma, which follows immediately from the semantics of the logic.

Lemma 4.1: If f_r and g_r satisfy either condition (1) or condition (2) with respect to f and g, respectively, then **F**f_r, **X**f_r, and $f_r \wedge g_r$ satisfy the same condition with respect to **F**f, **X**f, and f∧g, respectively.

Lemma 4.2: If $f = g_r \wedge h_r$, where g_r and h_r are regular, then reg2(g_r,h_r) satisfies conditions (1) and (2) with respect to f.

Proof: We proceed by induction on the length of f. If f has length 1, the lemma vacuously holds. Therefore, let f have length n>1 and assume the lemma for all formulas having length less than n. We now consider the cases as enumerated in the algorithm.

Case 1: Trivial.

Case 2: From the induction hypothesis, reg2(g'_r,h_r) satisfies the conditions with respect to $g'_r \wedge h_r$. From Lemma 4.1, $\alpha \wedge$reg2($g'_r \wedge h_r$) satisfies the conditions with respect to f.

Case 3: It follows from the semantics of the logic that $\mathbf{F}g'_r \wedge \mathbf{F}h'_r \equiv \mathbf{F}(g'_r \wedge \mathbf{F}h'_r) \vee \mathbf{F}(h'_r \wedge \mathbf{F}g'_r) \vee \mathbf{F}(g'_r \wedge h'_r)$. Therefore, if θ is an infinite valid sequence in a VRS $\mathcal{V}$ such that

$<\mathcal{V},\theta,0> \models f'$, where $f' \in \{\mathbf{F}(g'_r \wedge \mathbf{F}h'_r), \mathbf{F}(h'_r \wedge \mathbf{F}g'_r), \mathbf{F}(g'_r \wedge h'_r)\}$, then $<\mathcal{V},\theta,0> \models f$. From the induction hypothesis and Lemma 4.1, if $f_r \in \{\mathbf{F}(\text{reg2}(g'_r,\mathbf{F}h'_r)), \mathbf{F}(\text{reg2}(h'_r,\mathbf{F}g'_r)), \mathbf{F}\text{reg2}(g'_r,h'_r)\}$, then f_r satisfies condition (1). Furthermore, if $\mathcal{V}$ is a model for f, the $\mathcal{V}$ must clearly be a model for some $f' \in \{\mathbf{F}(g'_r \wedge \mathbf{F}h'_r), \mathbf{F}(h'_r \wedge \mathbf{F}g'_r), \mathbf{F}(g'_r \wedge h'_r)\}$. Thus, from the induction hypothesis and Lemma 4.1, condition (2) must hold for some $f_r \in \{\mathbf{F}(\text{reg2}(g'_r,\mathbf{F}h'_r)), \mathbf{F}(\text{reg2}(h'_r,\mathbf{F}g'_r)), \mathbf{F}\text{reg2}(g'_r,h'_r)\}$.

Case 4: It follows from the semantics of the logic that $\mathbf{F}g'_r \wedge \mathbf{X}h'_r \equiv \mathbf{X}(g'_r \wedge h'_r) \vee \mathbf{X}(h'_r \wedge \mathbf{F}g'_r)$. The conclusion therefore follows by a similar argument to that in Case 3.

Case 5: Symmetric to Case 4.

Case 6: It follows from the semantics of the logic that $\mathbf{X}g'_r \wedge \mathbf{X}h'_r \equiv \mathbf{X}(g'_r \wedge g'_r)$. The conclusion therefore follows as above. □

Lemma 4.3: If f is a formula in $\widetilde{L}(\mathbf{F},\mathbf{X})$, then reg(f) satisfies conditions (1) and (2) with respect to f.

Proof: Similar to the proof of Lemma 4.2. □

The second part of our algorithm is to nondeterministically generate a system of linear Diophantine inequalities from a given regular formula f_r. Let f be a regular formula with t TL operators, (v_0,U,V) be a conflict-free VRS, and n be an arbitrary natural number. The system will contain variables which we will group as vectors: k-dimensional vectors w_i, $n \le i \le n+t$, and m-dimensional vectors x_i, $n \le i \le n+t-1$, and $\bar{x}_i$, $n \le i \le n+t$. The vectors $w_n,\ldots,w_{n+t}$ represent certain vectors in a path in (v_0,U,V). $\bar{x}_i$ will represent a bit map indicating which addition rule is executed at w_i if one is executed. x_i will represent the Parikh map of the path beginning after the first rule is executed at w_i and ending at w_{i+1}. We now present Algorithm 2 to generate this system.

function gen(f,U,V,n)

case 1 f=ge(i,a):

return($\{w_n(i) \ge a, \sum_{j=1}^{m} \bar{x}_n(j) \le 1, w_n \ge U\bar{x}_n\}$)

case 2 f=¬ge(i,a):

return($\{w_n(i) < a, \sum_{j=1}^{m} \bar{x}_n(j) \le 1, w_n \ge U\bar{x}_n\}$)

case 3 f=en(j):

return($\{w_n \ge u_j, \sum_{j'=1}^{m} \bar{x}_n(j') \le 1, w_n \ge U\bar{x}_n\}$)

case 4 f=¬en(j):

guess i, $1 \le i \le k$;

return($\{w_n(i) < U(i,j), \sum_{j'=1}^{m} \bar{x}_n(j') \le 1, w_n \ge U\bar{x}_n\}$)

case 5 f=fi(j):

return($\{\bar{x}_n(j) = 1, \sum_{j'=1}^{m} \bar{x}_n(j') = 1, w_n \ge U\bar{x}_n\}$)

case 6 f=¬fi(j):

return($\{\bar{x}_n(j)=0, \sum_{j'=1}^{m} \bar{x}_n(j') \leq 1, w_n \geq U\bar{x}_n\}$)

case 7 f=$\alpha \wedge$g: **return**(gen(α,U,V,n)$\cup$gen(g,U,V,n))

case 8 f=**X**g:

return($\{\sum_{j=1}^{m} \bar{x}_n(j)=1, x_n=0^m, w_n \geq U\bar{x}_n, w_{n+1}=w_n+V\bar{x}_n\}$
$\cup$ gen(g,U,V,n+1))

case 9 f=**F**g:

guess w, $0^k \leq w \leq 1^k$;

guess $S \subseteq [1..m]$;

if there is a path from w containing exactly those rules v_j
such that $j \in S$

then

return($\{\sum_{j=1}^{m} \bar{x}_n(j)=1, w_n \geq U\bar{x}_n, w_n+V\bar{x}_n \geq w, w_{n+1}=w_n+V\bar{x}_n+Vx_n\}$
$\cup \{x_n(j)=0 : j \notin S\} \cup \{x_n(j) \geq 1 : j \in S\} \cup$ gen(g,U,V,n+1))

else fail

end

Algorithm 2

It can easily be seen that the number of recursive calls of gen is no more than the length of f. Furthermore, by Lemma 3.2, the condition in the **if** statement can be evaluated in time polynomial in the size of V. Therefore, Algorithm 2 operates in NP. We now give the following lemma, which characterizes the system of inequalities produced by gen.

Lemma 4.4: For any regular formula f and any conflict-free VRS $\mathcal{V}$=(v_0,U,V), $\{w_n=v_0\} \cup$ gen(f,U,V,n) has a nonnegative integer solution for some computation of gen iff there is a valid sequence θ such that $<\mathcal{V},\theta,0> \models$ f. Furthermore, we can require that $\bar{x}_n(j)=1$ iff v_j is the first rule in θ.

Proof: By induction on the length of f. If f is a literal, the lemma follows by inspection of cases 1-6; therefore, the induction is well-based. We now let f be any regular formula that is not a literal, and assume the lemma for all regular formulas shorter than f. We must therefore consider cases 7-9.

Case 7: Suppose $\{w_n=v_0\} \cup$ gen(f,U,V,n) has a nonnegative integer solution for some computation of gen. Call this solution X, and the set of inequalities S. Clearly, X also holds for $S_g = \{w_n=v_0\} \cup$ gen(g,U,V,n) and $S_\alpha = \{w_n=v_0\} \cup$ gen(α,U,V,n), since these are both subsets of S. Therefore, there are valid sequences θ_α and θ_g such that $<\mathcal{V},\theta_\alpha,0> \models \alpha$ and $<\mathcal{V},\theta_g,0> \models$ g. Now α is an assertion only on v_0 and first rule in θ_α. Therefore, for any valid sequence θ that starts with the same rule as the first rule in θ_α, $<\mathcal{V},\theta,0> \models \alpha$. Now by inspection of cases 1-6, gen(α,U,V,n) must contain either $\sum_{j=1}^{m} \bar{x}_n(j)=1$ or $\sum_{j=1}^{m} \bar{x}_n(j) \leq 1$. If $\sum_{j=1}^{m} \bar{x}_n(j)=0$ in X, then from the induction hypothesis, both θ_α and θ_g must be the empty sequence; so $<\mathcal{V},\theta_g,0> \models \alpha \wedge$g=f. If, on the other hand, $\sum_{j=1}^{m} \bar{x}_n(j)=1$ in X, from the induction

hypothesis, both θ_α and θ_g start with the same rule; so $<\mathcal{V},\theta_g,0> \models f$. Now suppose conversely that there is a valid sequence θ such that $<\mathcal{V},\theta,0> \models f$. Then $<\mathcal{V},\theta,0> \models \alpha$ and $<\mathcal{V},\theta,0> \models g$. From the induction hypothesis, there must be a solution X_α for the system $\{w_n=v_0\} \cup gen(\alpha,U,V,n)$ and a solution X_g for the system $\{w_n=v_0\} \cup gen(g,U,V,n)$ such that $\bar{x}_n(j)=1$ (in both solutions) iff v_j is the first rule in θ. Now by inspection of cases 1-6, $gen(\alpha,U,V,n)$ only contains the variables w_n and $\bar{x}_n$, whose values must clearly be the same in X_α and X_g. Therefore, X_g is a solution for $\{w_n=v_0\} \cup gen(f,U,V,n)$, for some computation of gen.

Case 8: Suppose X is a solution to $\{w_n=v_0\} \cup gen(f,U,V,n)$, for some computation of gen. Clearly, X is also a solution to $\{w_{n+1}=w_n+V\bar{x}_n\} \cup gen(g,U,V,n+1)$. In order to satisfy the inequalities introduced in case 8, $\bar{x}_n$ in X must have exactly one element with a value of 1, and the rest with values of 0. Let j be such that $\bar{x}_n(j)=1$. Then in X, $w_{n+1}=v_0+v_j$. Letting $\mathcal{V}'=(v_0+v_j,U,V)$, by the induction hypothesis, there is a sequence θ' such that $<\mathcal{V}',\theta',0> \models g$. In order to satisfy the inequalities introduced in case 8, $v_0 \geq u_j$. Therefore, by inserting v_j at the beginning of θ', we have a valid sequence θ in $\mathcal{V}$ such that $<\mathcal{V},\theta,0> \models f$. Now suppose conversely that θ is a valid sequence in $\mathcal{V}$ such that $<\mathcal{V},\theta,0> \models f$. Since θ must clearly have at least one addition rule, let v_j be the first rule in θ. Again letting $\mathcal{V}'=(v_0+v_j,U,V)$, and letting θ' be the sequence obtained by removing the first rule from θ, we have $<\mathcal{V}',\theta',0> \models g$. From the induction hypothesis, there is a solution X' to the system $\{w_{n+1}=v_0+v_j\} \cup gen(g,U,V,n+1)$. It is easily shown by induction that this system contains no occurrences of w_n, x_n, or $\bar{x}_n$. Therefore, by letting $\bar{x}_n(j)=1$, it is easily seen that $\{w_n=v_0\} \cup gen(f,U,V,n)$ has a solution.

Case 9: Suppose X is a solution to $\{w_n=v_0\} \cup gen(f,U,V,n)$. Clearly, X is also a solution to $\{w_{n+1}=w_n+V\bar{x}_n+Vx_n\} \cup gen(g,U,V,n+1)$. In order to satisfy the inequalities introduced in case 9, $\bar{x}_n$ in X must have exactly one element with a value of 1, and the rest with values of 0. Let j be such that $\bar{x}(j)=1$. Since gen terminated successfully, there must be a sequence θ'' enabled at some w, $0^k \leq w \leq 1^k$, using exactly those rules v_j such that $x_n(j) \geq 1$ in X. In order to satisfy the inequalities introduced in case 9, $v_0+v_j \geq w$, so θ'' must be enabled at v_0+v_j. Now since $w_{n+1} \geq \mathbf{0}$, from Lemma 3.2 in [17], there exists a path from v_0+v_j to $v_0+v_j+Vx_n$. Furthermore, in order to satisfy the inequalities introduced in case 9, v_j must be enabled at v_0, so there is a path from v_0 to $v_0+v_j+Vx_n$. Now by letting $\mathcal{V}'=(v_0+v_j+Vx_n,U,V)$, we have from the induction hypothesis that there is a sequence θ' such that $<\mathcal{V}',\theta',0> \models g$. Therefore, there is clearly a sequence θ beginning with v_j such that $<\mathcal{V},\theta,0> \models f$. Now suppose conversely that θ is a valid sequence such that $<\mathcal{V},\theta,0> \models f$. Now there must exist an $s>0$ such that $<\mathcal{V},\theta,s> \models g$. Let θ_1 be the first s rules in θ, and let θ_2 be the remainder of θ. Let y be the vector produced by executing θ_1 at v_0. Now by letting $\mathcal{V}'=(y,U,V)$, we have from the induction hypothesis that the system $\{w_{n+1}=y\} \cup gen(g,U,V,n+1)$ has a solution, say X'. Let $\bar{x}_n(j)=1$ if v_j is the first rule used in θ_1, 0 otherwise, and let $w_{n+1}=y$, $x_n=\Psi(\theta_1)-\bar{x}_n$. Since w_n, x_n, and $\bar{x}_n$ clearly do not appear in $\{w_{n+1}=y\} \cup gen(g,U,V,n+1)$, we only need to show that there is a w, $0^m \leq w \leq 1^m$, such that there is a path from w containing exactly

those rules v_j such that $x_n(j) \geq 1$ and $w \leq v_0 + V\bar{x}_n$. Let $S=\{j : x_n(j) \geq 1\}$. From Lemma 3.1, there is a sequence enabled at $v_0+V\bar{x}_n$ using exactly one occurrence of each v_j such that $j \in S$. Therefore, by letting $w(j)=0$ if $v_0+V\bar{x}_n=\mathbf{0}$, then $w(j)=1$; otherwise, w clearly satisfies the necessary conditions. Therefore, $\{w_n=v_0\} \cup gen(f,U,V,n)$ has a solution. □

We are now ready to show the model checking problem for conflict-free VRSs over $\widetilde{L}(\mathbf{F},\mathbf{X})$ to be NP-complete.

Theorem 4.3: The model checking problem for conflict-free VRSs when restricted to $\widetilde{L}(\mathbf{F},\mathbf{X})$ is NP-complete.

Proof: We first show the problem to be in NP. Let f be an arbitrary formula in $\widetilde{L}(\mathbf{F},\mathbf{X})$, and let $\mathcal{V}=(v_0,U,V)$ be an arbitrary conflict-free VRS. From Lemma 4.3, $\mathcal{V}$ is a model for f iff reg(f) can produce a formula f′ such that $\mathcal{V}$ is a model for f′. From Lemma 3.2, if there exists an infinite path in $\mathcal{V}$, then any finite path can be extended to an infinite path; furthermore, this property can be checked in polynomial time. We therefore verify that there is an infinite path in $\mathcal{V}$. We now define a function Φ mapping regular formulas to regular formulas by replacing the rightmost literal α with $\alpha \wedge \mathbf{X}$true. It is easy to show by induction on the size of f′ that for any $\mathcal{V}$ containing an infinite path, $\mathcal{V}$ is a model for f′ iff there is a valid sequence θ in $\mathcal{V}$ such that $<\mathcal{V},\theta,0> \models \Phi(f')$. From Lemma 4.4, there is a valid sequence θ in $\mathcal{V}$ such that $<\mathcal{V},\theta,0> \models \Phi(f')$ iff $\{w_n=v_0\} \cup gen(\Phi(f'),U,V,0)$ has a nonnegative integer solution for some computation of gen. Since reg and gen operate in NP, and since integer linear programming is in NP [4], model checking is in NP.

We now show the problem to be NP-hard. We use a reduction from the reachability problem for conflict-free VRSs. Let $\mathcal{V}=(v_0,U,V)$ be an arbitrary $k \times m$ conflict-free VRS, and let w be an arbitrary vector in N^k. Let $\mathcal{V}'$ be $\mathcal{V}$ with a column of zeros appended to U and V, and let $f=\mathbf{F} \wedge_{j=1}^{k} (ge(j,w(j)) \wedge \neg ge(j,w(j)+1))$. Clearly, $w \in R(\mathcal{V})$ iff $\mathcal{V}'$ is a model for f. Since reachability is NP-hard [17], model checking is NP-hard, and thus NP-complete. □

Corollary 4.1: The model checking problem for conflict-free VRSs is NP-complete when restricted to formulas of the form **F**f, where f contains no TL operators.

Acknowledgment: We would like to thank Prof. Vidal-Naquet for pointing out the definition of fairness given in [2, 37]. We would also like to thank the Petri net workshop referees for numerous comments which helped improve the presentation of these results. We are especially grateful to the referee that pointed out the bug in our original consideration of the 2′-fair nontermination problem.

References

[1] Apt, K. and Kozen, D., Limits for Automatic Verification of Finite-State Concurrent Systems, *Information Processing Letters 22* (1986), 307-310.

[2] Best, E., Fairness and Conspiracies, *Information Processing Letters 18* (1984), 215-220.

[3] Brams, G., *Reseaux de Petri: Theorie et Pratique -- Tome 1: Theorie et Analyse,* (Masson, Paris, 1983).

[4] Borosh, I. and Treybig, L., Bounds on Positive Integral Solutions of Linear Diophantine Equations, *Proc. AMS 55,* 2 (March 1976), pp. 299-304.

[5] Carstensen, H., Decidability Questions for Fairness in Petri Nets, *Proceedings of the 4th Symposium on Theoretical Aspects of Computer Science,* LNCS 247 (1987), 396-407.

[6] Carstensen, H. and Valk, R., Infinite Behaviour and Fairness in Petri Nets, in: Rozenberg, G., Ed., *Advances in Petri Nets 1984;* LNCS 188, (Springer, Berlin, 1985), pp. 83-100.

[7] Clarke, E., Grümberg, O., and Browne, M., Reasoning about Networks with Many Identical Finite-State Processes, *Proceedings of the 5th Symposium on Principles of Distributed Computing* (1986), 240-248.

[8] Crespi-Reghizzi, S. and Mandrioli, D., A Decidability Theorem for a Class of Vector Addition Systems, *Information Processing Letters 3,* 3 (1975), pp. 78-80.

[9] Emerson, E. and Lei, C., Modalities for Model Checking: Branching Time Logic Strikes Back, to appear in *Science of Computer Programming.* (Some of these results were presented at the *18th Annual Hawaii International Conference on System Sciences* and at the *12th Annual ACM Symposium on Principles of Programming Languages.*).

[10] Ginzburg, A. and Yoeli, M., Vector Addition Systems and Regular Languages, *J. of Computer and System Sciences 20* (1980), pp. 277-284.

[11] Grabowski, J., The Decidability of Persistence for Vector Addition Systems, *Information Processing Letters 11,* 1 (1980), pp. 20-23.

[12] Hart, S., Sharir, M., and Pnueli, A., Termination of Probabilistic Concurrent Programs, *ACM Transactions on Programming Languages and Systems 5* (1983), 356-380.

[13] Hopcroft, J. and Pansiot, J., On the Reachability Problem for 5-Dimensional Vector Addition Systems, *Theoret. Comp. Sci. 8* (1979), pp. 135-159.

[14] Howell, R., Rosier, L., Huynh, D., and Yen, H., Some Complexity Bounds for Problems Concerning Finite and 2-Dimensional Vector Addition Systems with States, *Theoret. Comp. Sci. 46* (1986), 107-140.

[15] Howell, R., and Rosier, L., Completeness Results for Reachability, Containment, and Equivalence with Respect to Conflict-Free Vector Replacement Systems, to be presented at the *14th International Colloquium on Automata, Languages, and Programming,* July, 1987, Karlsruhe, F.R.G. Also Rep. 86-21, The University of Texas at Austin, Austin, Texas, 78712, 1986.

[16] Howell, R., Rosier, L., and Yen, H., An $O(n^{1.5})$ Algorithm to Decide Boundedness for Conflict-Free Vector Replacement Systems, *Information Processing Letters 25* (1987), 27-33.

[17] Howell, R., and Rosier, L., *Completeness Results for Conflict-Free Vector Replacement Systems*, Rep. 86-21, (The University of Texas at Austin, Austin, Texas, 78712, 1986).

[18] Jones, N., Space-Bounded Reducibility Among Combinatorial Problems, *J. of Computer and System Sciences 11* (1975), 68-75.

[19] Jones, N., Landweber, L. and Lien, Y., Complexity of Some Problems in Petri Nets, *Theoret. Comp. Sci. 4* (1977), pp. 277-299.

[20] Karp, R. and Miller, R., Parallel Program Schemata, *J. of Computer and System Sciences 3*, 2 (1969), pp. 147-195.

[21] Keller, R.M., Vector Replacement Systems: A Formalism for Modelling Asynchronous Systems, TR 117, (Princeton University, CSL, 1972).

[22] Kosaraju, R., Decidability of Reachability in Vector Addition Systems, *Proceedings of the 14th Annual ACM Symposium on Theory of Computing* (1982), pp. 267-280.

[23] Landweber, L., Decision Problems for ω-Automata, *Math. Syst. Theory 3* (1969), 376-384.

[24] Landweber, L. and Robertson, E., Properties of Conflict-Free and Persistent Petri Nets, *JACM 25*, 3 (1978), pp. 352-364.

[25] Lehman, D., Pnueli, A., and Stavi, J., Impartiality, Justice, and Fairness: The Ethics of Concurrent Termination, *Proceedings of the 8th International Colloquium on Automata, Languages, and Programming*, LNCS 115 (1981), 264-277.

[26] Lichtenstein, O., and Pnueli, A., Checking that Finite State Concurrent Programs Satisfy their Linear Specification, *Proceedings of the 12th Annual ACM Symposium on Principles of Programming Languages* (1985), 97-107.

[27] Lichtenstein, O., Pnueli, A., and Zuck, L., The Glory of the Past, *Proceedings of the Workshop on Logics of Programs* (1985), 196-218.

[28] Manna, Z., and Pnueli, A., The Modal Logic of Programs, *Proceedings of the 6th International Colloquium on Automata, Languages, and Programming*, LNCS 71 (1979), 385-410.

[29] Mayr, E., An Algorithm for the General Petri Net Reachability Problem, *SIAM J. Comput. 13*, 3 (1984), pp. 441-460. A preliminary version of this paper was presented at the *13th Annual Symposium on Theory of Computing*, 1981.

[30] Mayr, E., Persistence of Vector Replacement Systems is Decidable, *Acta Informatica 15* (1981), pp. 309-318.

[31] Minsky, M., Recursive Unsolvability of Post's Problem of 'Tag' and Other Topics in the Theory of Turing Machines, *Annals of Mathematics 74* (1961), 437-455.

[32] Müller, H., Decidability of Reachability in Persistent Vector Replacement Systems, *Proceedings of the 9th Symposium on Mathematical Foundations of Computer Science* , LNCS 88 (1980), pp.

426-438.

[33] Owicki, S., and Lamport, L., Proving Liveness Properties of Concurrent Programs, *ACM Trans. on Programming Languages and Syst. 4* (1982), 455-495.

[34] Peterson, J., *Petri Net Theory and the Modeling of Systems,* (Prentice Hall, Englewood Cliffs, NJ, 1981).

[35] Pnueli, A., and Koren, T., There Exist Decidable Context-Free Propositional Dynamic Logics, *CMU Workshop on Logics of Programs*, LNCS 164, (1983).

[36] Pnueli, A., The Temporal Logic of Programs, *Proceedings of the 19th Annual Symposium on Foundations of Computer Science* (1977).

[37] Queille, J., and Sifakis, J., Fairness and Related Properties in Transition Systems--A Temporal Logic to Deal with Fairness, *Acta Informatica 19* (1983), 195-220.

[38] Rosier, L. and Yen, H., Logspace Hierarchies, Polynomial Time and the Complexity of Fairness Problems Concerning ω-Machines, *Proceedings of the 3rd Annual Symposium on Theoretical Aspects of Computer Science,* LNCS 210 (1986), pp. 306-320. To appear in *SIAM J. Comput.*

[39] Rosier, L. and Yen, H., On the Complexity of Deciding Fair Termination of Probabilistic Concurrent Finite-State Programs, *Proceedings of the 13th International Colloquium on Automata, Languages and Programming*, LNCS 226 (1986), 334-343.

[40] Sistla, A., and Clarke, E., The Complexity of Propositional Linear Temporal Logic, *JACM 32* (1985), 733-749.

[41] Suzuki, I., Fundamental Properties and Applications of Temporal Petri Nets, *Proceedings of the 19th Annual Conference on Information Sciences and Systems*, The Johns Hopkins University (1985), 641-646.

[42] Valk, R. and Vidal-Naquet, G., Petri Nets and Regular Languages, *J. of Computer and System Sciences 23* (1981), pp. 299-325.

[43] Valk, R., and Jantzen, M., The Residue of Vector Sets with Applications to Decidability Problems in Petri Nets, *Acta Informatica 21* (1985), 643-674.

[44] Vardi, M., Automatic Verification of Probablistic Concurrent Finite-State Programs, *Proceedings of the 26th Annual Symposium on Foundations of Computer Science* (1985), 327-338.

[45] Yamasaki, H., On Weak Persistency of Petri Nets, *Information Processing Letters 13*, 3 (1981), pp. 94-97.

[46] Zuck, L., Past Temporal Logic, Ph.D. Thesis, The Weizmann Institute of Science, Rehovot, Isreal, August, 1986.