MODEL CHECKING UNDER GENERALIZED FAIRNESS CONSTRAINTS

E. Allen Emerson* and Chin Laung Lei

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

TR-84-20 June 1984

*Work supported in part by NSF Grant MCS8302878.

Abstract

In this paper, we consider whether a given finite state concurrent system meets a specification (p_0, Φ_0), where p_0 is a functional assertion expressed in a branching time logic, and Φ_0 is the the underlying fairness assumption expressed in terms of infinitary linear temporal operators $\overset{\infty}{F}p$ and $\overset{\infty}{G}p$. We will give an efficient algorithm for this problem when the fairness constraint Φ_0 is an appropriate "normal form". We will also show that the problem is, in general, NP-complete.

1. Introduction

In [CES83] an efficient algorithm is given for verifying that the global transition graph of a finite state concurrent system is a model of a specification expressed in a propositional branching time temporal logic. For finite state systems, the model checking algorithm provides a mechanizable alternative to the traditional approach to concurrent program verification where a proof of correctness is constructed by hand using various axioms and inference rules. An extension to handle certain restricted types of fairness is also considered.

In this paper, we present a model checking method to handle generalized fairness. In particular, we consider the *Model Checking Problem* (FMCP) for *Fair Computation Tree Logic* (FCTL). FCTL is a branching time system which generalizes the (ordinary) CTL of [EC82], [EH82] and [CES83] by having all path quantifiers relativized to a fairness assumption Φ_0 composed of arbitrary boolean combinations of the infinitary linear time operators $\overset{\infty}{F}p$ ("infinitely often p") and $\overset{\infty}{G}p$ ("almost everywhere p"). Its basic modalities are thus of the form A_Φ ("for all fair paths") or E_Φ ("for some fair path") followed by a single linear time operator: Fp ("sometimes p"), Gp ("always p"), Xp ("nexttime p"), or [p U q] ("p holds until q becomes true"). The infinitary operators of Φ_0 make it possible to express and reason under a wide variety of "practical" fairness assumptions from the literature including weak fairness ([LA80], [PA80], [QS82]) ($\Phi_0 = \wedge_{i=1}^{n}(\overset{\infty}{G}\text{enabled}_i \Rightarrow \overset{\infty}{F}\text{executed}_i)$), strong fairness ([EC80], [LA80], [PA80], [QS82]) ($\Phi_0 = \wedge_{i=1}^{n}(\overset{\infty}{F}\text{enabled}_i \Rightarrow \overset{\infty}{F}\text{executed}_i)$), and impartiality [LPS81] ($\Phi_0 = \wedge_{i=1}^{n}(\overset{\infty}{F}\text{executed}_i)$). Only fairness of the last type is considered in [CES83].

We will first argue that FMCP can be efficiently reduced to the Fair State Problem (FSP): Starting from which states is there some path along which Φ_0 holds? We then show that if Φ_0 is of the form $\bigwedge_{i=1}^{n}(\overset{\infty}{F}p_i \vee \overset{\infty}{G}q_i)$, FSP and FMCP for FCTL can be solved in time linear on the size of the input specification and input structure. Since it turns out that many practical types of fairness used in the literature can be succinctly

expressed using such a Φ_0, a verification method based on our model checking algorithm is potentially of wide applicability. Finally we will show that FSP and FMCP for arbitrary Φ_0 composed of $\overset{\infty}{F}$ p and $\overset{\infty}{G}$p are NP-complete.

The remainder of the paper is organized as follows: Section 2 describes the syntax and semantics of our specification language -- FCTL. In section 3, the model checking problem and fair path problem are defined, and their complexity is analyzed. Applications and examples are give in section 4.

2. The Specification Language

A Fair Computation Tree Logic (FCTL) specification (p_0, Φ_0) consists of a functional assertion p_0 and an underlying fairness assumption Φ_0. The functional assertion p_0 is expressed in essentially CTL syntax with basic modalities of the form either A_Φ ("for all fair paths"), or E_Φ ("for some fair path") followed by one of the linear time operators Fp ("sometimes p"), Gp ("always p"), Xp ("nexttime p"), or [p U q] ("p holds until q becomes true"). We subscript the path quantifiers with the symbol Φ to emphasize that they range over paths meeting the fairness constraint Φ_0, and to syntactically distinguish FCTL from CTL. A fairness constraint Φ_0 is build up from atomic propositions, the infinitary linear time operators $\overset{\infty}{F}$p ("infinitely often p") and $\overset{\infty}{G}$p ("almost always p"), and boolean connectives. Note that p_0 is a state formula (true or false of states) whereas Φ_0 is a path formula (true or false of paths).

2.1. Syntax

Formally, the class of functional assertions in FCTL specifications is defined inductively as follows:

1. Any atomic proposition P is a functional assertion.
2. If p, q are functional assertions then so are $\neg$p, and (p $\wedge$ q).
3. If p, q are functional assertions then so are A_ΦXp, E_ΦXp, A_Φ(pUq), and E_Φ(pUq).

A propositional formula is one formed by rules 1, 2 above. A fairness constraint is then formed by the following rules:

4. If p, q are propositional formulae then $\overset{\infty}{F}$p, $\overset{\infty}{G}$p are fairness constraints.
5. If p, q are fairness constraints then so are $\neg$p, and (p $\wedge$ q).

The other connectives can then be defined as abbreviations in the usual way: p $\vee$ q abbreviates $\neg(\neg p \wedge \neg q)$, p $\Rightarrow$ q abbreviates $\neg p \vee q$, A_ΦFp abbreviates A_Φ(true U p),

$E_{\Phi}Fp$ abbreviates $E_{\Phi}(\text{true U } p)$, $A_{\Phi}Gp$ abbreviates $\neg E_{\Phi}F\neg p$, etc.

Remark: Recall that CTL* is the full branching time logic in which the basic modalities have the form: A or E followed by an arbitrary combination (involving boolean connectives and nesting) of linear time operators F, G, X, and U. We could thus view the assertions of FCTL as a sublanguage of CTL* where, eg., the $A_{\Phi}Fp$ is an abbreviation for the CTL* formula $A[\Phi_0 \Rightarrow Fp]$. However, the corresponding CTL* formula might be rather unwieldy due to the need to repeatedly write down multiple copies of the actual fairness formula Φ_0.

2.2. Semantics

Let AP be the underlying set of atomic propositions. A structure M=(S, R, L) is a labeled transition graph, where

- S is a nonempty set of states.
- R is a binary relation on S which gives the possible transitions between states.
- L: $S \rightarrow 2^{AP}$, is a labeling which assigns to each state a set of atomic propositions (intuitively, the propositions true at that state).

The size of a structure M=(S, R, L), written |M|, is defined to be the sum of the number of states in S and the number of transitions in R.

A *fullpath* x is an infinite sequence of states $(s_0, s_1, s_2, \ldots)$ such that $\forall i \geq 0[(s_i, s_{i+1})] \in R]$. We use x^i to denote the suffix of x beginning at state x_i, i.e. $x^i=(x_i, x_{i+1}, \ldots)$. Sometimes we use "computation sequence" instead of "fullpath". We write $M, x|=\Phi_0$ to mean that fullpath x in structure M meets fairness constraint Φ_0. We define the |= relation inductively in the usual way:

1. $M, x|=P$ iff $P \in L(s_0)$, for any atomic proposition P.
2. $M, x|=\neg p$ iff not$(M, x|=p)$
3. $M, x|=p \wedge q$ iff $M, x|=p$ and $M, x|=q$
4. $M, x|=\overset{\infty}{F} p$ iff there exists infinitely many $i \geq 0$ such that $M, x^i |= p$
5. $M, x|=\overset{\infty}{G} p$ iff $\exists i \geq 0[\forall j \geq i(M, x^j|=p)]$

We write $M, s|=E\Phi_0$ if there is a fullpath x starting at s such that $M, x|=\Phi_0$. We say that x is a fair path in structure M under fairness assumption Φ_0 if $M, x|=\Phi_0$ holds. A state is fair iff it lies on some fair path. A substructure C of M is called a fair component if C is strongly connected and contains some fair path.

An FCTL specification (p_0, Φ_0) is interpreted with respect to a structure M. We write $M, s|=_{\Phi_0} p_0$ to mean that functional assertion p_0 is true at state s of structure M under fairness assumption Φ_0. We define $|=_{\Phi_0}$ inductively on the structure of the functional assertion p_0:

1. $M, s|=_{\Phi_0} P$ iff $P \in L(s)$, for any atomic proposition P.
2. $M, s|=_{\Phi_0} \neg p$ iff not$(M, s|=_{\Phi_0} p)$
3. $M, s|=_{\Phi_0} p \wedge q$ iff $M, s|=_{\Phi_0} p$ and $M, s|=_{\Phi_0} q$
4. $M, s_0|=_{\Phi_0} E_{\Phi} Xp$ iff there exists a path $x=(s_0, s_1, s_2, \ldots)$ such that $M, x|=\Phi_0$, and $M, s_1|=_{\Phi_0} p$
5. $M, s_0|=_{\Phi_0} A_{\Phi} Xp$ iff for all paths $x=(s_0, s_1, s_2, \ldots)$ $[M, x|=\Phi_0 \Rightarrow M, s_1|=_{\Phi_0} p]$
6. $M, s_0|=_{\Phi_0} E_{\Phi}(p \ U \ q)$ iff there exists a path $x=(s_0, s_1, s_2, \ldots)$ such that $M, x|=\Phi_0$ and $\exists j \geq 0[M, s_j|=_{\Phi_0} q \wedge \forall i<j(M, s_i|=_{\Phi_0} p)]$
7. $M, s_0|=_{\Phi_0} A_{\Phi}(p \ U \ q)$ iff for all paths $x=(s_0, s_1, s_2, \ldots)$, $M, x|=\Phi_0$ implies $\exists j \geq 0[M, s_j|=_{\Phi_0} q \wedge \forall i<j(M, s_i|=_{\Phi_0} p)]$

We say that an FCTL specification (p_0, Φ_0) is valid provided that for each structure $M=(S, R, L)$, and each state $s \in S$, $M, s|=_{\Phi_0} p_0$ holds. An FCTL specification (p_0, Φ_0) is satisfiable provided that for some structure $M=(S, R, L)$ and some state $s \in S$, $M, s|=_{\Phi_0} p_0$. In the later case, we say that M is a model for (p_0, Φ_0).

If (p_0, Φ_0) is an FCTL specification then the set of subformulae of functional assertion p_0, written $SF(p_0)$, is the minimal set satisfying the following conditions:

1. $p_0 \in SF(p_0)$
2. $\neg p \in SF(p_0) \Rightarrow p \in SF(p_0)$
3. $p \wedge q \in SF(p_0) \Rightarrow p, q \in SF(p_0)$
4. $E_{\Phi} Xp \in SF(p_0) \Rightarrow p \in SF(p_0)$
5. $A_{\Phi} Xp \in SF(p_0) \Rightarrow p \in SF(p_0)$
6. $E_{\Phi}(p \ U \ q) \in SF(p_0) \Rightarrow p, q, E_{\Phi} XE_{\Phi}(p \ U \ q) \in SF(p_0)$
7. $A_{\Phi}(p \ U \ q) \in SF(p_0) \Rightarrow p, q, A_{\Phi} XA_{\Phi}(p \ U \ q) \in SF(p_0)$

Note that $|SF(p_0|=O(|p_0|)$, where $|SF(p_0)|$ denotes the number of elements in $SF(p_0)$, and $|p_0|$ denotes the length of the formula p_0.

3. Model Checking Problem and Fair State Problem

The Model Checking Problem for FCTL (FMCP) is: Given a structure M=(S, R, L), and an FCTL specification (p_0, Φ_0), determine for each state s∈S whether M, $s|=_{\Phi_0} p_0$. The Fair State Problem (FSP) is: Given a structure M=(S, R, L), and a fairness constraint Φ_0, determine for each state s∈S whether there is a fullpath x in M starting at s such that M, $x|=\Phi_0$.

3.1. Reduction of FMCP to FSP

Since the FSP condition is equivalent to M, $s|=_{\Phi_0} E_\Phi Xtrue$, FSP may be viewed as a special case of FMCP. However, we can generalize a method in [CES83] to reduce FMCP to FSP. The reduction yields an algorithm for for FMCP that runs in time linear in the size of the input (specification and structure) and the time to solve FSP. The reduction exploits the observation that, for any fairness constraint Φ_0 and for any fullpaths x and y such that x is a suffix of y, M, $x \models \Phi_0$ iff M, $y \models \Phi_0$. We thus get the following equivalences:

(1) M, $s|=_{\Phi_0} E_\Phi Xp$ iff $\exists(s, t)\in R[(M, t|=E\Phi_0) \wedge (M, t|=_{\Phi_0} p)]$

(2) M, $s|=_{\Phi_0} A_\Phi Xp$ iff $\forall(s, t)\in R[(M, t|=E\Phi_0) \Rightarrow (M, t|=_{\Phi_0} p)]$

(3) M, $s|=_{\Phi_0} E_\Phi[p\ U\ q]$ iff there is a finite path of nodes satisfying p leading to a fair node which satisfies q.

(4) M, $s|=_{\Phi_0} A_\Phi[p\ U\ q]$ iff M, $s|=_{\Phi_0} \neg(E_\Phi(\neg q\ U\ (\neg p \wedge \neg q)) \vee E_\Phi G(\neg q))$

The equivalences (1) and (2) are immediate; (3) is just a restatement of the definition. To check $A_\Phi[p\ U\ q]$, equivalence (4) shows that we can first check whether $E_\Phi(\neg q\ U\ (\neg p \wedge \neg q))$ using equivalence (3). To next check $E_\Phi G(\neg q)$, we let M' be the substructure of M obtained by deleting all nodes where q holds (inductively, we assume nodes of M are labeled with the true subformulae). Then $E_\Phi G(\neg q)$ holds at a node s iff there is a finite path from s to a fair node t in M'. Detection of fair nodes is done by the algorithm for FSP. The reduction algorithm (we call it AFMCP) is described in greater detail below (Figure 1).

We claim that when AFMCP terminates, the following assertion holds: $\forall f\in SF(p_0)[f\in label(s)$ iff M, $s|=_{\Phi_0} f\ ]$. The proof is by induction on the structure of f. The details are left to the reader.

Let $T_A(M, \Phi_0)$ denote the time complexity of algorithm AFSP(M, Φ_0). Then we calculate that AFMCP requires time $O(|p_0|\cdot\max(|M|,T_A(M, \Phi_0)))$. To see this note that steps 1, 2 are executed only once and require time $O(T_A(M, \Phi_0)+O|M|)$. Moreover, step 3 is a for loop which is executed $|SF(p_0)|$ times and each case in the loop requires time $O(|M|)$ except the case where $f=E_\Phi[p\ U\ q]$ which requires time $O(|M|)+T_A(M, \Phi_0)$. Hence, step 3 requires time less than $|SF(p_0)|\cdot\max(O(|M|),T_A(M, \Phi_0))=O(|p_0|\cdot\max(|M|, T_A(M, \Phi_0)))$, so does the whole algorithm.

3.2. Efficient Algorithm for Fair State Problem

In many interesting cases, FSP (and hence FMCP) can be solved in time linear in the size of the input structure and specification. We now consider a normal form for these cases: $\Phi_0=\bigwedge_{i=1}^{n}(\overset{\infty}{F} p_i \vee \overset{\infty}{G} q_i)$. Since it turns out that almost all practical fairness notions can be succinctly expressed by a Φ_0 of the form, we believe that FCTL restricted to such fairness constraints is still of great applicability. We will develop the linear time algorithm for FMCP under such a restriction in the coming paragraphs.

We first show how detect fair components (recall that a strongly connected structure is fair iff there is a fair path in it). Given a strongly connected structure C=(S, R, L), and a fairness constraint $\Phi_0=\bigwedge_{i=1}^{k}(\overset{\infty}{F} p_i \vee \overset{\infty}{G} q_i)$, the algorithm described in Figure 2 can decide whether C is a fair component w.r.t. Φ_0 in time $O(|C|\cdot|\Phi_0|)$.

The proof that C is a fair component w. r. t. Φ_0 iff the recursive function AFC(C, Φ_0) returns true is by induction on the number of the conjuncts k in Φ_0.

Basis: k=0, Φ_0=true, and the program AFC returns true immediately. Hence the hypothesis holds. (Note that any strongly connected component is fair w.r.t. "true".)

Induction step: Assume true for k<n, prove for k=n as follows:

[Only if part]: If AFC returns true, then it must do so either at statement (6) or statement (8). Case 1: AFC returns true at statement (6). By induction hypothesis, at least one of the strongly components in C' is fair w.r.t. Φ_0'. Let D be one of such strongly connected components. Since D is contained in C' and every nodes of C' satisfies q_i, every path in D satisfies $\overset{\infty}{G} q_i$. Hence D is also a fair component w.r.t. to Φ_0. Hence C is a fair component.

Case 2: AFC returns true at statement (8). In this case, some node in C satisfies p_i $\forall i\in[1,n]$. Hence any cycle in C which includes all nodes of C is a fair path w.r.t. Φ_0

(because C is strongly connected, there exists at least one such path). Let x be one such cycle; it's obvious that M, x$|=\Phi_0$. Hence C is a fair component.

[If part]: Assume that C is a fair component, we prove that AFC will return true either at statement (6) or at statement (8). (The following argument is essentially the reverse of the previous proof.)

Case 1: $\forall j\in[1,n](\exists s\in S(C, s|=_{\Phi_0} p_j))$. In this case the condition of statement (5) is always false. Hence the program will terminate at statement (8).

Case 2: $\exists j\in[1,n](\forall s\in S(\text{not } C, s|=_{\Phi_0} p_j))$. Let i be the smallest integer such that $\forall s\in S(\text{not } C, s |=_{\Phi_0} p_j)$. Since C is fair, C contains some fair cycle x w.r.t. Φ_0. Every node on x must satisfy q_i. Hence x must be included in some strongly connected component D of C'. By induction hypothesis, AFC(D, Φ_0') will return true, and so will FC(C, Φ_0).

To analyze the complexity, let m=|C|, n=|Φ_0|, and k= the number of of conjuncts of Φ_0. Define function T(i, j) to be the time complexity of AFC when the size of input structure is i and the input fairness constraint has k conjuncts. Let X={D_1, ... D_l} be the set of strongly connected components of C'. If we let d_i denote $|D_i|$, then $\sum_{i=1}^{l} d_i \leq |C'| \leq m$. Clearly, T(m, 0)=O(1) since the program AFC returns true immediately. Note that for any recursive call each statement in AFC can be executed at most k times. Furthermore, the compound statement beginning at (5) can be executed at most once (because it always returns control to the caller). Hence we have the following recurrence relation:

$$T(m, k)\leq \sum_{i=1}^{k} O(m\cdot|p_i|) + \sum_{i=1}^{l} T(d_i, k\text{-}1)$$

Which can be simplified to $T(m, k)\leq O(m\cdot n) + \sum_{i=1}^{l} T(d_i, k\text{-}1)$. By induction on k, we can easily show that $T(m, k)\leq O(m\cdot n)$. The details are left to the reader.

The program AFS(M, Φ_0, S') of Figure 3 is an algorithm for FSP of time complexity $O(|M|\cdot|\Phi_0|)$. The for-loop checks fair components in M, if a state s is in a fair component then it is a fair state. The while-loop picks out nodes which can reach a fair state. It is clear that a state which can reach another fair state is a fair state. So the algorithm picks fair states. Conversely, for any fair state s in M, if s lives on a fair component then it will be put in S' during the execution of the for-loop ; otherwise, s can reach a fair state and will be put into S' during the execution of the while-loop.

The complexity bound follows from the complexity analysis of algorithm AFC. To

see this, assume that M=(S, R, L) can be partitioned into l strongly connected components C_1, C_2, ... C_l. Then each step of the for-loop requires time $|AFC(C_i,\Phi_0)|+O|C_i|$ which is equal to $O(|C_i|\cdot|\Phi_0|)$. Hence the for-loop requires time $O(|M|\cdot|\Phi_0|)$. The while loop requires only O(|M) time, so the whole algorithm takes only $O(|M)\cdot|\Phi_0|)$ time.

We have thus established,

Theorem 1: FMCP for input structure M=(S, R, L), and input specification (p_0, Φ_0) with $\Phi_0=\bigwedge_{i=1}^{n}(\overset{\infty}{F} p_i \vee \overset{\infty}{G} q_i)$ can be solved in time $O(|p_0|\cdot|M|\cdot|\Phi_0|)$.

3.3. Complexity of The General Case

We show, in this section, that FSP (and hence also FMCP) is NP-complete for general fairness specification Φ_0.

Theorem 2: FSP is NP-complete.

Proof: We will reduce 3-SAT to FSP, with fairness constraint of the form $\bigwedge_{i=1}^{n}(\overset{\infty}{G}\neg p_i \vee \overset{\infty}{G}\neg q_i)$.

Given a formula g in 3-CNF with n variables and m factors, we show how to construct, in polynominal time, a structure M=(S, R, L) with a designated state s∈S, and a fairness constraint Φ_0 such that there is a path z in M starting from s and M, z $\models \Phi_0$ iff g is satisfiable.

Let x_1, x_2, ... x_n and C_1, C_2, ... C_m be the variables and factors of g (ie. $g=\bigwedge_{i=1}^{m} C_i$), where $C_i=l_{i1} \vee l_{i2} \vee l_{i3}$ for $1\leq i\leq m$, and $l_{ij}=x_k$ or $\neg x_k$ for some k∈[1,n]. Take $AP=\{p_1, p_2, \ldots p_n, q_1, q_2, \ldots q_n\}$ as the underlying set of atomic propositions. Construct a structure M=(S, R, L) as follows:

$S=(s,t) \cup \{v_{ij}: 1\leq i\leq m, 1\leq j\leq m\}$

$R=\{(s,v_{1j}): 1\leq j\leq 3\} \cup \{(v_{mj},t): 1\leq j\leq 3\} \cup \{(t, s)\}$

$\quad \cup \{(v_{ij}, v_{i+1,k}): 1\leq i\leq m\text{-}1 \text{ and } 1\leq j, k\leq 3\}$

$L(s)=L(t)=\emptyset$

$L(v_{ij})=\{p_k\}$ if $l_{ij}=x_k$

$\quad =\{q_k\}$ if $l_{ij}=\neg x_k$

Let $\Phi_0=\bigwedge_{i=1}^{n}(\overset{\infty}{G}\neg p_i \vee \overset{\infty}{G}\neg q_i)$.

Structure M is shown in Figure 4. It is quite clear that the above construction can be

done in polynomial time. We claim that g is satisfiable iff there is some path z in M starting from s such that M, z $\models \Phi_0$. Proof of the claim is given in the appendix.

[Membership]: It has already been shown in [SC82] that the model checking problem for linear time temporal logic with F, and G operators can be solved in NP time, Hence FSP is in NP, thus FSP is NP-complete. In [SC82] it was shown that, in effect, FSP for any arbitrary linear time formula over F, G is NP-complete. For FSP with Φ_0 of the type we construct, membership in NP follows since our language of fairness constraints may be viewed as a sublanguage of linear time logic by the equivalences $\overset{\infty}{F}p \equiv GFp$ and $\overset{\infty}{G}p \equiv FGp$. But NP-hardness for Φ_0 of our type does not follow from the proof in [sc82] which used a different reduction to a formula $Fp_1 \wedge \ldots \wedge Fp_n$. Because Fp is not expressible in our Φ_0 language, that proof cannot be applied. Since our Φ_0 language has a more restricted syntax, its decision problem might be easier. Our NP-hardness argument shows that such is not the case: even a simple Φ_0 of the form $\overset{\infty}{G}p \vee \overset{\infty}{G}q$ can be "hard". This is surprising since the dual property $\overset{\infty}{F}p \wedge \overset{\infty}{F}q$ is "easy".

Corollary: FMCP is NP-complete.

Proof: NP-hardness follows directly from Theorem 2, and NP-membership follows from the reduction algorithm in Figure 1 and the Theorem 2.

4. Applications

Our canonical form fairness specification $\Phi_0 = \bigwedge_{i=1}^{n} (\overset{\infty}{G}p_i \vee \overset{\infty}{F}q_i)$ allows us to perform model checking efficiently under many commonly used "practical" fairness assumptions and a wide variety of fairness related notions. To handle a given fairness-type notion using the canonical Φ_0 form, it is often helpful to apply the following equivalences:

(1) $E(p \vee q) \equiv Ep \vee Eq$

(2) $\overset{\infty}{F}(p \vee q) \equiv \overset{\infty}{F}p \vee \overset{\infty}{F}q$

(3) $\overset{\infty}{G}(p \wedge q) \equiv \overset{\infty}{G}p \wedge \overset{\infty}{G}q$

Note that a consequence of (1) is that any Φ_0 of the form $\bigvee_{i=1}^{n} \bigwedge_{j=1}^{n_i} (\overset{\infty}{G}p_{ij} \vee \overset{\infty}{F}q_{ij})$ can be handled with linear time complexity as well because $E\Phi_0 \equiv \bigvee_{i=1}^{n} E \bigwedge_{j=1}^{n_i} (\overset{\infty}{G}p_{ij} \vee \overset{\infty}{F}q_{ij})$. To see if a node is fair w.r.t. Φ_0, one merely checks if it is fair w.r.t. one of the disjuncts of Φ_0. Such a Φ_0 may thus viewed as a generalized canonical form for doing model checking

in linear time.

4.1. Expressible Fairness Notions

We can succinctly express the following fairness notions using our generalized canonical form (with a liberal interpretation of the meanings of atomic propositions):

1. Impartiality: $\bigwedge_{i=1}^{n} (\overset{\infty}{\mathrm{F}}\, executed_i)$
2. Weak Fairness: $\bigwedge_{i=1}^{n} (\overset{\infty}{\mathrm{G}} enabled_i \Rightarrow \overset{\infty}{\mathrm{F}}\, executed_i)$
   $\equiv \bigwedge_{i=1}^{n} (\overset{\infty}{\mathrm{F}} (\neg enabled_i) \vee \overset{\infty}{\mathrm{F}}\, executed_i)$
   $\equiv \bigwedge_{i=1}^{n} (\overset{\infty}{\mathrm{F}} (\neg enabled_i \vee executed_i))$
3. Strong Fairness: $\bigwedge_{i=1}^{n} (\overset{\infty}{\mathrm{F}}\, enabled_i \Rightarrow \overset{\infty}{\mathrm{F}}\, executed_i) \equiv \bigwedge_{i=1}^{n} (\overset{\infty}{\mathrm{G}} \neg enabled_i \vee \overset{\infty}{\mathrm{F}}\, executed_i)$

The following definitions of acceptance for finite automata on infinite strings are closely related to fairness and can also be readily expressed:

4. Buchi Acceptance: $\overset{\infty}{\mathrm{F}}\, Green$
5. Pairs Acceptance: $\bigvee_{i=1}^{n} (\neg \overset{\infty}{\mathrm{F}}\, Red_i \wedge \overset{\infty}{\mathrm{F}}\, Green_i) \equiv \bigvee_{i=1}^{n} (\overset{\infty}{\mathrm{G}} (\neg Red_i) \wedge \overset{\infty}{\mathrm{F}}\, Green_i)$
6. Complemented Pairs Acceptances: $\bigwedge_{i=1}^{n} (\overset{\infty}{\mathrm{F}}\, Red_i \Rightarrow \overset{\infty}{\mathrm{F}}\, Green_i)$
   $\equiv \bigwedge_{i=1}^{n} (\overset{\infty}{\mathrm{G}} \neg Red_i \vee \overset{\infty}{\mathrm{F}}\, Green_i)$

4.2. Extended Semantics with Arc Labels

There was a point glossed over in our rendering of the fairness properties above, whereas the enabling condition for performing a step of process i is properly viewed as a predicate on states (i.e. nodes), the actual execution of the step is more naturally modeled as a transition (i.e. traversal of an arc). To allow a precise differentiation between execution of transition actions and enabling of state conditions, we can extend the semantics of FCTL so that a structure $M=(S,A_1,A_2,...,A_p,L)$ where each $A_i \subseteq S \times S$ represents (the atomic actions of) process i, and where we think of each each arc $(s_1,s_2) \in A = A_1 \cup ... \cup A_p$ as being labeled with the set $\{i: (s_1,s_2) \in A_i\}$ of processes which can cause a transition from state s_1 to state s_2. We can now extend the fairness specifications to allow atomic assertions: $executed_i$ hold at (s_1,s_2) iff $(s_1,s_2) \in A_i$. The fairness

specifications such as $\overset{\infty}{F}$ *enabled*$_i \Rightarrow \overset{\infty}{F}$ *executed*$_i$ can be given a rigorous definition. It is straightforward to formalize this approach and to extend our efficient model checking algorithm to the extended semantics, but the details are tedious.

As an alternative, we can encode the extended semantics with arc labels into the original semantic framework. One way to do this involves duplication of nodes: Given an extended structure $M=(S,A_1,...A_p,L)$ we can form an ordinary structure $M'=(S',A',L')$ where $S'=\{(s,i)\in S\times[0:p]$: $s\in S$ and [(i=0 and s has no predecessor in M) or ($i\in[1:p]$ and s is an A_i-successor for some state t in M)]}, $A'=\{((s,i),(t,j))\in S'\times S'$: $(s,t)\in A_j\}$, and for $i\in[1:p]$, $L'((s,i))=L(s)\cup\{$*executed*$_i\}$ where *executed*$_i$ is a distinguished proposition while for i=0, $L'((s,0))=L(s)$. Intuitively, (s,i) is a copy of state s reached by a transition of process i. See Figure 5. Also note that one potential drawback of this method is that the size of the graph representation of the new structure can be p^2 times the size of the old structure, where p is the number of processes. This method is similar to that of [Pn77] and [CES83].

Another way to encode the extended semantics into the original framework is to insert an intermediate node "in the center" of each arc. Each such inserted node is labeled with a distinguished proposition *Arc* plus the process numbers labeling its corresponding arc. The label of each original node is augmented with a distinguished proposition *Node*. Formally, if $M=(S,A_1,...,A_p,L)$ then the new structure $M'=(S',A',L')$ where, letting $A=A_1\cup...\cup A_p$, $S'=S\cup A$, $A'=\{(s,(s,t))$: $s\in S$ and $(s,t)\in A\}\cup\{((s,t),t)$: $(s,t)\in A$ and $t\in S\}$, $L'(s)=L(s)\cup\{$*Node*$\}$ for $s\in S$, and $L'((s,t))=\{$*Arc*$\}\cup\{i\in[1:p]$: $(s,t)\in A_i\}$.

Let †: assertion of M → assertion of M' be a translation defined recursively as follows:

(1) $P\dagger = P$ where P is an atomic proposition
(2) $(\neg p)\dagger = \neg p\dagger$
(3) $(p\wedge q)\dagger = p\dagger\wedge q\dagger$
(4) $(E_\phi Xp)\dagger = E_\phi XE_\phi Xp\dagger$
(5) $(A_\phi Xp)\dagger = A_\phi XA_\phi Xp\dagger$
(6) $[E_\phi(pUq)]\dagger = E_\phi[(Node\Rightarrow p\dagger)U(Node\wedge q\dagger)]$

We then rewrite the original specification according to the translation †. The method is illustrated in Figure 6. One advantage of this encoding is that the size of (the graph representing) the new structure does not depending on the number of processes; rather, the increase in size is linear. If the graph representing the original structure has v nodes and e arcs, the new structure's graph has v+e nodes and 2·e arcs. Of course, there is also a linear blowup in the size of the specification formula.

4.3. Examples

Example 1 (Dijkstra's random number generator): Figure 7.(a) is a guarded command program for a random number generator, the corresponding global state graph is given in Figure 7.(b), and Figure 7.(c) is the resulting graph of 7.(b) after applying the first encoding scheme discussed in the previous section. We want to show that under strong fair scheduling of processes, the program fairly terminates, i.e. the program terminates on every path satisfying the strong fair scheduling condition. We can express the above statement as an FCTL specification: $(p_o,\Phi_0)=(A_\Phi FA_\Phi X\mathit{false}, (\overset{\infty}{F}\mathit{enabled}_1\Rightarrow\overset{\infty}{F}\mathit{executed}_1) \wedge (\overset{\infty}{F}\mathit{enabled}_2\Rightarrow\overset{\infty}{F}\mathit{executed}_2)$, where Φ_0 is the fairness assumption which requires that if a process is enabled infinitely often then it must be executed infinitely often, and p_o is the functional assertion which asserts that along every fair path the program terminates. We then apply our model checking algorithm to (p_o, Φ_0) on the model in Figure 7.(c). The result is given in Figure 7.(d) which shows that p_o holds on every state of the structure. Hence the program in Figure 7.(a) fairly terminates.

Note that if we change the underlying fairness assumption in the above example to either weak fairness or impartiality, program 7.(a) still terminates fairly because the only infinite execution sequence is unfair w.r.t. all these three notions of fairness. However, if no fairness assumption is taken, the program might always execute process 1 and never terminates.

Example 2 (Mutual Exclusion): We illustrate our efficient model checking algorithm by considering a solution to the mutual exclusion problem for two processes P_1 and P_2. In the solution each process is always in exactly one of the three code regions:

N_i the Noncritical region.
T_i the Trying region.
C_i the Critical region.

A global state transition graph is given in Figure 8. Note that we only record transitions between different regions of code; internal moves within the same region are not considered.

To establish absence of starvation, we must show that $T_i\Rightarrow A_\Phi FC_i$ for each process i. Note that the solution is not starvation free under an unfair scheduler, for example, in the infinite execution sequence $s_0,s_1,s_4,s_7,s_1,s_4,s_7\ldots$ process 1 is in its trying region infinitely often but it never enters its critical region. We will show that the solution is starvation free under the weak fairness assumption $\Phi_0=(\overset{\infty}{G}\mathit{enabled}_1\Rightarrow\overset{\infty}{F}\mathit{executed}_1) \wedge (\overset{\infty}{G}\mathit{enabled}_2\Rightarrow\overset{\infty}{F}\mathit{executed}_2)$. Without loss of generality, we only consider the starvation

free property for process 1: $p_o = A_\Phi G(\neg T_1 \vee A_\Phi F C_1)$. The states of the global transition graph will be labeled with subformulae of p_o during execution of model checking algorithm. On termination every state will be labeled with $\neg T_1 \vee A_\Phi F C_1$, as shown in Figure 8.(b). Thus we we can conclude that $s_o \models_{\Phi_0} p_o$. It follows that process 1 cannot be prevented from entering its critical region once it has entered its trying region.

Appendix: Proof of Theorem 1 Continued

[only if part]: Assume that g is satisfiable. Since g is satisfiable, there exists a truth assignment A such that g is true under A, i.e. for any factor C_i, there is a literal l_{ij} in C_i such that l_{ij} is true under this particular truth assignment. Now consider a cycle z in M formed by nodes s, v_{1j_1}, ... v_{nj_n}, t, such that for all i, v_{ij_i} is true under the assignment A.

We will show that M, z $\models \Phi_0$ by showing that $\overset{\infty}{G}\neg p_k \vee \overset{\infty}{G}\neg q_k$ holds on z for every $k \in [1,n]$. If $\overset{\infty}{G}\neg p_k$ holds on z then $\overset{\infty}{G}\neg p_k \vee \overset{\infty}{G} q_k$ also holds on z. Hence, we only have to show that when $\overset{\infty}{G}\neg p_k$ does not hold on z, $\overset{\infty}{G}\neg p_k \vee \overset{\infty}{G}\neg q_k$ still holds on z. Because $\overset{\infty}{G}\neg p_k$ does not hold on z, there must be some node v in z such that $p_k \in L(v)$. Note that $L(s)=L(t)=\emptyset$. Hence v is v_{ij_i} for some $i \in [1,n]$. By the construction of the labeling function L, we conclude that $l_{ij_i} = x_k$. By the construction of z, l_{ij_i} is true, ie. the assignment A assigns true to x_k. Hence $\neg x_k$ is false under A. Clearly under A, $l_{i'j_{i'}} \neq \neg x_k$ for any $i \in [1,n]$. Again by the definition of L, $q_k \neq l_{i'j_{i'}}$, ie. $\overset{\infty}{G}\neg q_k$ holds on z. Hence $\overset{\infty}{G}\neg p_k \vee \overset{\infty}{G}\neg q_k$ holds on z for any $k \in [1,n]$. We conclude that Φ_0 holds on z.

[if part]: Assume that there is a path z in M starting from s such that Φ_0 holds on z. Let z' be a suffix of z starting from state s such that $\bigwedge_{i=1}^{n} (G\neg p_k \vee G\neg q_k)$ holds on z'. Note that either p_k or q_k does not appear on the label of any node on z'. Consider the truth assignment A: $x_k \rightarrow \{T, F\}$ as follows:

$$A(x_k) = T \quad \text{if } \exists i,j[\, p_k \in L(v_{ij})]$$
$$= F \quad \text{if } \exists i,j[\, q_k \in L(v_{ij})]$$

It is quite easy to check that A is consistent in the sense that A assigns a unique value to each x_k. Furthermore, the assignment caused by any $L(v_{ij})$ will guarantee that C_i is true under the assignment A. Hence g is satisfiable. This completes our proof.

REFERENCES:

[CES83] Clark, E. M., Emerson, E. A., and Sistla, A. P., Automatic Verification of Finite State Concurrent System Using Temporal Logic, 10th Annual ACM Symp. on Principles of Programming Languages, 1983

[EC80] Emerson, E. A., and Clarke, E. M., Characterizing Correctness Properties of Parallel Programs Using Fixpoints, Proc. ICALP 80, LNCS Vol. 85, Springer Verlag, 1980, pp. 169-181.

[CE82] Emerson, E. A., and Clarke, E. M., Using Branching Time Temporal Logic to Synthesize Synchronization Skeletons, Tech. Report TR-208, Univ. of Texas, 1982.

[EH82] Emerson, E. A., and Halpern, J. Y., Decision Procedures and Expressiveness in the Temporal Logic of Branching Time, 14th Annual ACM Symp. on Theory of Computing, 1982.

[GPSS80] Gabbay, D., Pnueli A., et al., On The Temporal Analysis of Fairness, POPL, 1980, pp. 163-173.

[LA80] Lamport, L., Sometimes is Sometimes "Not Never" - on the temporal logic of programs, POPL, 1980, pp. 174-185.

[LPS81] Lehmann. D., Pnuuli, A., and Stavi, J., Impartiality, Justice and Fairness: The Ethics of Concurrent Termination, ICALP 1981, LNCS Vol. 115, pp 264-277.

[PA80] Park, D., On The Semantics of Fair Parallelism, Abstract Software Specification, LNCS Vol. 86, Springer Verlag, 1980, pp. 504-524.

[PN77] Pnueli, A., The Temporal Logic of Programs, 19th annual Symp. on Foundations od Computer Science, 1977.

[QS82] Queille, J. P., and Sifkis, J., Fairness and Related Properties in Transition Systems, Research Report #292, IMAG, Grenoble, 1982.

[SC83] Sistla, A. P., and Clarke, E. M., The Complexity of Propositional Temporal Logic, 14 Annual ACM Symp. on Theory of Computing, 1982.

Let AFSP(M, Φ_0) be an algorithm for solving FSP which returns the set of fair states of the input structure M w.r.t. fairness constraint Φ_0.

```
procedure AFMCP(M, (p0, Φ0));
   /* M=(S, R, L) is the input structure, and (p0, Φ0) is the specification */
begin
1.      S' := AFSP(M, (p0, Φ0)) /* use algorithm AFSP to identify fair states in M */
2.      for each s∈S do if s∈S' then label(s) := {Φ0} else label(s) := {¬Φ0};
3.      for each formula f∈SF(p0) do /* Inductively, taking the shortest formula first. */
          case f of the form
3.1         atomic formula: for each s∈S do if f∈L(s) then label(s) := label(s) ∪ {f};
3.2         ¬p: for each s∈S do if f∉label(s) then label(s) := label(s) ∪ {f};
3.3         p ∧ q: for each s∈S do if p, q∈label(s) then label(s) := label(s) ∪ {f};
3.4         EΦXp: for each s ∈S do if ∃(s, t)∈R[p, Φ0∈label(t)]
                                      then label(s) := label(s) ∪ {f} ;
3.5         AΦXp: for each s ∈S do if ∀(s, t)∈R[Φ0∈label(s) ⇒ p∈label(s)]
                                      then label(s) := label(s) ∪ {f};
3.6         EΦ(p U q):
              EU := empty set;
              for each s∈S do
                  if q,Φ0 ∈label(s) then begin label(s) := label(s) ∪ {f}
                                               EU := EU ∪ {s}
                                         end;
              while EU ≠ ∅ do
                begin
                  let t be the first element of EU;
                  D := {s∈S: (s, t)∈R ∧ p∈label(s) ∧ q, EΦ(p U q)∉label(s);
                  for each s∈D do label(s) := label(s) ∪ {f};
                  EU := EU ∪ D \ {t};
                end of while;
3.7         AΦ(p U q):
              Label the states of M with ¬p, ¬q, ¬p ∧ ¬q by using 3.2 and 3.3.
              Label the states of M with EΦ[¬q U (¬p ∧ ¬q)} using 3.6.
              Label the states of M with ¬EΦ[¬q U (¬p ∧ ¬q)} using 3.2.
              S' := {s∈S: ¬q∈label(s)};
              M' := (S', R|S'xS', L|S');
              FS' := A(M', Φ0);
              for all s∈FS' do label(s) := label(s) ∪ {EΦG¬q};
              Label the states of M with ¬EΦG¬q;
              for all s∈S do
                 if ¬EΦG¬q, ¬EΦ[(¬q) U (¬p ∧ ¬q)]
                   then label(s) := label(s) ∪ {AΦ[p U q];
          end of cases;
end of procedure;
```

Figure 1: Reduction Algorithm

```
Recursive Boolean Procedure AFC(C, Φ0)
/* input: C=(S, R, L) is a strongly connected structure, and
          Φ0 = ⋀_{i=1}^{n} (F^∞ p_i ∨ G^∞ q_i) is a fairness constraint
   output: true - if C is a fair component
           false - otherwise */
begin
1    if k=0 then return(true);
2    for i:=1 to k do
       begin
3        p_occurs[i] := false;
4        for each s∈S do if C, s |=_{Φ0} p_i then p_occurs[i] := true;
5        if p_occurs[i]=false then
           begin
             Φ0' := ⋀_{j=1}^{i-1} (F^∞ p_j ∨ G^∞ q_j) ∧ ⋀_{j=i+1}^{k} (F^∞ p_j ∨ G^∞ q_j);
             C' := maximal substructure of C s.t. every states of C' satisfies q_i;
             X := {D: D is a maximal strongly connected component of C'};
6            for each D∈X do if FC(D, Φ0')=true then return(true);
7            returns(false)
           end
       end;
8    return(true)
end;
```

Figure 2: Fair Component Detection Algorithm

```
procedure AFS(M, Φ0, S');
/*  input: M=(S, R, L) is a structure, and
          Φ0 = ⋀_{i=1}^{n} (□◇p_i ∨ ◇□q_i)
   output: S' -- the set of fair states of structure M  */
begin
      S' := ∅;
      let X={C: C is a maximal strongly connected component of M};
      for each C∈X do if AFC(C, Φ0) then S' := S' ∪ {s: s is a state of C};

      /* calculate the set of states in S which can reach some state in S' */
      CLOSE := S';
      while CLOSE≠∅ do
      begin
            choose an arbitrary element t from CLOSE;
            D := {s: (s, t)∈R ∧ s∉S'};
            S' := S' ∪ D;
            CLOSE := CLOSE ∪ D \ {t}
       end;
end;
```

Figure 3: Algorithm for Calculating Fair States.

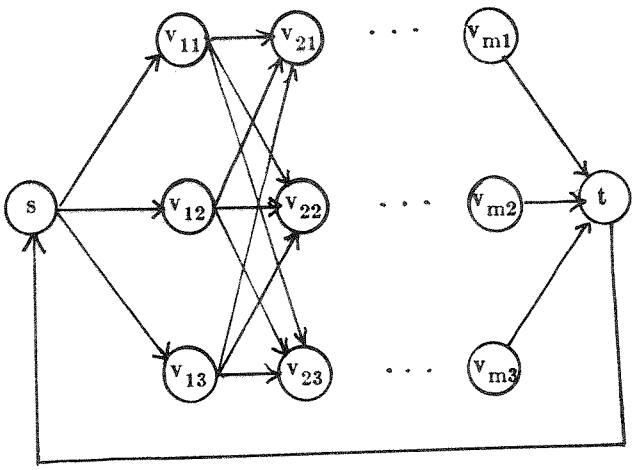
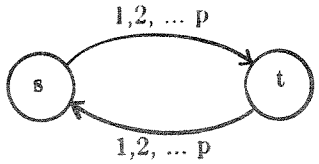
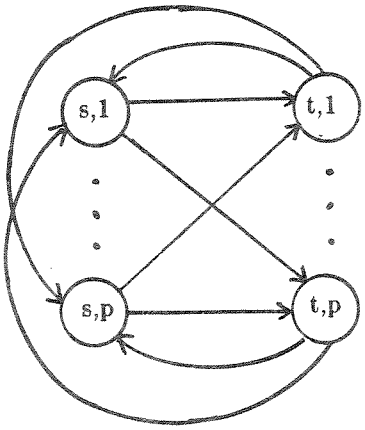


Figure 4 Structure M=(S, R, L)

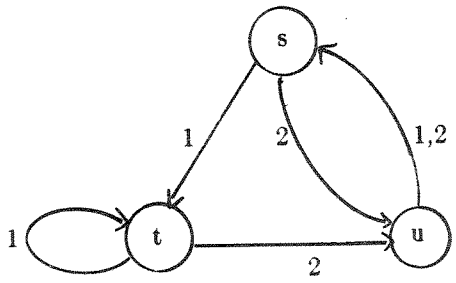


(a) A structure with arc labels

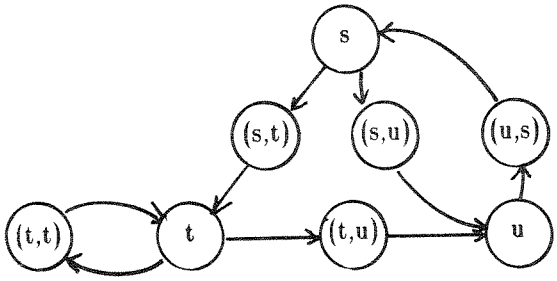


(b) A structure without arc labels

Figure 5



(a) A structure $M=(S,A_1,A_2,L)$



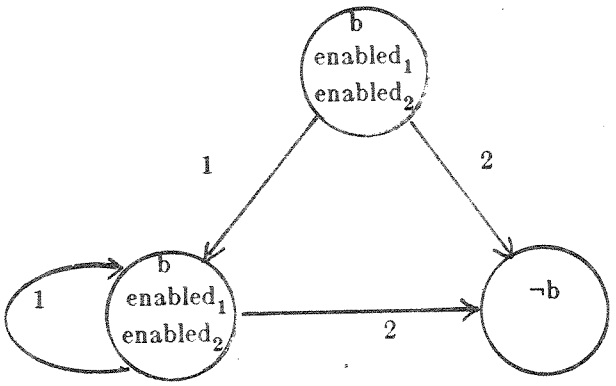
L'(s)=L(s)∪{Node}
L'(t)=L(t)∪{Node}
L'(u)=L(u)∪{Node}
L'((s,t))=L'((t,t))={Arc,1}
L'((s,u))=L'((t,u))={Arc,2}
L'((u,s))={Arc,1,2}

(b) M'=(S',A',L')

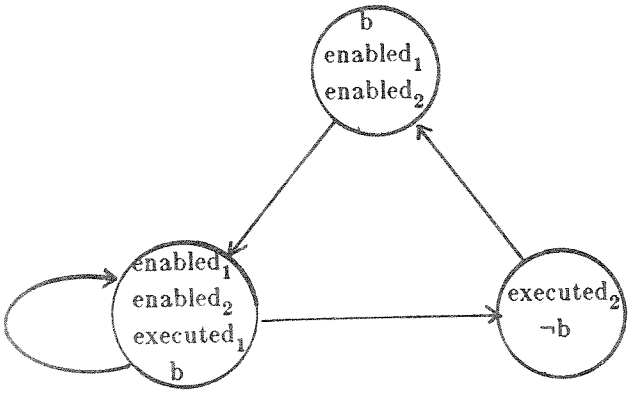
Figure 6

```
x:=0;  b:=2rue;
*[     P1:  b → x:=x+1;
  □    P2:  b → b:=false;
]
```

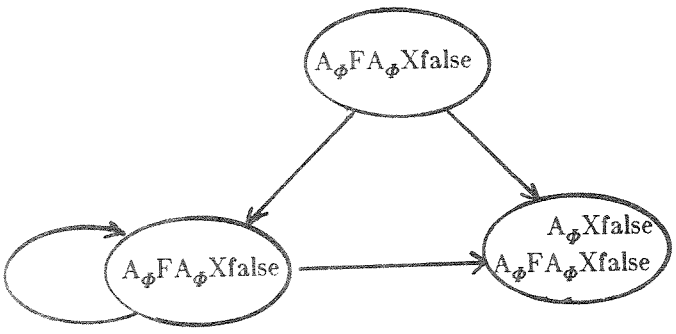
(a)



(b)

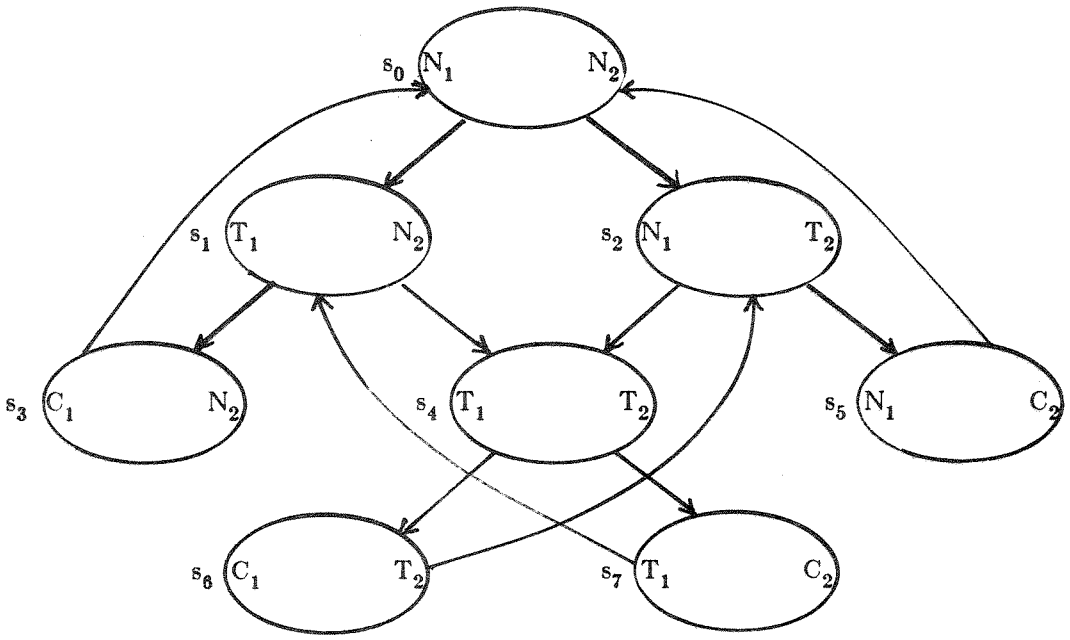


(c)

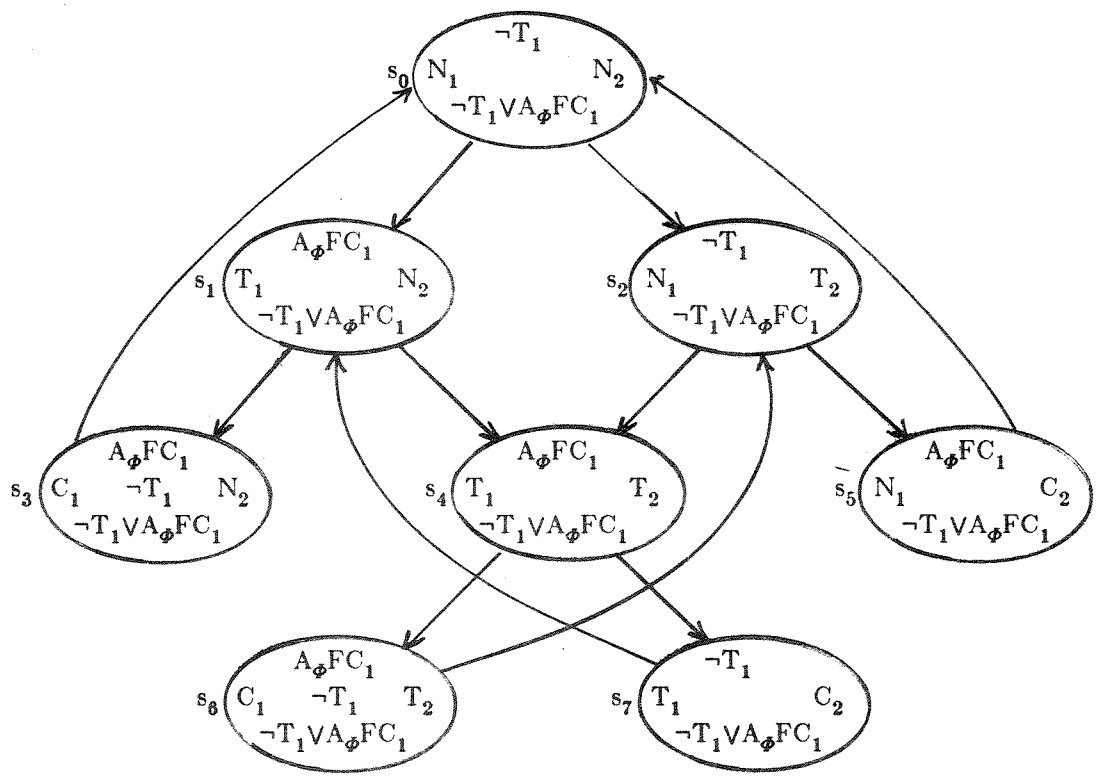


(d)

Figure 7



(a) Global state transition graph for two processes mutual exclusion problem.



(b) Global state graph after termination of model checking algorithm

Figure 8