Science of Computer Programming 8 (1987) 275-306
North-Holland

MODALITIES FOR MODEL CHECKING: BRANCHING TIME LOGIC STRIKES BACK†

E. Allen EMERSON and Chin-Laung LEI
Department of Computer Sciences, University of Texas at Austin, Austin, TX 78712, U.S.A.

Communicated by K. Apt
Received April 1985
Revised September 1985

Abstract. We consider automatic verification of finite state concurrent programs. The global state graph of such a program can be viewed as a finite (Kripke) structure, and a *model checking* algorithm can be given for determining if a given structure is a model of a specification expressed in a propositional temporal logic. In this paper, we present a unified approach for efficient model checking under a broad class of generalized fairness constraints in a branching time framework extending that of Clarke et al. (1983). Our method applies to any type of fairness expressed in a certain canonical form. Almost all 'practical' types of fairness from the literature, including the fundamental notions of impartiality, weak fairness, and strong fairness, can be succinctly written in our canonical form. Moreover, our branching time approach can easily be adapted to handle types of fairness (such as fair reachability of a predicate) which cannot even be expressed in a linear temporal logic. We go on to argue that branching time logic is always better than linear time logic for model checking. We show that given any model checking algorithm for any system of linear time logic (in particular, for the usual system of linear time logic) there is a model checking algorithm of the same order of complexity (in both the structure and formula size) for the corresponding full branching time logic which trivially subsumes the linear time logic in expressive power (in particular, for the system of full branching time logic CTL*). We also consider an application of our work to the theory of finite automata on infinite strings.

1. Introduction

It is a point of continuing controversy in the concurrency community as to whether branching time or linear time temporal logic is more appropriate for reasoning about concurrent programs (cf. [17, 10, 28]). In linear time logic, temporal operators are provided for describing events along a *single* future, although when a linear formula is used for program specification there is usually an *implicit universal quantification* over all possible futures. Commonly used linear time operators include Fp ("sometimes p"), Gp ("always p"), Xp ("nexttime p"), and [p U q] ("p until q"). In contrast, in branching time logic the operators usually reflect the branching nature

† This work was supported in part by NSF Grants MCS8302878, DCR8511354. Some of these results were presented at the 18th Annual Hawaii International Conference on Systems Sciences (in the paper 'Temporal model checking under generalized fairness constraints', which won the Best Paper Award for the Software Track) and at the 12th Annual ACM Symposium on Principles of Programming Languages (in the paper 'Modalities for model checking: Branching time strikes back').

of time by allowing explicit quantification over possible futures. The basic modalities of these logics are generally of the form: either A ("for *all* futures") or E ("for *some* future") followed by a combination of the usual linear time operators F, G, X, and U. One argument presented by the supporters of branching time logic is that it offers the ability to reason about *existential* properties of concurrent programs (e.g., potential for deadlock along *some* future) in addition to *universal* properties (e.g., inevitability of service along *all* futures).

Another advantage cited for branching time logic over linear time logic concerns the complexity of automatic verification for finite state concurrent programs. The global state graph of such a program can be viewed as a finite (Kripke) structure, and a *model checking* algorithm can be given for determining if a given structure is a model of a specification expressed in a propositional temporal logic. Provided that the algorithm is efficient, this approach is potentially of wide applicability since a large class of concurrent programming problems have finite state solutions, and the interesting properties of many such systems can be specified in a propositional temporal logic. For example, many network communication protocols (e.g., the Alternating Bit Protocol [3]) can be modeled at some level of abstraction by a finite state system.

For the branching time logic CTL (which has basic modalities of the form: A or E followed by a single occurrence of F, G, X, or U), Clarke, Emerson, and Sistla [5] give an algorithm that runs in time $O(|M||p|)$ which is *linear* in both the size of the input structure M and the length of the specification formula p; hence, this branching time approach is readily mechanizable. In contrast, the model checking problem formulated for linear time logic is known [34] to be PSPACE-complete.

On the other hand, we are frequently interested only in correctness along *fair* computation sequences. Roughly speaking, a fairness condition asserts that an event (e.g., execution of a step of a particular process) which is enabled 'sufficiently often' will eventually be performed.[1] Fairness has been widely studied in the literature (see for example [25, 17, 18, 13]) because appropriate fairness assumptions are often crucial to establishing that a program meets a certain liveness property such as absence of starvation. Unfortunately, while fairness is readily handled in linear temporal logic, it is known (cf. [17, 10]) that the branching time logic CTL used in [5] does not permit reasoning under fairness assumptions. A partial remedy to this problem is given in [5] by incorporating semantic restrictions on path quantification into the underlying structure, but it does not handle, e.g., strong fairness.

In a recent paper, Lichtenstein and Pnueli [19] suggest that efficient—*in practice*—model checking algorithms exist for linear time logic as well. By forming the cross product of the input structure M with the tableau for testing satisfiability of the linear time formula p, they develop an algorithm for model checking linear time specifications that runs in time $O(|M| \exp(|p|))$ which is *linear* in the structure size but *exponential* in the formula length. They then claim that, in practice, the

[1] Our model of concurrency is the usual one where concurrent execution of a system of processes is modelled as the nondeterministic interleaving of atomic steps of the individual processes.

specification is relatively small while the structure can be quite large. Thus, the argument goes, it is the small polynomial complexity in the size of the structure which really matters. They conclude that linear time logic is at least as good as branching time logic for model checking, and may be better because of its superior expressiveness which, in particular, allows reasoning about types of fairness not handled by [5].

In this paper, we present a model checking algorithm which permits efficient model checking in a branching time framework under any one of a broad class of generalized fairness assumptions (including, among others, strong fairness). In particular, we consider the *Model Checking Problem* (FMCP) for *Fair Computation Tree Logic* (FCTL). An 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"). All path quantifiers are thus relativized to the underlying fairness assumption Φ_0 specified by an arbitrary boolean combination of the infinitary linear time temporal operators $\overset{\infty}{F}p$ ("infinitely often p") and $\overset{\infty}{F}Gp$ ("almost everywhere p").

To develop our FMCP algorithm, we will first argue that FMCP can be reduced in time linear in the length of the functional assertion p_0 to the *Fair State Problem* (FSP): Starting from which states does there exist some path along which Φ_0 holds? Our reduction applies for any fairness specification Φ_0 involving an arbitrary boolean combination of the $\overset{\infty}{F}$, $\overset{\infty}{F}G$ operators as above. We then show that when Φ_0 is in the special *canonical form* $\bigvee_{i=1}^{n} \bigwedge_{j=1}^{n_i} (\overset{\infty}{G}p_{ij} \vee \overset{\infty}{F}Fq_{ij})$, then FSP can be solved in time $O(|M||\Phi_0|^2)$ which is linear in the size of the input structure M and quadratic in the length of the fairness specification Φ_0. While any Φ_0 can be translated into an equivalent Φ_0' in canonical form, the translation can cause an exponential increase in length (resulting in an exponential time solution to the original instance of FMCP). However, it turns out that almost all 'practical' types of fairness considered in the literature including impartiality [18], weak fairness [17], strong fairness [17], fair reachability of predicates [31], state fairness [27], as well as the technical notion of 'limited looping' fairness [1] can be directly specified using a canonical Φ_0. Hence, in practice, the fairness specification Φ_0 is in canonical form, and we can do model checking for a corresponding FCTL specification (p_0, Φ_0) on structure M efficiently in time $O(|M||p_0||\Phi_0|^2)$, which is *linear* in the size of the input structure and functional assertion and *quadratic* in the size of the fairness constraint. On the other hand, we are able to classify the complexity of FSP and FMCP for an arbitrary Φ_0: they are NP-complete.

We believe that this work offers a convincing refutation to the (apparently) popular misconception that fairness cannot be handled practically and efficiently in branching time temporal logic (cf. [17, 10]). At least for the model checking problem, all the basic types of fairness (impartiality, weak fairness, strong fairness) can be

handled in branching temporal logic as readily as in linear temporal logic. Moreover, we have presented a unified approach for handling a broad class of general fairness constraints including more than just the three basic types of fairness above. Our branching time approach can even be adapted to handle types of fairness (such as fair reachability of a predicate) which cannot be handled at all in linear temporal logic.

It is still true, however, that there are correctness properties not involving fairness which are expressible in linear temporal logic, but not expressible in the FCTL formalism, so that one might still think that linear time logic is preferable to branching time logic for some applications. Nonetheless, we can now argue that branching time logic is always better than linear time logic for model checking: We show that given a model checking algorithm for a system of linear time logic (in particular, for the usual system of linear time logic over F, G, X, and U), there is a model checking algorithm of the same order of complexity (in both the structure and formula size) for the corresponding full branching time logic which trivially subsumes the linear time logic in expressive power (in particular, for the system of *full* branching time logic CTL* in which the basic modalities are of the form: A or E followed by an *unrestricted* formula of linear time logic over F, G, X, and U). We demonstrate that handling explicit path quantifiers and even nested path quantifiers costs (essentially) nothing. Thus, there is no reason to restrict oneself to linear time logic. Use instead the corresponding full branching time logic for the same cost.

We go on to show that the formalism of FCTL can be extended to a *Generalized Fair Computation Tree Logic* (GFCTL). GFCTL is a branching time system which generalizes FCTL by allowing each path quantifier to be relativized to its own (in general, distinct) fairness constraint Φ_i. Its model checking problem is also efficiently decidable provided that each Φ_i is in the canonical form. Hence, reasoning under virtually any *combination* of different, practical fairness constraints is also feasible.

Our results strongly suggest that the real issue involved for model checking is not whether to use branching time or linear time logic, but simply: what are the basic modalities of my branching time logic? I.e., what linear time formulae can follow the path quantifiers? (Remark: In a *basic modality* of a branching time logic, the linear time formula following the path quantifier is a 'pure' linear time formula involving no nested path quantifiers.) It turns out that the relationship between the structural complexity of the basic modalities and the computational complexity of the associated model checking problem is a rather subtle one. For example, the infinitary operators $\overset{\infty}{\mathrm{F}}p$ and $\overset{\infty}{\mathrm{G}}p$ used in describing fairness properties, which are often thought of as causing all sorts of problems with discontinuities and non-definability in first order arithmetic, etc. (cf. [7, 14]), can actually simplify the problem of model checking. These matters are discussed in greater detail in the conclusion.

Finally, we consider an application of our algorithm for FSP to the theory of *finite automata on infinite strings* (ω-fa) [33] where acceptance is defined by a condition such as repeating a designated set of states infinitely often. There has

been a resurgence of interest lately in such automata because of their intimate relationship to temporal logic. For example, in testing satisfiability of a formula p_0 of linear temporal logic a directed graph labelled with appropriate subformulae, known as a *tableau*, is constructed. This tableau may be viewed as defining an ω-fa on infinite strings over (sets of) atomic propositions which accepts an input string iff it defines a model of p_0. The satisfiability problem for linear temporal logic is thus reduced to the emptiness problem for ω-fas. We will describe how the ω-fa emptiness problem can be viewed as an instance of FSP. Moreover, for all the common types of acceptance conditions, (e.g., Buchi acceptance, Rabin or Pairs acceptance, etc.) the fairness condition Φ_0 for the corresponding instance of FSP can be succinctly expressed in our canonical form, and the emptiness problem can therefore be solved in (small) polynomial time.

The remainder of the paper is organized as follows: The utility of the model checking approach to verification is discussed in Section 2. Section 3 describes the syntax and semantics of our temporal languages. Section 4 describes how to do efficient model checking in the branching time FCTL system whenever the fairness constraint Φ_0 is in canonical form, and analyzes the complexity of the general case where Φ_0 is arbitrary. In Section 5 a variety of types of practical fairness are defined and canonically specified, and an example application of FMCP to a concurrent programming problem is given. Section 6 gives the reduction of the model checking problem for full branching time logic to that for the corresponding linear time logic, while in Section 7 we show how this reduction can be applied to extend the model checking algorithm for FCTL to GFCTL. Section 8 describes how one may apply the algorithm for FSP to testing nonemptiness of finite automata on infinite strings. Finally, some concluding remarks are made in Section 9.

2. Advantages of the model checking approach to verification

Numerous approaches to reasoning about correctness of concurrent programs have been proposed in the literature. Most of these approaches can be partitioned into one of two categories:

(1) Formal systems designed with mathematical elegance as the primary motivation. Unfortunately, the designers of such systems usually pay little attention to pragmatic issues and the resulting systems are often of little practical use in proving actual (or even toy) programs correct.

(2) Systems (or methodologies) designed with practical utility as the primary motivation. Papers in this category generally illustrate the proposed method by applying it to establish correctness for a number of example programs in an effort to convince the reader of the usefulness of the approach. Unfortunately, such systems often lack the underlying mathematical framework necessary to provide a clear-cut characterization of their range of applicability (i.e., to what class of concurrent programs does the method apply). Moreover, in some of these systems even the

underlying specification language (or formalism) lacks a syntax and semantics that is mathematically well-defined. In such cases it is out of the question to consider formal justifications of the methods' adequacy and utility (e.g., soundness, deductive completeness, expressive completeness, etc.).

We would argue that our model checking approach transcends this dichotomy, and enjoys many of the best features of both categories. From a formal standpoint, the model checking approach exhibits a certain elegance: the method is applicable to a well-defined class of concurrent programs, the finite state programs. The specification language, (an appropriately chosen, particular system of) propositional temporal logic, has a precise syntax and rigorously well-defined semantics. Over finite state concurrent programs, our model checking algorithm trivially ensures that the proof method is sound and complete.

Empirical evidence demonstrates that model checking also has considerable potential as a practical verification tool. In particular, the model checking method as described in [5] has actually been implemented. The implemented EMC (Extended Model Checker) system described there has been used to mechanically verify the correctness of, e.g., the mutual exclusion example program previously proved correct by hand in [24]. It has also been successfully applied to the verification of VLSI circuits. In [6] it is described how the EMC system was used to detect an error in a circuit from Mead and Conway's VLSI text [21] and also to verify that an amended circuit was correct. Finally, we point out that the large size of the state graph encountered in certain applications need not present an insurmountable obstacle. For example, methods based on graph reachability analysis similar to our model checking algorithm have been successfully used to mechanically verify network protocols with large state spaces for European telecommunications companies ([23]; cf. [2]). We believe that our model checking algorithm, because of its low complexity, may also be suitable for similar applications.

3. Syntax and semantics of temporal logics

We inductively define a class of state formulae, which are true or false of states (intuitively corresponding to branching time logic) and a class of path formulae which are true or false of paths (intuitively corresponding to linear time logic):

S1. Any atomic proposition P is a state formula.
S2. If p, q are state formulae, then so are $p \wedge q, \neg p$.
S3. If p is a path formula, then Ep is a state formula.
P1. Any state formula p is a path formula.
P2. If p, q are path formulae, then so are $p \wedge q, \neg p$.
P3 If p, q are path formulae, then so are Xp, $(p \, U \, q)$.

Other connectives can be introduced as abbreviations in the usual way: $p \vee q$ for $\neg(\neg p \wedge \neg q)$, $p \Rightarrow q$ for $\neg p \vee q$, $p \equiv q$ for $(p \Rightarrow q) \wedge (q \Rightarrow p)$, Ap for $\neg E \neg p$, Fp for *true* U p, Gp for $\neg F \neg p$, $\overset{\infty}{F}p$ for GFp, and $\overset{\infty}{G}p$ for $\neg \overset{\infty}{F} \neg p$.

The *length* of (either a state or path) formula p, denoted $|p|$, is defined inductively as follows:

- $|P| = 0$ for atomic proposition P,
- $|p \wedge q| = 1 + |p| + |q|$ for state formulae (or path formulae) p, q,
- $|\neg p| = 1 + |p|$ for state formula (or path formula) p,
- $|\mathrm{E}p| = 1 + |p|$ for path formula p,
- $|\mathrm{X}p| = 1 + |p|$ for path formula p,
- $|p \mathrm{U} q| = 1 + |p| + |q|$ for path formulae p, q.

Thus, $|p|$ corresponds to the number of internal nodes in the 'syntax tree' for p. Note that, if $\|p\|$ denotes the number of symbols in p considered as a string in the obvious way, we have that $|p| = \theta(\|p\|)$.

The intuitive meanings of the formulae are as follows: $p \wedge q$ means the conjunction of p and q, $p \vee q$ means the disjunction of p with q, $\neg p$ means the negation of p, $p \Rightarrow q$ means p implies q, $p \equiv q$ means p is equivalent to q, $\mathrm{E}p$ means along some path p holds, $\mathrm{A}p$ means along all paths p holds, $\mathrm{X}p$ means next time p, $p \mathrm{U} q$ means q eventually holds and p holds continuously until then, $\mathrm{F}p$ means p holds at some future time, $\mathrm{G}p$ means that p always holds, $\overset{\infty}{\mathrm{F}}p$ means that p is true infinitely often, and $\overset{\infty}{\mathrm{G}}p$ means that p is true almost everywhere, i.e., at all but a finite number of times.

We now formally define the semantics of temporal logic formulae. A *prestructure* M is a triple (S, R, L) where

- S is a nonempty *set of states*,
- R is a nonempty *binary relation* on S, and
- L is a *labelling* which assigns to each state a set of atomic propositions true in the state.

We say that the binary relation R is *total* iff for each $s \in S$, there exists $t \in S$ such that $(s, t) \in R$, and that a prestructure $M = (S, R, L)$ is a *structure* provided that R is total. The semantics of a temporal logic formula is then defined with respect to a structure M. The *size* of a (pre)structure $M = (S, R, L)$, written $|M|$, is defined to be $|S| + |R|$, i.e., the sum of the number of states in S and the number of transitions in R. A *fullpath* $(s_0, s_1, s_2, \ldots)$ is an infinite sequence of states such that $(s_i, s_{i+1}) \in R$ for all i. We write $M, s \models p$ ($M, x \models p$) to mean that state formula p (path formula p) is true in structure M at state s (of path x, respectively). When M is understood, we write simply $s \models p$ ($x \models p$). We define $\models$ inductively using the convention that $x = (s_0, s_1, s_2, \ldots)$ denotes a fullpath and x^i denotes the suffix fullpath $(s_i, s_{i+1}, s_{i+2}, \ldots)$:

S1. $s \models P$ iff $P \in L(s)$, for any atomic proposition P.
S2. $s \models p \wedge q$ iff $s \models p$ and $s \models q$
$s \models \neg p$ iff not $(s \models p)$.
S3. $s \models \mathrm{E}p$ iff for some fullpath x starting at s, $x \models p$.
P1. $x \models p$ iff $s_0 \models p$, for any state formula p.
P2. $x \models p \wedge q$ iff $x \models p$ and $x \models q$
$x \models \neg p$ iff not $(x \models p)$.

P3. $x \models \mathrm{X}p$ iff $x^1 \models p$

$x \models (p \mathrm{U} q)$ iff for some $i \geq 0$, $x^i \models q$ and for all $j \geq 0$ [$j < i$ implies $x^j \models p$].

We say that state formula p is *valid*, and write $\models p$, if for every structure M and every state s in M we have $M, s \models p$. We say that state formula p is *satisfiable* if for some structure M and some state s in M we have $M, s \models p$. In this case we also say that M defines a *model* of p. We define validity and satisfiability similarly for path (i.e., linear time) formulae.

The set of path formulae generated by rules S1, P1, P2, and P3 (the set of 'pure' path formulae which contain no path quantifiers A or E) forms the usual language of linear time logic. The set of state formulae generated by all the above rules forms the language CTL*. The language CTL is the subset of CTL* where only a single linear time operator (F, G, X, or U) can follow a path quantifier (A or E) (cf. [9, 10]).

We next define FCTL (Fair CTL). An FCTL specification (p_0, Φ_0) consists of a functional assertion p_0, which is a state formula, and an underlying fairness assumptions Φ_0, which is a pure path formula. 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 F, G, X, or U. 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 a boolean combination of the infinitary linear time operators $\overset{\infty}{\mathrm{F}}$ ("infinitely often") and $\overset{\infty}{\mathrm{G}}$ ("almost always"), applied to propositional arguments. We can then view a subformula such as $\mathrm{A}_\Phi \mathrm{F}P$ of functional assertion p_0 as an abbreviation for the CTL* formula $\mathrm{A}[\Phi_0 \Rightarrow \mathrm{F}P]$. Similarly, $\mathrm{E}_\Phi \mathrm{G}P$ abbreviates $\mathrm{E}[\Phi_0 \wedge \mathrm{G}P]$. Note that *all path quantifiers in the functional assertion are relativized to the same (single) underlying fairness constraint* Φ_0. If we were to expand the abbreviations for E_Φ and A_Φ in a functional assertion, the resulting CTL* formula might be rather unwieldy due to the need to repeatedly write down multiple copies of the actual fairness formula Φ_0. Thus, when we mention the *length* of p_0, we refer to the unexpanded formula.

Formally, we define the class of FCTL *functional assertions* as follows:

FA1. Any atomic proposition P is a functional assertion.

FA2. If p, q are functional assertions, then so are $\neg p$ and $(p \wedge q)$.

FA3. If p, q are functional assertions, then so are $\mathrm{E}_\Phi \mathrm{X}p$, $\mathrm{E}_\Phi[p \mathrm{U} q]$, and $\mathrm{E}_\Phi[\neg(p \mathrm{U} q)]$.

A *propositional formula* is one formed by rules FA1, FA2 above. A *fairness constraint* is then formed by the following rules:

FC1. If p, q are propositional formulae, then $\overset{\infty}{\mathrm{F}}p$ is a fairness constraint.

FC2. If p, q are fairness constraints, then so are $\neg p$, and $(p \wedge q)$.

We can then write $\mathrm{A}_\Phi \mathrm{X}p$ for $\neg \mathrm{E}_\Phi \mathrm{X} \neg p$, $\mathrm{E}_\Phi \mathrm{F}p$ for $\mathrm{E}_\Phi[\mathit{true}\ \mathrm{U}\ p]$, $\mathrm{A}_\Phi \mathrm{G}p$ for $\neg \mathrm{E}_\Phi \mathrm{F} \neg p$, $\mathrm{A}_\Phi[p \mathrm{U} q]$ for $\neg \mathrm{E}_\Phi[\neg(p \mathrm{U} q)]$, $\mathrm{A}_\Phi \mathrm{F}p$ for $\mathrm{A}_\Phi[\mathit{true}\ \mathrm{U}\ p]$, and $\mathrm{E}_\Phi \mathrm{G}p$ for $\neg \mathrm{A}_\Phi \mathrm{F} \neg p$.

We now define the semantics of an FCTL specification (p_0, Φ_0). The fairness constraint Φ_0 is a CTL* path formula, in a restricted syntax specialized to describing fairness properties, so that $M, x \models \Phi_0$ is defined by the rules S1, P1, P2, P3. The functional assertion p_0 is an abbreviation for a CTL* state formula p_0^t obtained by expanding the $E_\Phi \ldots$ abbreviations as $E[\Phi_0 \wedge \cdots]$. Technically, the translation (i.e., expansion) t is defined as follows:

- $P^t = P$ for any atomic proposition P,
- $(p \wedge q)^t = p^t \wedge q^t$ for functional assertions p, q,
- $(\neg p)^t = \neg p^t$ for functional assertion p,
- $(E_\Phi(\Psi(p, q)))^t = E[\Phi_0 \wedge \Psi(p^t, q^t)]$ where p, q are sub-functional assertions and $\Psi(p, q)$ denotes one of Xp, $(p \mathrm{U} q)$, or $\neg(p \mathrm{U} q)$.

We write $M, s \models_{\Phi_0} p_0$ for $M, s \models p_0^t$ which means that functional assertion p_0 is true at state s of structure M under fairness assumption Φ_0. We say that fullpath x is a *fair path* in structure M under fairness assumption Φ_0 if $M, x \models \Phi_0$ holds. A state s_0 is a *fair state* iff starting at s_0 there is some fair path. A directed cycle $(s_0, s_1, \ldots, s_k, s_0)$ in structure M is a *fair cycle* if the fullpath $(s_0, s_1, \ldots, s_k, s_0, s_1, \ldots, s_k, s_0, s_1, \ldots, s_k, \ldots)$ obtained by unwinding the cycle is a fair path. A substructure C of M is called a *fair component* if C is a total, strongly component of M which contains some fair path.

We can also define a *Generalized Fair Computation Tree Logic* (GFCTL) where each path quantifier A or E is associated with a (possibly) different fairness specification Φ_i. Moreover, the arguments to the $\overset{\infty}{\mathrm{F}}$ and $\overset{\infty}{\mathrm{G}}$ operators can be generalized to be arbitrary GFCTL subformulae.

Formally, we define GFCTL as the set of state formulae generated by rules S1-S3 above together with the set of path formulae generated by rules GF1-2, GP1 below:

GF1. If p is a state formula, then $\overset{\infty}{\mathrm{F}} p$ is a fairness formula.
GF2. If Φ_1, Φ_2 are fairness formulae, then so are $\Phi_1 \wedge \Phi_2$ and $\neg\Phi_1$.

GP1. If Φ_1 is a fairness formula and p, q are state formula, then each of $[\Phi_1 \wedge Xp]$, $[\Phi_1 \wedge (p \mathrm{U} q)]$, $[\Phi_1 \wedge \neg(p \mathrm{U} q)]$ is a path formula.

We can then write $A[\Phi_1 \Rightarrow Xp]$ for $\neg E[\Phi_1 \wedge X\neg p]$, $A[\Phi_1 \Rightarrow (p \mathrm{U} q)]$ for $\neg E[\Phi_1 \wedge \neg(p \mathrm{U} q)]$, etc.

Since each GFCTL formula is also a CTL* formula, GFCTL inherits its semantics directly from the rules for CTL*. As we shall see in Section 7, the model checking algorithm for FCTL can be extended to GFCTL.

4. Model checking for fair computation tree logic

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 $s \in S$ whether $M, s \models_{\Phi_0} p_0$. The *Fair State Problem* (FSP) is: Given a structure $M = (S, R, L)$, and

a fairness constraint Φ_0, determine for each state $s \in S$ whether there is a fullpath x in M starting at s such that $M, x \models \Phi_0$.

4.1. Reduction of FMCP to FSP

Since the FSP condition is equivalent to $M, s \models_{\Phi_0} \mathrm{E}_\Phi \mathrm{X} \mathit{true}$, FSP may be viewed as a special case of FMCP. However, we can generalize a method in [5] to reduce FMCP to FSP. The reduction yields an algorithm for FMCP that runs in time linear in the size of the input functional assertion and the time to solve FSP. The reduction exploits

Observation 4.1. Any fairness constraint Φ_0 built up from $\overset{\infty}{\mathrm{F}}$ or $\overset{\infty}{\mathrm{G}}$ is 'oblivious' to the addition or deletion of finite prefixes, i.e. if x is a fullpath and y is a fullpath obtained by appending a finite prefix to x or by deleting a finite prefix of x, then $M, x \models \Phi_0$ iff $M, y \models \Phi_0$.

We thus get the following:

Proposition 4.2. *Let M be a structure, Φ_0 a fairness constraint, and p' denote the expansion of functional assertion p by substituting $\mathrm{E}[\Phi_0 \wedge \cdots]$ for $\mathrm{E}_\Phi \ldots$ as in the definition of* FCTL. *Then we have the following equivalences*:

(1) $M, s_0 \models_{\Phi_0} \mathrm{E}_\Phi \mathrm{X} p$ *iff* $M, s_0 \models \mathrm{EX}(\mathrm{E}\Phi_0 \wedge p')$ *iff* $\exists s_1 \in S\ [(s_0, s_1) \in R$ *and* $M, s_1 \models_{\Phi_0} (p \wedge \mathrm{E}_\Phi \mathrm{X} \mathit{true})]$;

(2) $M, s_0 \models_{\Phi_0} \mathrm{E}_\Phi [p\ \mathrm{U}\ q]$ *iff* $M, s \models \mathrm{E}[p'\ \mathrm{U}\ (q' \wedge \mathrm{E}\Phi_0)]$
iff $\exists k \geq 0\ \exists$ *a finite path* $(s_0, \ldots, s_k)$ *in* M *such that* $M, s_k \models_{\Phi_0} (q \wedge \mathrm{E}_\Phi \mathrm{X} \mathit{true})$ *and* $\forall i$, *if* $0 \leq i < k$, *then* $M, s_i \models_{\Phi_0} p$;

(3) $M, s \models_{\Phi_0} \mathrm{E}_\Phi [\neg(p\ \mathrm{U}\ q)]$ *iff* $M, s \models_{\Phi_0} (\mathrm{E}_\Phi(\neg q\ \mathrm{U}\ (\neg p \wedge \neg q)) \vee \mathrm{E}_\Phi \mathrm{G}(\neg q))$.

Proof. See Appendix. □

The reduction algorithm, AFMCP, is shown in Fig. 1. The algorithm operates in stages, doing stage 1, stage 2, ... etc. In stage i, it computes the truth value at all states in M for subformulae of length i using the truth values of shorter subformulae which were computed in previous stages. We assume that AFMCP calls AFSP which is an algorithm for FSP that runs in time $T_A(M, \Phi_0)$.

Proposition 4.3. *Algorithm* AFMCP *correctly solves* FMCP *by correctly labeling each state s of the input structure M with the set of subformulae of p_0 true at s, and runs in time* $\mathrm{O}(|p_0| \cdot \max(|M|, T_A(M, \Phi_0)))$.

Proof. To establish correctness, we argue by induction on i that by the end of stage i,

$$\forall f \in SF(p_0)\ \forall s \in S \text{ if } |f| \leq i \text{ then } (f \in L(s) \text{ iff } M, s \models_{\Phi_0} f). \qquad (*)$$

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, Φ0) /* use algorithm AFSP to identify fair states in M */
2.       for each s∈S do if s∈S' then L(s) := L(s) ∪ {E_φXtrue};
3.       for each formula f∈SF(p0) do /* Inductively, taking the shortest formula first. */
            case f of the form
3.1           atomic formula: skip; /* nothing to do */
3.2           ¬p: for each s∈S do if p∉L(s) then L(s) := L(s) ∪ {¬p};
3.3           p ∧ q: for each s∈S do if p, q∈L(s) then L(s) := L(s) ∪ {p ∧ q};
3.4           E_φXp: for each s∈S do if ∃(s, t)∈R[p, E_φXtrue∈L(t)] then L(s) := L(s) ∪ {E_φXp};
3.5           E_φ(p U q):
                 EU := empty set;
                 for each s∈S do
                    if q, E_φXtrue∈L(s) then begin L(s) := L(s) ∪ {E_φ[p U q]};
                                                   EU := EU ∪ {s}
                                              end;
              while EU ≠ ∅ do
                 begin
                    remove an element t from EU
                    D := {s∈S: (s,t)∈R and p∈L(s) and q, E_φ(p U q)∉L(s)};
                    for each s∈D do L(s) := L(s) ∪ {E_φ[p U q]};
                    EU := EU ∪ D;
                 end of while;
3.6           E_φ[¬(p U q)]:
                 Label the states of M with ¬p, ¬q, ¬p ∧ ¬q if appropriate according to 3.2 and 3.3.
                 Label the states of M with E_φ[¬q U (¬p ∧ ¬q)] if appropriate according to 3.5.
                 S' := {s∈S: ¬q∈L(s)};
                 M' := (S', R|S'×S', L|S');
                    /* X|Y denotes the mapping X restricted to domain Y */
                 FS' := AFSP(M', Φ0);
                 for all s∈FS' do L(s) := L(s) ∪ {E_φG¬q};
                 for all s∈S do
                    if E_φG¬q∈L(s) or E_φ[(¬q) U (¬p ∧ ¬q)]∈L(s) then L(s) := L(s) ∪ {E_φ[¬(p U q)]};
            end of cases;
   end of procedure;
```

Fig. 1. Reduction algorithm.

The basis case $i=0$ holds because the formulae of length 0 are the atomic proposition which are already correctly labelled by the definition of a structure. We assume that (*) holds for all $j<i$, and argue that (*) holds for i as well. The argument proceeds in cases based on the structure of f.

For $f=\neg p$, by induction hypothesis we know that for each state s, $L(s)$ contains p iff p is true at s; hence, we add $\neg p$ iff p is absent. Similarly, for $f=p\wedge q$, we add $p\wedge q$ to the label exactly when p and q are already present.

For $f=\mathrm{E}_\Phi \mathrm{X}p$, we add $\mathrm{E}_\Phi \mathrm{X}p$ to $L(s)$ iff there is an R-successor t of s with $\mathrm{E}_\Phi \mathrm{X}\mathit{true}$, $p\in L(t)$ as required by equivalence (1) of Proposition 4.2 above.

For $f = E_\Phi(p\ U\ q)$ we use equivalence (2) of Proposition 4.2. We first compute in EU the set of all states already labelled with q and $E_\Phi X\mathit{true}$. Each of these states satisfies q, by induction hypothesis, and is the start state of a fair path. Each obviously satisfies $E_\Phi(p\ U\ q)$ which is added to the state's label. We then use the while loop to compute the states to which the $E_\Phi(p\ U\ q)$ label should be propagated. In general, EU = the set of states already labelled with $E_\Phi(p\ U\ q)$ for which we have not yet propagated $E_\Phi(p\ U\ q)$ to its predecessors. We remove a state t from EU, and for each R-predecessor s of t such that $p \in \mathit{label}(s)$, and $E_\Phi(p\ U\ q) \notin L(s)$ already, we add $E_\Phi(p\ U\ q)$ to $L(s)$ and s to EU. Plainly, each state thus labelled with $E_\Phi(p\ U\ q)$ satisfies $E_\Phi(p\ U\ q)$.

Conversely, if s_0 satisfies $E_\Phi(p\ U\ q)$, then there is a fullpath $(s_0, s_1, s_2, \ldots)$ and a least $k \geq 0$ such that for each j, $0 \leq j < k$, p holds of s_j and q, $E_\Phi X\mathit{true}$ hold at s_k. Thus s_k will be put in EU initially and labelled with $E_\Phi(p\ U\ q)$, and if $k > 0$, each of $s_{k-1}, \ldots, s_0$ will be added to EU and labelled with $E_\Phi(p\ U\ q)$ subsequently by the while loop.

For $f = E_\Phi[\neg(p\ U\ q)]$, the algorithm exploits equivalence (3) above: $E_\Phi[\neg(p\ U\ q)] \equiv E_\Phi[\neg q\ U\ (\neg p \wedge \neg q)] \vee E_\Phi G(\neg q)$. By induction hypothesis, the states are already labelled appropriately with p and q. This labelling is extended to $\neg p$, $\neg q$, and $\neg p \wedge \neg q$. Then we check for $E_\Phi[\neg q\ U\ (\neg p \wedge \neg q)]$ using statement 3.5. To check for $E_\Phi G(\neg q)$, we let M' be the substructure of M obtained by deleting all states where q holds. Then $E_\Phi G(\neg q)$ holds at a state s iff there is a finite path from s to a fair state t in M'. Detection of fair states is done by the algorithm AFSP.

We now analyze the complexity of AFMCP. Step 1 takes time $T_A(M, \Phi_0)$ while step 2 takes time $O(|M|)$. Now, step 3 is for a loop which is executed $|SF(p_0)| = O(|p_0|)$ times. Its body is a case statement. It is easy to see that cases 3.1–3.4 use time $O(|M|)$. Case 3.5 for $f = E_\Phi[p\ U\ q]$ also requires time $O(|M|)$. To see this, first observe that to initialize EU requires time $O(|S|)$. The while loop which propagates $E_\Phi[p\ U\ q]$ can be executed at most $|S|$ times since a given state t can be removed from EU at most once. The time to process t exclusive of the time to examine all of its R-predecessors s is constant. Since each arc (s, t) is examined only once, the total time spent examining predecessors s for all t is $O(|R|)$, and the total time spent in the while loop is $O(|S|) + O(|R|) = O(|M|)$. Finally, in case 3.6 for $f = E_\Phi[\neg(p\ U\ q)]$, checking for $E_\Phi[\neg q\ U\ (\neg p \wedge \neg q)]$ requires time $O(|M|)$. To check for $E_\Phi G\neg q$, the call to AFSP requires time $T_A(M', \Phi_0) \leq T_A(M, \Phi_0)$. The total time for Case 3.6 is therefore $O(|M|) + T_A(M, \Phi_0)$, and the time for the case statement is $O(\max(|M|, T_A(M, \Phi_0)))$. Thus step 3 requires time $O(|p_0| \cdot \max(|M|, T_A(M, \Phi_0)))$, as does the entire algorithm. □

4.2. *Efficient algorithm for fair state problem*

We will now develop an efficient algorithm for FSP when Φ_0 is in the (*restricted*) *canonical form* $\Phi_0 = \bigwedge_{i=1}^{n} (\overset{\infty}{F} p_i \vee \overset{\infty}{G} q_i)$. As shown in the next subsection this will actually yield an efficient algorithm for FSP (and hence FMCP) when Φ_0 is in the (*full*) *canonical form* $\bigvee_{i=1}^{n} \bigwedge_{j=1}^{n_1} (\overset{\infty}{F} p_{ij} \vee \overset{\infty}{G} q_{ij})$.

The first step is detection of fair components. Given a total, strongly connected structure $C=(S, R, L)$, where S is finite, and a fairness constraint $\Phi_0 = \bigwedge_{i=1}^{k} (\overset{\infty}{\mathrm{F}} p_i \vee \overset{\infty}{\mathrm{G}} q_i)$, we check if C is fair w.r.t. Φ_0 as follows: if there is a fullpath in C satisfying all the $\overset{\infty}{\mathrm{F}} p_i$, then C is fair; otherwise, there is some p_j which is never true at any state in C. In this case C is fair iff the substructure obtained from C by deleting all states which do not satisfy q_j contains a component that is fair w.r.t. the fairness constraint resulting from deleting the jth conjunct of Φ_0. The algorithm AFC described in Fig. 2 is a recursive implementation of this idea. (Note: The strongly connected components of a directed graph can be found in time linear in the size (number of nodes + number of arcs) of the graph. See [37].)

Proposition 4.4. *Given a strongly connected structure $C=(S, R, L)$ where S is finite, and a fairness constraint $\Phi_0 = \bigwedge_{i=1}^{k} (\overset{\infty}{\mathrm{F}} p_i \vee \overset{\infty}{\mathrm{G}} q_i)$, the algorithm* AFC *decides whether C is a fair component w.r.t. Φ_0 in time* $O(|C| \cdot |\Phi_0|^2)$.

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

Basis: $k=0$, $\Phi_0 = true$, and the program AFC returns true immediately. Hence the hypothesis holds. (Note that any total, strongly connected component is fair w.r.t. *true*.)

```
    Recursive Boolean Procedure AFC(C, Φ0)
    /* input: C=(S, R, L) is a strongly connected structure, and
                Φ0 = ⋀_{i=1}^{k} (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);
              S' := {s ∈ S: M, s ⊨ q_i};
              C' := (S', R|S'×S', L|S');
              X := {D: D is a total strongly connected component of C'};
6             for each D∈X do if AFC(D, Φ0')=true then return(true);
7             return(false)
            end
        end;
8     return(true)
    end;
```

Fig. 2. Fair component detection algorithm.

Induction step: We assume the induction hypothesis for $k < n$, and prove it for $k = n$ as follows:

[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 total, strongly connected components in C' is fair w.r.t. Φ_0', call it D. Since D is contained in C' and every state of C' satisfies q_j, every path in D satisfies $\overset{\infty}{\mathrm{G}} q_i$. Hence D is also a fair component w.r.t. to Φ_0. Hence C itself is a fair component w.r.t. to Φ_0.

Case 2: AFC returns true at statement 8. In this case, $\forall i \in [1, n]$ some state in C satisfies p_i. Hence any cycle in C which includes all states of C defines a fair path w.r.t. Φ_0 (because C is strongly connected, there exists at least one such cycle). Let x be one such cycle; it is obvious that $M, x \models \Phi_0$. Hence C is a fair component.

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

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

To analyze the complexity, let $T(m, n, k)$ denote the complexity of AFC where $m = |C|$, $n = |\Phi_0|$, and $k =$ the number of conjuncts of Φ_0. Let $X = \{D_1, \ldots, D_l\}$ be the set of total, 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, $T(m, n, 0) = \mathrm{O}(1)$ 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:

$$
\begin{aligned}
T(m, n, k) &\leq \sum_{i=1}^{k} \mathrm{O}(m \cdot |p_i|) + \sum_{j=1}^{l} T(d_j, |\Phi_0'|, k-1) \\
&\leq \mathrm{O}(m \cdot n) + T(m, n, k-1) \\
&\leq \mathrm{O}(m \cdot n) + \mathrm{O}(m \cdot n) + T(m, n, k-2) \\
&\vdots \\
&\leq \mathrm{O}(m \cdot n) + \mathrm{O}(M \cdot n) + \cdots + \mathrm{O}(m \cdot n) \ [k \text{ times}] \\
&\leq \mathrm{O}(m \cdot n \cdot k).
\end{aligned}
$$

Since k is $\mathrm{O}(n)$ we get that $T(m, n, k) = \mathrm{O}(m \cdot n^2)$. □

Proposition 4.5. *The program* AFSP(M, Φ_0) *of Fig.* 3 *is an algorithm for* FSP *of time complexity* $O(|M| \cdot |\Phi_0|^2)$.

Proof. The program initially computes the fair (w.r.t. Φ_0) components C of the input structure M. By definition of fair component, each state t in such a C satisfies $E_\Phi X\mathit{true}$ and is added to S'. The program then determines which states s can reach a state t already satisfying $E_\Phi X\mathit{true}$. Each of these states s also satisfies $E_\Phi X\mathit{true}$ and is added to S'. Thus every state placed in S' satisfies $E_\Phi X\mathit{true}$. Conversely, if state s_0 satisfies $E_\Phi X\mathit{true}$ there exists some infinite fullpath $x = (s_0, \ldots, s_k, s_{k+1}, \ldots)$ such that every state s_i, $i \geq k$, appears infinitely often along x. Each of these states s_i lie in the same total strongly connected component of M which will be identified as a fair component. Thus each s_i will be added to S'. Since, s_0 can reach s_k, the while loop will add s_0 to S' as well. Thus, the program returns exactly the set of states S' that satisfy $E_\Phi X\mathit{true}$.

The complexity bound follows from the complexity analysis of algorithm AFC. To see this, assume that $M = (S, R, L)$ contains l total, strongly connected components $C_1, C_2, \ldots, C_l$. Then each step of the for loop requires time $|\mathrm{AFC}(C_i, \Phi_0)| + O(|C_i|)$ which is equal to $O(|C_i| \cdot |\Phi_0|^2)$. Hence the for loop requires time $O(|M| \cdot |\Phi_0|^2)$. The while loop requires only $O(|M|)$ time, so the whole algorithm takes only $O(|M| \cdot |\Phi_0|^2)$ time □

4.3. The full canonical form

Using the equivalence $E(p \vee q) \equiv Ep \vee Eq$, we see that an efficient algorithm for FSP can also be given when the fairness specification is in the full *canonical form*,

```
procedure AFSP(M, Φ0);
/* input: M=(S, R, L) is a prestructure, and
        Φ0 = ⋀_{i=1}^{n} (F^∞ p_i ∨ G^∞ q_i)
  output: S' -- the set of fair states of prestructure M */
begin
     S' := ∅;
     let X={C: C is a total 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
          remove an element t from CLOSE;
          D := {s: (s, t)∈R ∧ s∉S'};
          S' := S' ∪ D;
          CLOSE := CLOSE ∪ D
     end;
     return(S');
end;
```

Fig. 3. Algorithm for calculating fair states.

$\Phi_0 = \bigvee_{i=1}^{n} \bigwedge_{j=1}^{n_i} (\overset{\infty}{\mathrm{G}} p_{ij} \vee \overset{\infty}{\mathrm{F}} q_{ij})$. Since $\mathrm{E}\Phi_0 \equiv \bigvee_{i=1}^{n} \mathrm{E} \bigwedge_{j=1}^{n_i} (\overset{\infty}{\mathrm{G}} p_{ij} \vee \overset{\infty}{\mathrm{F}} q_{ij})$, to see if a state is fair w.r.t. Φ_0, it suffices to check if it is fair w.r.t. one of the disjuncts of Φ_0. We have thus established

Theorem 4.6. FMCP *for input specification* (p_0, Φ_0) *with* $\Phi_0 = \bigvee_{i=1}^{n} \bigwedge_{j=1}^{n_i} (\overset{\infty}{\mathrm{G}} p_{ij} \vee \overset{\infty}{\mathrm{F}} q_{ij})$ *and for input structure* $M = (S, R, L)$ *can be solved in time* $O(|p_0| \cdot |M| \cdot |\Phi_0|^2)$.

Proof. By the preceding remark AFSP can be used to solve FSP for Φ_0 in full canonical form in time $T_A = O(|M| \cdot |\Phi_0|^2)$. Then AFMCP solves FMCP in time $O(|p_0| \cdot \max(|M|, T_A)) = O(|p_0| \cdot |M| \cdot |\Phi_0|^2)$. □

Note that *any* arbitrary Φ_0 can be placed in canonical form by first putting it in *Disjunctive Normal Form* (which can cause an exponential blowup) and then 'padding' with $\overset{\infty}{\mathrm{F}}$*false* or $\overset{\infty}{\mathrm{G}}$*false* as needed (which causes only a linear blowup): $\bigvee_{i=1}^{n} (\bigwedge_{j=1}^{n_i} \overset{\infty}{\mathrm{F}} p_{ij} \wedge \bigwedge_{k=1}^{m_i} \overset{\infty}{\mathrm{G}} q_{ik})$, which is in DNF, is changed, exploiting the equivalence $\overset{\infty}{\mathrm{G}} p' \wedge \overset{\infty}{\mathrm{G}} p'' \equiv \overset{\infty}{\mathrm{G}} (p' \wedge p'')$, into $\bigvee_{i=1}^{n} (\bigwedge_{j=1}^{n_i} \overset{\infty}{\mathrm{F}} p_{ij} \wedge \overset{\infty}{\mathrm{G}} q_i')$, where $q_i' = \bigwedge_{k=1}^{m_i} q_{ik}$. This is padded to get $\bigvee_{i=1}^{n} (\bigwedge_{j=1}^{n_i} (\overset{\infty}{\mathrm{F}} p_{ij} \vee \overset{\infty}{\mathrm{G}}\mathit{false}) \wedge (\overset{\infty}{\mathrm{F}}\mathit{false} \vee \overset{\infty}{\mathrm{G}} q_i'))$. However, many 'practical' fairness specifications, as in the next section, can be massaged into canonical form with only a linear blowup.

4.4. Complexity of the general case

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

Theorem 4.7. FSP *is* NP-*complete.*

Proof. [NP-hardness]: We will reduce 3-SAT to FSP, with fairness constraint of the form $\bigwedge_{i=1}^{n} (\overset{\infty}{\mathrm{G}} \neg p_i \vee \overset{\infty}{\mathrm{G}} \neg q_i)$. Given a formula g in 3-CNF with n variables and m factors, we show how to construct, in polynomial time, a structure $M = (S, R, L)$ with a designated state $s \in S$, and a fairness constraint Φ_0 such that there is a fullpath z in M starting from s and $M, z \models \Phi_0$ iff g is satisfiable.

Let $x_1, x_2, \ldots, x_n$ and $C_1, C_2, \ldots, C_m$ be the variables and factors of g (i.e. $g = \bigwedge_{i=1}^{m} C_i$), 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 $k \in [1, n]$. 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 shown in Fig. 4. Formally, we let

$$S = \{s, t\} \cup \{v_{ij} : 1 \leq i \leq m, 1 \leq j \leq 3\},$$

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

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

$$L(v_{ij}) = \begin{cases} \{p_k\} & \text{if } l_{ij} = x_k, \\ \{q_k\} & \text{if } l_{ij} = \neg x_k, \end{cases} \qquad L(s) = L(t) = \emptyset.$$

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

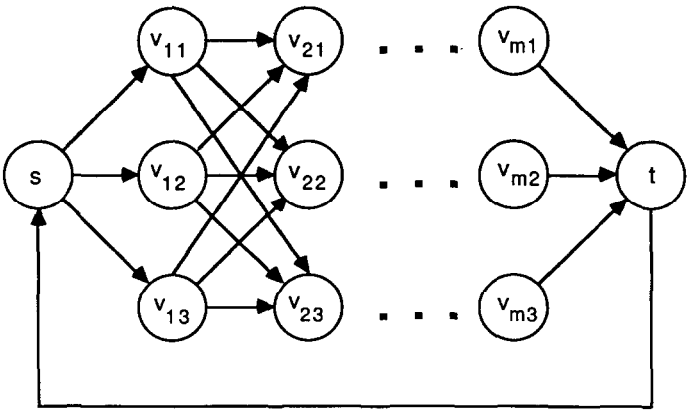


Fig. 4. Structure $M = (S, R, L)$.

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 [34] 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. □

Remark. In [34] it was shown that, in effect, FSP for Φ_0 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 $\overset{\infty}{\mathrm{F}}p \equiv \mathrm{GF}p$ and $\overset{\infty}{\mathrm{G}}p \equiv \mathrm{FG}p$. But NP-hardness for Φ_0 of our type does *not* follow from the proof in [34]. That proof involved a different reduction to a formula $\mathrm{F}p_1 \wedge \cdots \wedge \mathrm{F}p_n$. Because F$p$ is not expressible in our Φ_0 language, such an argument cannot be applied. Since our Φ_0 language has more restricted syntax, its decision problem might be easier. Our NP-hardness argument shows that this is not the case.

Corollary 4.8. FMCP *is* NP-*complete.*

Proof. Since FSP is a special case of FMCP, NP-hardness follows directly from Theorem 4.7, and NP-membership follows from Proposition 4.3 and Theorem 4.7. □

5. Applications

5.1. Fairness notions

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

(1) *Impartiality* [18]: An infinite computation sequence is *impartial* iff every process is executed infinitely often during the computation. This notion can be expressed as $\bigwedge_{i=1}^{n}(\overset{\infty}{\mathrm{F}}executed_i)$, where $executed_i$ is a proposition which asserts that process i is being executed.

(2) *Weak fairness* [17] (also known as *justice* [18]): An infinite computation sequence is *weakly fair* iff every process enabled almost everywhere is executed infinitely often. The following FCTL formulae express weak fairness:

$$\bigwedge_{i=1}^{n}(\overset{\infty}{\mathrm{G}}enabled_i \Rightarrow \overset{\infty}{\mathrm{F}}executed_i) \equiv \bigwedge_{i=1}^{n}(\overset{\infty}{\mathrm{F}}(\neg enabled_i \vee executed_i)).$$

(3) *Strong fairness* [17] (called simply *fairness* in [18]): An infinite computation sequence is *strongly fair* iff every process enabled infinitely often is executed infinitely often. This notion of fairness can be expressed using the following FCTL formulae:

$$\bigwedge_{i=1}^{n}(\overset{\infty}{\mathrm{F}}enabled_i \Rightarrow \overset{\infty}{\mathrm{F}}executed_i) \equiv \bigwedge_{i=1}^{n}(\overset{\infty}{\mathrm{G}}\neg enabled_i \vee \overset{\infty}{\mathrm{F}}executed_i).$$

(4) *Generalized fairness* [13]: Note that we can replace the propositions $executed_i$ and $enabled_i$ by an ordinary propositions so that we can reason not only about, say, strong fairness w.r.t. process enabling and execution but also strong fairness w.r.t. the occurrence of any propositional properties. This is the idea behind *generalized fairness*. Let $\mathscr{F} = ((P_1, Q_1), (P_2, Q_2), \ldots, (P_k, Q_k))$ be a finite list of pairs of propositions (where we think of each proposition as representing an arbitrary state or transition property). Then we can express that a computation is *unconditionally* $\mathscr{F}$-fair by $\bigwedge_{i=1}^{k}\overset{\infty}{\mathrm{F}}Q_i$, *weakly* $\mathscr{F}$-*fair* by $\bigwedge_{i=1}^{k}(\overset{\infty}{\mathrm{G}}P_i \Rightarrow \overset{\infty}{\mathrm{F}}Q_i)$, and *strongly* $\mathscr{F}$-*fair* by $\bigwedge_{i=1}^{k}(\overset{\infty}{\mathrm{F}}P_i \Rightarrow \overset{\infty}{\mathrm{F}}Q_i)$.

In the sequel, let $M = (S, R, L)$ be a structure, s a state in S with successcors $t_1, \ldots, t_n$, and $x = (s_0, s_1, s_2, \ldots)$ be a full computation path in M.

(5) *State fairness* [27] (also called *fair choice from states* [31]): We say that an infinite computation x is *state fair for state* s provided that if s appears infinitely often along x, then every transition (s, t) in M out of s also appears infinitely often along x. We say that x is *state fair* iff it is state fair for all s in M. Using a suggestive interpretation of atomic propositions, we can express state fairness in canonical form. Considering the structure M as finite state concurrent system, an arc (s, t) along a computation x in M may be viewed as a computation step which takes the system from state s to state t. Let $at(s)$ denote that the system is at state s, and $in(s, t)$ denote that the system is performing a transition from state s to state t. Thus we can express state fairness for s in M as $\bigwedge_{(s,t)\in R}(\overset{\infty}{\mathrm{F}}at(s) \Rightarrow \overset{\infty}{\mathrm{F}}in(s, t))$.

(6) '*Limited looping*' *fairness* [1]: We say that fullpath x is *limited looping fair for state* s provided that if s occurs infinitely often along x, then each state t accessible from s in M also occurs infinitely often along x. We say that x is *limited looping fair* iff x is limited looping fair for all states s in M. Let $r(s)$ denote the set of all states t reachable from s in M. Then a computation is limited looping fair for s iff

$\bigwedge_{t \in r(s)} (\overset{\infty}{\mathrm{F}} at(s) \Rightarrow \overset{\infty}{\mathrm{F}} at(t))t))$ holds along it. Similarly, $\bigwedge_{(s,t) \in R} (\overset{\infty}{\mathrm{F}} at(s) \Rightarrow \overset{\infty}{\mathrm{F}} at(t))$ means limited looping fair.

(7) *Fair rechability of predicate P* [31]: We say that a computation x is *fair w.r.t. reachability of predicate P* provided that if there are infinitely many states s occurring along x from which a state satisfying proposition P is reachable, then there are infinitely many states t along x which themselves satisfy P. This can be formulated as $\overset{\infty}{\mathrm{F}}\mathrm{EF}P \Rightarrow \overset{\infty}{\mathrm{F}}P$.[2]

It is worth pointing out that, despite the seeming complexity of the state fairness and limited looping fairness specifications, they exhibit several nice properties which simplify our model checking algorithm. In fact, we do not even have to explicitly express these fairness constraints, and can still do model checking correctly and efficiently.

Proposition 5.1. *For any finite structure $M = (S, R, L)$, and for all states s in S, there is a state fair (limited looping fair) path starting from s.*

Proof. From each state s there is a fullpath ending in a terminal strongly connected component with all the arcs (states, resp.) of the strongly connected component appearing infinitely often on the path. □

Due to Proposition 5.1, FSP under the above two fairness notions becomes trivial. Furthermore, the model checking procedure for formulae of the form $\mathrm{A}_\Phi \mathrm{X}p$, $\mathrm{E}_\Phi \mathrm{X}p$, or $\mathrm{E}_\Phi[p\ \mathrm{U}\ q]$ reduce to exactly the same as the corresponding CTL formulae. To see how to do model checking for $\mathrm{A}_\Phi[p\ \mathrm{U}\ q]$, recall that $\mathrm{A}_\Phi[p\ \mathrm{U}\ q] \equiv \neg \mathrm{E}_\Phi \mathrm{G}(\neg q) \vee \neg \mathrm{E}_\Phi[(\neg q)\ \mathrm{U}\ (\neg p \wedge \neg q)]$. Hence we only have to describe how to check formulae of the form $\mathrm{E}_\Phi \mathrm{G}r$. The key idea is that every state fair (limited looping fair) fullpath must end in a terminal strongly connected component (of the structure in question), and every state in the terminal component must occur infinitely often on the path. Therefore, a state s satisfies $\mathrm{E}_\Phi \mathrm{G}r$ iff there is a finite path starting from s leading to a terminal strongly connected component such that all states involved satisfy proposition r. We thus have

Theorem 5.2. FMCP *for input functional assertion p_0 with the fairness constraint Φ_0 corresponding to state fairness (or limited looping fairness) and input structure M, can be solved in time* $O(|p_0| \cdot |M|)$.

We should also point out that our method can be used to perform model checking for the probabilistic branching temporal logic PTL_f of [15] interpreted over finite

[2] Here EFP is a formula of ordinary CTL. Technically speaking, it is not a propositional formula, but it is straightforward to extend our method to allow it as a 'primitive argument' in fairness specifications; simply compute all states from which P is reachable and label them appropriately with EFP or ¬EFP, before applying the algorithm for FSP. Alternatively, this fairness specification can be directly expressed as a fairness formula of GFCTL. See Section 7.

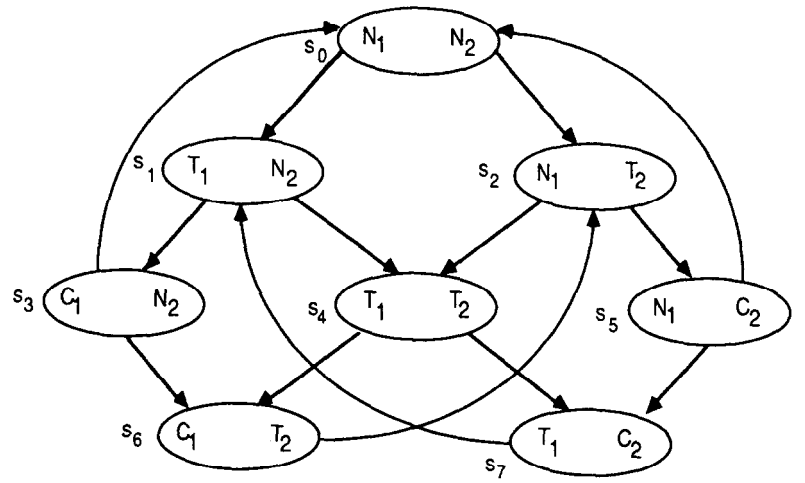
Markov chains. The syntax of PTL_f is very similar to FCTL but an assertion such as $\mathrm{A}_\Phi \mathrm{F}p$ means intuitively that p will eventually hold with probability one. We can define a simple translation from PTL_f into FCTL such that a PTL_f formula holds in a finite Markov chain iff the corresponding FCTL formula holds in the chain viewed as a structure, provided that the underlying fairness assumption is state fairness.

Remark. There was a technical fine 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 transition (i.e. traversal of an arc). To allow a precise differentiation between excecution of transition actions and enabling of state conditions, we can extend the semantics of FCTL so that a structure $M = (S, A_1, A_2, \ldots, A_p, L)$ where $A_i \subseteq S \times S$ represents (the atomic actions of) process i, and where we think of each arc $(s_1, s_2) \in A = A_1 \cup \cdots \cup A_p$ 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 arc assertions: $executed_i$ hold at (s_1, s_2) iff $(s_1, s_2) \in A_i$. The fairness specifications such as $\overset{\infty}{\mathrm{F}} enabled_i \Rightarrow \overset{\infty}{\mathrm{F}} executed_i$ can thus 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. Alternatively, we can encode the extended semantics with arc labels into the original semantic framework of only having state labels through 'duplication' of states as is done in [26].

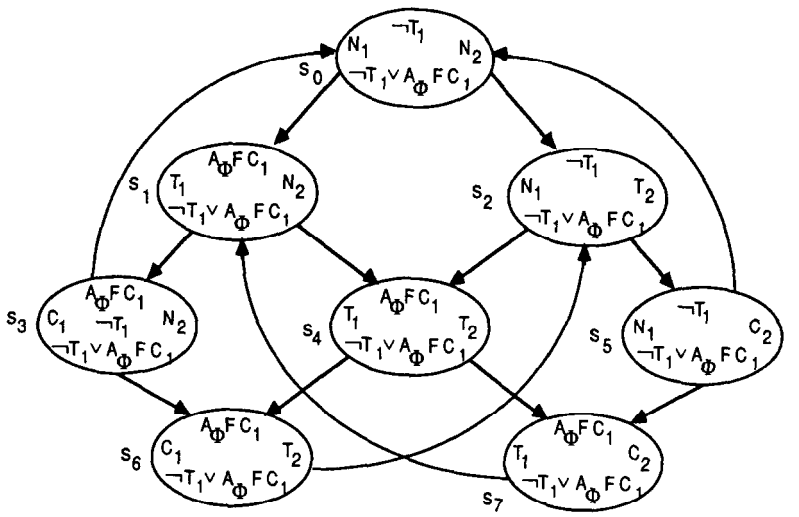
5.2. *Example*: *Mutual exclusion problem*

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, or C_i, the Critical region. A global state transition graph is given in Fig. 5(a). 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 $T_i \Rightarrow \mathrm{A}_\Phi \mathrm{F} C_i$ for each process i. Note that this solution is not starvation free under an unfair scheduler, nor is it starvation free under weak fairness. For example, the infinite execution sequence $x = (s_0, s_1, s_4, s_7, s_1, s_4, s_7, \ldots)$ is a weakly fair path but along this path process 1 never enters its critical region even though it is almost always in its trying region. However, we will show that the solution is indeed starvation free under the strong fairness assumption $\Phi_0 = (\overset{\infty}{\mathrm{F}} enabled_1 \Rightarrow \overset{\infty}{\mathrm{F}} executed_1) \wedge (\overset{\infty}{\mathrm{F}} enabled_2 \Rightarrow \overset{\infty}{\mathrm{F}} executed_2)$. Without loss of generality, we only consider the starvation free property for process 1: $p_0 = \mathrm{A}_\Phi \mathrm{G}(\neg T_1 \vee \mathrm{A}_\Phi \mathrm{F} C_1)$. The states of the global transition graph will be labeled with subformulae of p_0 during execution of model checking algorithm. On termination every state will be labeled with $\neg T_1 \vee \mathrm{A}_\Phi \mathrm{F} C_1$, as shown in Fig. 5(b). Thus we can conclude that $s_0 \models_{\Phi_0} p_0$. It follows that process 1 cannot be prevented from entering its critical region once it has entered its trying region.



(a)



(b)

Fig. 5. (a) Global state transition graph for two processes mutual exclusion problem. (b) Global state graph after termination of model checking algorithm.

6. Model checking for full branching time logic

The *Branching Time Logic Model Checking Problem* (BMCP) formulated for CTL* is: Given a finite structure $M = (S, R, L)$ and a CTL* formula p determine for each state s in S whether $M, s \models p$ and, if so, label s with p. The *Model Checking Problem for Linear Time Logic* (LMCP) can be similarly formulated as follows (cf. [34]): We are given a finite structure $M = (S, R, L)$ and a formula p of ordinary linear

temporal logic over F, G, X, and U. Formally, p is a path formula generated by rules S1, P1, P2, P3 in the previous section (so that it contains *no* path quantifiers A or E). Then determine for each state in S, whether there is a fullpath satisfying p starting at s, and, if so, label s with Ep.

Remarks. (1) Note that the [19] algorithm can be trivially modified to do this.

(2) This definition of LMCP may not, at first glance, correspond to how one thinks it should be formulated because most proponents of linear time logic observe the convention that linear time formula p is true of a structure (representing a concurrent program) iff it is true of *all paths in the structure.* Please note, however, that p is true of all paths in the structure iff Ap holds at all states of the structure. Since $Ap \equiv \neg E \neg p$, by solving our formulation LMCP and then scanning all states to check whether Ap holds, we get a solution to the 'alternative' formulation.

(3) Also, observe that FSP may be viewed as a special case of LMCP, and FMCP as a special case of BMCP.

Despite the superficially plausible intuition that handling, e.g., nested, alternating path quantifiers would make BMCP more difficult than LMCP, that is not the case.

Definition 6.1. Let $\mathcal{R}$ denote the set of nonnegative real numbers. An n-ary function $f: \mathcal{R}^n \to \mathcal{R}$ is *superadditive in its* i*th argument* provided that $f(x_1, \ldots, x_{i-1}, a+b, x_{i+1}, \ldots, x_n) \geq f(x_1, \ldots, x_{i-1}, a, x_{i+1}, \ldots, x_n) + f(x_1, \ldots, x_{i-1}, b, x_{i+1}, \ldots, x_n)$ for all $a, b \in \mathcal{R}$. f is *superadditive* iff it is superadditive in all of its arguments.

Proposition 6.2. *A differentiable function $f: \mathcal{R} \to \mathcal{R}$ is superadditive if its derivative f' is nondecreasing and $f(0) = 0$.*

Proof. If f' is nondecreasing, then for all a, b in $\mathcal{R}$ we have: $f(a+b) - [f(a)+f(b)] = \int_0^{a+b} f'(x)\,dx - \int_0^a f'(x)\,dx - \int_0^b f'(x)\,dx = \int_a^{a+b} f'(x)\,dx - \int_0^b f'(x)\,dx = \int_0^b [f'(x+a) - f'(x)]\,dx \geq 0$. Hence, f is superadditive. □

Corollary 6.3. *Let $f(x_1, \ldots, x_n)$ be differentiable with respect to x_i. Then f is superadditive in x_i if its partial derivative with respect to x_i is nondecreasing and $f(x_1, \ldots, x_{i-1}, 0, x_{i+1}, \ldots, x_n) = 0$.*

Remark. Intuitively, superadditivity of a complexity function requires that it takes at least as long as to solve a large problem as it takes to solve both of two subproblems obtained by decomposing the original problem. Observe that any sum of positive coefficient polynomials or exponential functions is superadditive.

Theorem 6.4. *Suppose we are given a model checking algorithm* LMCA *of superadditive complexity $f(|M|, |p_0|)$ for the usual system of linear time logic. Then there is a model checking algorithm* BMCA *of complexity* $O(f(|M|, |p_0|))$ *for the corresponding full branching time logic* CTL* *which trivially subsumes the linear time logic in expressive power.*

Proof. The key point is that we can actually use LMCA to evaluate Ep for an *arbitrary* path formula p, in particular for one which contains nested path quantifiers. To model check an arbitrary CTL* state formula p_0, we simply model check on each subformula by *recursive descent* based on the inductive definition of CTL* state formulae using LMCA as a subroutine to evaluate Ep formulae:

(1) If p_0 is an atomic proposition P, then there is nothing to do since the structure is already labelled correctly with the propositions true in each state.

(2) If p_0 is a conjunction $p \wedge q$ of two state formulae p, q, then recursively model check for each of p and q; then add p_0 to the label of each state whose label contains both p and q.

(3) If p_0 is a negation $\neg p$ of a state formula p, then recursively model check for p. Add $\neg p$ to the label of each state not containing p.

(4) If p_0 is of the form Ep where p is a path formula, then let E$q_1, \ldots,$ Eq_k be the list of all 'top level' proper E-subformulae of p (i.e., each Eq_i is a subformula of Ep, but is not a subformula of any subformula Er of Ep where Er is different from Ep and from Eq_i) If this list is empty, then p is a 'pure' linear time formula with no nested path quantifiers so call the linear time model checker LMCA for p. Otherwise, for each Eq_i recursively call this state model checker. When all recursive calls have returned, each state s will be labelled with Eq_i as appropriate. Introduce a list of new, 'fresh' atomic propositions $Q_1, \ldots, Q_k$. Augment the labelling of each state s in the structure for each i, with Q_i if Eq_i holds at s. Let p' be the path formula resulting from substituting each Q_i for its corresponding Eq_i in p. Call the linear model checker LMCA for p'. When it returns exactly those states at which Ep' holds will be labelled with Ep'. Augment the labels of those states with Ep.

We claim that if LMCA is of time complexity bounded by superadditive functions $f(|M|, |p_0|)$, then the recursive descent algorithm BMCA is of complexity $O(f(|M|, |p_0|))$. (In particular, the BMCP algorithm for CTL* resulting from the [19] algorithm for LMCP for ordinary linear temporal logic is of the same order of complexity.)

To establish this claim, first note that since f is superadditive in both arguments, we have that $c_1 \cdot m \cdot n \leq f(m, n)$ for some $c_1 > 0$. In particular, $c_1 \cdot m \leq f(m, 1)$. Also note that (with appropriate data structures) we can install a formula in the label of a state in constant time c_2, for some $c_2 > 0$. Hence, e.g., installing the $\neg p$ formula in the label of each state already determined not to satisfy p can be done in time $\leq c_2 \cdot |M|$. Take $c = (c_1 + c_2)/c_1 + 1$. Observe that $c_2 \cdot |M| \leq ((c_1 + c_2)/c_1) \cdot c_1 \cdot |M| \leq c \cdot f(|M|, 1)$.

We will analyze the complexity of BMCA by charging the costs to two disjoint accounts. Let $f''(|M|, |p_0|)$ be the cost of manipulating and installing the labels of the auxiliary propositions, and let $f'(|M|, |p_0|)$ be the cost exclusive of manipulating these propositions. Since there are only $O(|p_0|)$ auxiliary propositions, $f''(|M|, |p_0|) = O(|M| \cdot |p_0|) \leq c_3 \cdot f(|M|, |p_0|)$ for some constant $c_3 > 0$.

We will show, by induction on the structure of formula p_0, $f'(|M|, |p_0|) \leq c \cdot f(|M|, |p_0|)$:

If p_0 is an atomic proposition, then clearly the hypothesis holds.
If p_0 is $\neg p$, then

$$\begin{aligned} f'(|M|, |\neg p|) &= \text{the cost of a recursive call for } p \\ &\quad + \text{the cost of labelling with } \neg p \text{ appropriately} \\ &= f'(|M|, |p|) + c_2 \cdot |M| \\ &\leq c \cdot f(|M|, |p|) + c_2 \cdot |M| \quad \text{(by induction hypothesis)} \\ &\leq c \cdot f(|M|, |p|) + c \cdot f(|M|, 1) \\ &\leq c \cdot f(|M|, |p|+1)) \quad \text{(by superadditivity)} \\ &= c \cdot f(|M|, |\neg p|). \end{aligned}$$

If p_0 is $p \wedge q$, then

$$\begin{aligned} f'(|M|, |p \wedge q|) &= \text{the cost of recursive calls for } p, q \\ &\quad + \text{the cost of labelling with } p \wedge q \text{ appropriately} \\ &= f'(|M|, |p|) + f'(|M|, |q|) + c_2 \cdot |M| \\ &\leq c \cdot f(|M|, |p|) + c \cdot f(|M|, |q|) \\ &\quad + c \cdot |M| \quad \text{(using induction hypothesis and definition of } c\text{)} \\ &\leq c \cdot f(|M|, |p|) + c \cdot f(|M|, |q|) + c \cdot f(|M|, 1) \\ &\leq c \cdot f(|M|, |p|+|q|+1) \quad \text{(by superadditivity)} \\ &\leq c \cdot f(|M|, |p \wedge q|). \end{aligned}$$

If p_0 is $\mathrm{E}p$, then, if p is a pure linear time formula,

$$\begin{aligned} f'(|M|, |\mathrm{E}p|) &= \text{the cost of calling LMCA for } p \\ &= f(|M|, |p|) \\ &\leq f(|M|, |\mathrm{E}p|) \leq c \cdot f(|M|, |\mathrm{E}p|); \end{aligned}$$

Otherwise, $\mathrm{E}p$ is of the form $\mathrm{E}p'(\mathrm{E}q_1, \ldots, \mathrm{E}q_k)$ as above. Then

$$\begin{aligned} f'(|M|, |\mathrm{E}p|) &= \text{the cost of recursive calls for the } \mathrm{E}q_i \\ &\quad + \text{the cost of LMCA for } p' \\ &\quad + \text{the cost of labelling each state already labelled by } \mathrm{E}p' \text{ with } \mathrm{E}p \end{aligned}$$

$$= f'(|M|, |\mathrm{E}q_1|) + \cdots + f'(|M|, |\mathrm{E}q_k|) + f(|M|, |p'|) + c_2 \cdot |M|$$

$$\leq c \cdot f(|M|, |\mathrm{E}q_1|) + \cdots + c \cdot f(|M|, |\mathrm{E}q_k|) + f(|M|, |p'|)$$

$$+ c \cdot f(|M|, 1) \quad \text{(using induction hypothesis)}$$

$$\leq c \cdot f(|M|, |\mathrm{E}q_1| + \cdots + |\mathrm{E}q_k| + |\mathrm{E}p'|) \quad \text{(by superadditivity)}$$

$$\leq c \cdot f(|M|, |\mathrm{E}p|).$$

Hence, we conclude that the total complexity of BMCA is $f'(|M|, |p_0|) + f''(|M|, |p_0|) \leq c \cdot f(|M|, |p_0|) + c_3 \cdot f(|M|, |p_0|) = \mathrm{O}(f(|M|, |p_0|))$. □

Remark. The technique of introducing auxiliary propositions as above in order to reduce the depth of nesting of modal operators appears to go back at least to [12]. It was previously used to reduce the depth of nesting of path quantifiers for CTL* in [5] and [11]. We use it here so that we can reduce BMCP to our exact original formulation of LMCP. Note that the key idea of the algorithm, however, is really not the introduction of auxiliary propositions, but simply evaluation by recursive descent. In an actual implementation, for example, there is no need to use the auxiliary propositions, since the Ep subformulae could themselves be viewed as 'atomic'. I.e., the labels of a state could be implemented using bit vectors or pointers for linked lists. These could refer to an Ep formula, indeed any state formula, as easily as to an atomic proposition.

It is also easy to see that this reduction will work for any linear temporal formalism (with any linear time operators, e.g., until operator, interval operators, precedence operators etc.) and its corresponding full branching temporal logic. Formally, let L be a linear time logic generated by a set of rules, PRULES. Then the corresponding *full branching time logic for L* is defined to be the set of state formulae generated by S1, S2, S3, P1, and PRULES. We thus have the following general theorem:

Theorem 6.5. *Given any model checking algorithm* LMCA *for any system of linear time logic there is a model checking algorithm* BMCA *of the same order of complexity (in both the structure and formula size) for the corresponding full branching time logic which trivially subsumes the linear time logic in expressive power.*

7. Model checking for generalized fair computation tree logic

In most cases, the formalism of FCTL should provide ample generality because we typically reason about behaviors of concurrent systems under a single fairness assumption over the entire system. In some applications, however, we might wish

to reason about one portion of a system under one type of fairness and about another portion under a different type of fairness. For example, we might wish to reason about a CSP program where we assume that one type fairness is imposed on the scheduling of enabled guarded commands while another type of fairness is assumed for the scheduling of *pairs* of processes mutually able to rendezvous. This type of reasoning under a combination of different fairness constraints is not accommodated by the FCTL formalism. However, multiple fairness constraints are permitted in the *Generalized Fair Computation Tree Logic* (GFCTL) where each path quantifier is associated with a (possibly) different fairness specification Φ_i, and, the arguments to the $\overset{\infty}{\mathrm{F}}$ and $\overset{\infty}{\mathrm{G}}$ operators are arbitrary GFCTL subformulae.

The results of Section 4 show that we can do model checking on structure M for FCTL specification (p_0, Φ_0) in time $O(|M| \cdot |p_0| \cdot |\Phi_0|^2)$, provided Φ_0 is in canonical form. In particular, this bound holds for each of the FCTL formulae $E_\Phi X p$, $E_\Phi[p \mathrm{U} q]$, and $E_\Phi[\neg(p \mathrm{U} q)]$, where p, q are propositional formulae. If we were to measure the complexity in terms of the expanded formulae p' ($E[\Phi_0 \wedge Xp]$, $E[\Phi_0 \wedge (p \mathrm{U} q)]$, or $E[\Phi_0 \wedge \neg(p \mathrm{U} q)]$, respectively), it would be $O(|M| \cdot |p'|^2)$. Thus, we have an algorithm for LMCP over the language of path formulae $L = \{(\Phi_0 \wedge Xp), (\Phi_0 \wedge (p \mathrm{U} q)), (\Phi_0 \wedge \neg(p \mathrm{U} q)$: Φ_0 is any FCTL fairness constraint in canonical form and p, q are propositional formulae$\}$ of time complexity $O(|M| \cdot |p''|^2)$, where M is the input structure, and $p'' \in L$ is the input linear time formula. Since L is precisely the set of linear time formulae corresponding to the basic modalities of GFCTL, applying Theorem 6.5 we have established

Theorem 7.1. BMCP *for* GFCTL *with input structure M and input formula p' can be solved in time* $O(|M| \cdot |p'|^2)$, *provided that each fairness formula in p' is in canonical form.*

8. Finite automata on infinite strings

There is an extensive literature for finite automata on infinite strings, and the reader is referred to [22, 20, 32, 33, 35, 38, 11] for detailed discussions. In order to present our results regarding testing emptiness, we content outselves with a brief review of the following definitions:

A *finite automaton* $\mathscr{A}$ *on infinite strings* consists of a tuple (Σ, S, δ, s_0) where Σ is the finite *input alphabet*, S the finite set of *states*, $\delta : S \times \Sigma \to PowerSet(S)$ the *transition function*, and $s_0 \in S$ the *start state*, plus an *acceptance condition* as described subsequently. A *run* r of $\mathscr{A}$ on infinite input string $x = a_1 a_2 a_3 \ldots$ is an infinite sequence $r = s_0 s_1 s_2 s_3 \ldots$ of states such that $\forall i \geq 0\ \delta(s_i, a_{i+1}) \supseteq \{s_{i+1}\}$. We let $In\, r$ denote the set of states in S that appear infinitely often along r. For a *Buchi* automaton acceptance is defined in terms of a distinguished set of states, GREEN: x is *accepted* iff there exists a run r on x such that $In\, r \cap \mathrm{GREEN} \neq \emptyset$. Acceptance for a *pairs* automaton is defined with respect to a finite list

$((\text{RED}_1, \text{GREEN}_1), \ldots, (\text{RED}_k, \text{GREEN}_k))$ of pairs of sets of state: x is *accepted* iff there exists a run r on x such for some pair $i \in [1:k]$ $In\, r \cap \text{RED}_i = \emptyset$ and $In\, r \cap \text{GREEN}_i \neq \emptyset$. For a *complemented pairs* automaton acceptance is also defined using a finite list of pairs $((\text{RED}_1, \text{GREEN}_1), \ldots, (\text{RED}_k, \text{GREEN}_k))$. However, it *accepts* input x iff there exists a run r on x such that the above pairs condition is false. Finally, for a *designated subsets* automaton acceptance is defined in terms of a family $\mathcal{F} = \{S_1, \ldots, S_k\} \subseteq PowerSet(S)$ of subsets of S. It accepts iff there exists a run r such that $In\, r \in \mathcal{F}$.

All but the last acceptance condition can be readily visualized in terms of flashing lights controlled by the automaton. If we think of a Buchi automaton flashing a green light upon entering any state in GREEN, then it accepts iff there exists a run which causes the green lights to flash infinitely often. We can think of a pairs automaton as having pairs of colored lights where the red light of the ith pair is flashed upon entering any state of set RED_i, and the green light of the ith pair is flashed upon entering any state of GREEN_i, etc.; then we see that it accepts iff there exists a run which causes, for some pair $i \in [1:k]$ the RED_i light to flash only finitely often and the GREEN_i light to flash infinitely often. Analogously, a complemented pairs automaton is seen to accept iff for all pairs $i \in [1:k]$, there exist infinitely many flashes of the GREEN_i light implies that there exist infinitely many flashes of the RED_i light.

In order to test emptiness of a finite automaton $\mathcal{A} = (\Sigma, S, \delta, s_0)$, we note that its transition diagram may be viewed as defining a finite structure $M = (S, A, L)$. There is an arc (s, t) in A iff there is an arc (s, t) in the transition diagram. (Note: because we are testing emptiness, we can ignore the symbol labelling the arc (s, t) in $\mathcal{A}$. All we care about is if there exists an accepted string, not what string is accepted.) If $\mathcal{A}$ is, say, a pairs automaton with acceptance condition given by $((\text{RED}_1, \text{GREEN}_1), \ldots, (\text{RED}_k, \text{GREEN}_k))$, we let the underlying set of atomic propositions for M be $\{P\text{-RED}_1, P\text{-GREEN}_1, \ldots, P\text{-RED}_k, P\text{-GREEN}_k\}$. Then let $P\text{-RED}_i$ appear in $L(s)$ for exactly those $s \in \text{RED}_i$ and similarly for $P\text{-GREEN}_i$, $i \in [1:k]$. The runs r of $\mathcal{A}$ thus correspond to paths in M starting from s_0. It is easy to see that for each $i \in [1:k]$, $In\, r \cap \text{GREEN}_i \neq \emptyset$ iff $M, r \models \overset{\infty}{\text{F}} P\text{-GREEN}_i$ and $In\, r \cap \text{RED}_i = \emptyset$ iff $M, r \models \overset{\infty}{\text{G}} \neg P\text{-RED}_i$. Thus a run r of $\mathcal{A}$ is accepted by the pairs criterion iff the run r viewed as a path in M (which starts at s_0) satisfies the fairness constraint $\Phi_0 = \bigvee_{i \in [1:k]} \overset{\infty}{\text{G}} \neg P\text{-RED}_i \wedge \overset{\infty}{\text{F}} P\text{-GREEN}_i$. So there is an accepting run of $\mathcal{A}$ on some input iff there is a path starting at s_0 in M meeting the fairness constraint Φ_0. Thus, emptiness of $\mathcal{A}$ may be tested by running the FSP algorithm with input M and Φ_0, and then checking to see if s_0 is fair with respect to Φ_0. Similarly, for a complemented pairs automaton with acceptance condition $((\text{RED}_1, \text{GREEN}_1), \ldots, (\text{RED}_k, \text{GREEN}_k))$, the corresponding fairness condition is $\bigwedge_{i \in [1:k]} (\overset{\infty}{\text{F}} P\text{-GREEN}_i \Rightarrow \overset{\infty}{\text{F}} P\text{-RED}_i)$. And for a Buchi automaton with acceptance condition GREEN, the fairness condition is simply $\overset{\infty}{\text{F}} P\text{-GREEN}$.

We can also handle a designated subsets automaton with acceptance condition $\mathcal{F} = \{S_1, \ldots, S_k\}$ where each $S_i = \{s_{i1}, s_{i2}, \ldots, s_{in_i}\}$. We let the underlying atomic

propositions be $P\text{-}S_i$, $P\text{-}s_{i1}$, $P\text{-}s_{i2}, \ldots, P\text{-}s_{in_i}$, $i \in [1:k]$, where $P\text{-}S_i$ appears in $L(s)$ for exactly those $s \in S_i$ and each $P\text{-}s_{ij}$ appears only in $L(s_{ij})$. Then the corresponding fairness specification $\Phi_0 = \bigvee_i (\overset{\infty}{\mathrm{F}} P\text{-}s_{i1} \wedge \overset{\infty}{\mathrm{F}} P\text{-}s_{i2} \wedge \cdots \wedge \overset{\infty}{\mathrm{F}} P\text{-}s_{in_i} \wedge \overset{\infty}{\mathrm{G}} P\text{-}S_i)$. In each disjunct the $\overset{\infty}{\mathrm{F}}$ formulae ensure that $S_i \subseteq In\ r$ while the $\overset{\infty}{\mathrm{G}}$ formula ensures that $In\ r \subseteq S_i$.

We leave it to the reader to check that each of these fairness specifications corresponding to an automaton acceptance condition can be succinctly massaged into the canonical form thereby showing that emptiness for any of these acceptance conditions can be tested in (small) polynomial time.

9. Conclusion

We have shown that model checking under fairness assumptions can be handled readily in the framework of branching time. In particular, we have presented a unified approach for efficient model checking under a broad class of generalized fairness constraints in a branching time temporal logic. Our method applies to any type of fairness expressed in the canonical form $\Phi_0 = \bigvee_{i=1}^{n} \bigwedge_{j=1}^{n_i} (\overset{\infty}{\mathrm{G}} p_{ij} \vee \overset{\infty}{\mathrm{F}} q_{ij})$. Since almost all 'practical' types of fairness from the literature, including the fundamental notions of impartiality, weak fairness, and strong fairness, can be succinctly written in our canonical form, our approach is potentially of wide applicability. Moreover, our branching time approach can easily be adapted to handle types of fairness (such as fair reachability of a predicate) which cannot even be expressed in a linear temporal logic.

We then showed that the problem of model checking in a branching time logic can be efficiently reduced to the problem of model checking in a linear temporal logic: given a model checking algorithm for a system of linear time logic (in particular, for the usual system of linear time logic over F, G, X, and U), there is a model checking algorithm of the same order of complexity (in both the structure and formula size) for the corresponding full branching time logic which trivially subsumes the linear time logic in expressive power (in particular, for the system of *full* branching time logic CTL* in which the basic modalities are of the form: A or E followed by an *unrestricted* formula of linear time logic over F, G, X, and U). Thus, the real issue involved for model checking is not whether to use branching time or linear time, but simply: what are the basic modalities of my branching time logic? I.e., what linear time formulae can follow the path quantifiers? (Remark: In a *basic modality* of a branching time logic, the linear time formula following the path quantifier is a 'pure' linear time formula involving no nested path quantifiers.) The results of [34] show that when an arbitrary combination (i.e., allowing boolean connectives and nesting) of linear time operators is allowed, the model checking problem is PSPACE-complete. And, as we should expect, for the algorithm of [19] it is indeed the linear formula (following the implicit path quantifier) which causes the exponential blowup in the complexity of model checking for linear time logic

(and for CTL*). At the other extreme, as we might expect, [5] shows that model checking is easy for the simple modalities of CTL where only a single linear time operator is allowed to follow a path quantifier. When we consider modalities of intermediate structural complexity, the results of [34] show that model checking is NP-hard even for linear time logic over just F and G. It is quite surprising, however, to note that, while [34] shows that even for the simple modality $\mathrm{E}[\mathrm{F}P_1 \wedge \cdots \wedge \mathrm{F}P_n]$ the modal checking problem is NP-hard, for the apparently closely related modalities $\mathrm{E}[\overset{\infty}{\mathrm{F}}P_1 \wedge \cdots \wedge \overset{\infty}{\mathrm{F}}P_n]$ and $\mathrm{E}[\overset{\infty}{\mathrm{G}}P_1 \wedge \cdots \wedge \overset{\infty}{\mathrm{G}}P_n]$ model checking can be done in linear time. (The first modality is related to the second because FP means 'there exists at least one state satisfying P' while $\overset{\infty}{\mathrm{F}}P$ means 'there exist infinitely many states satisfying P'; the first modality is related to the third because $\overset{\infty}{\mathrm{G}}P$ is equivalent to FGP.)

Thus, the infinitary operators $\overset{\infty}{\mathrm{F}}$ and $\overset{\infty}{\mathrm{G}}$ used in describing fairness properties, which are often thought of as causing all sorts of problems with discontinuities, non-definability in first-order arithmetic, etc. (cf. [7, 14]), can actually simplify the problem of model checking. In trying to account for why $\overset{\infty}{\mathrm{F}}$ and $\overset{\infty}{\mathrm{G}}$ seem easier to handle than expected one notices that these modalities satisfy the property of being oblivious to the addition to or deletion of finite prefixes of paths (Observation 4.1). Indeed, this property was used in an essential way in our polynomial time model checkers FCTL and GFCTL. One is therefore tempted to attribute the nice behavior of these modalities to the obliviousness property. However, one notices that any boolean combinations of $\overset{\infty}{\mathrm{F}}$ and $\overset{\infty}{\mathrm{G}}$ enjoys this obliviousness, and we have already exhibited (Theorem 4.7) a very simple boolean combination $(\bigwedge_{i=1}^{n}(\overset{\infty}{\mathrm{G}}p_i \vee \overset{\infty}{\mathrm{G}}q_i))$ which is NP-complete.

It thus appears that the relationship between the structural complexity of the basic modalities and the computational complexity of the associated model checking problem is a rather suble one. We encourage additional research to enhance our understanding of this issue.

Acknowledgment

We would like to thank the referees for their extremely thorough reading of the previous version of the manuscript which caught several rather subtle technical glitches, and for suggesting significant improvements to the presentation. We would also like to thank the members of Edsger W. Dijkstra's Austin Tuesday Afternoon Club for helpful criticisms and comments, as well as Graham Gough.

Appendix

Proof of Proposition 4.2. For equivalence (1), $M, s_0 \models_{\Phi_0} \mathrm{E}_{\Phi}\mathrm{X}p$ iff $M, s_0 \models \mathrm{E}[\Phi_0 \wedge \mathrm{X}p']$ (by definition of $\models_{\Phi_0}$) iff $M, s_0 \models \mathrm{E}[\mathrm{X}(\Phi_0 \wedge p')]$ (by Observation 4.1) iff

$M, s_0 \models \mathrm{EXE}(\Phi_0 \wedge p')$ (by CTL* semantics) iff $M, s_0 \models \mathrm{EX}(\mathrm{E}\Phi_0 \wedge p')$ (because p' is a state formula) iff $\exists (s_0, s_1) \in R$ $M, s_1 \models [p' \wedge \mathrm{E}\Phi_0]$ iff $\exists (s_0, s_1) \in R$ $M, s_1 \models_{\Phi_0} [p \wedge \mathrm{E}_\Phi \mathrm{X} true]$.

For equivalence (2), $M, s_0 \models_{\Phi_0} \mathrm{E}_\Phi [p \mathrm{U} q]$ iff $M, s_0 \models \mathrm{E}[\Phi_0 \wedge (p' \mathrm{U} q')]$ (by definition of $\models_{\Phi_0}$) iff $M, s_0 \models \mathrm{E}[p' \mathrm{U} (q' \wedge \Phi_0)]$ (by Observation 4.1) iff $M, s_0 \models \mathrm{E}[p' \mathrm{U} \mathrm{E}(q' \wedge \Phi_0)]$ (by CTL* semantics) iff $M, s_0 \models \mathrm{E}[p' \mathrm{U} (\mathrm{E}q' \wedge \Phi_0)]$ (since q' is a state formula) iff $\exists k \geq 0$ $\exists$ a finite path $(s_0, \ldots, s_k)$ in M such that $\forall i$, if $0 \leq i < k$, then $M, s_i \models_{\Phi_0} p$ and $M, s_k \models_{\Phi_0} (q \wedge \mathrm{E}_\Phi \mathrm{X} true)$.

Finally, for equivalence (3) we first note the linear time equivalence $M, x \models \neg(p \mathrm{U} q)$ iff $M, x \models (\neg q \mathrm{U} (\neg p \wedge \neg q)) \vee \mathrm{G}\neg q$. Plainly, the right-hand side implies the left. To see the converse, if $(p \mathrm{U} q)$ is false on x, then either q is always false—$\mathrm{G}\neg q$—or there is a first time q is true, but prior to that time, i.e., in an initial interval where q is false, p is false somewhere—$(\neg q \mathrm{U} (\neg p \wedge \neg q))$. Thus, $M, s \models_{\Phi_0} \mathrm{E}_\Phi [\neg(p \mathrm{U} q)]$ iff $M, s \models_{\Phi_0} \mathrm{E}_\Phi ([\neg q \mathrm{U} (\neg p \wedge \neg q)] \vee \mathrm{G}\neg q)$ iff $M, s \models_{\Phi_0} \mathrm{E}_\Phi [\neg q \mathrm{U} (\neg p \wedge \neg q)] \vee \mathrm{E}_\Phi \mathrm{G}\neg q$. □

Proof of Theorem 4.7 (continued). [On if part]: Assume that g is satisfiable. Since g is satisfiable, there exists a truth assignment $\mathscr{A}$ such that g is true under $\mathscr{A}$, 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 states $s, v_{1j_1}, \ldots, v_{nj_n}, t$, such that for all i, l_{ij_i} is true under the assignment $\mathscr{A}$.

We will show that $M, z \models \Phi_0$ by showing that $\overset{\infty}{\mathrm{G}}\neg p_k \vee \overset{\infty}{\mathrm{G}}\neg q_k$ holds on z for every $k \in [1, n]$. If $\overset{\infty}{\mathrm{G}}\neg p_k$ holds on z, then $\overset{\infty}{\mathrm{G}}\neg p_k \vee \overset{\infty}{\mathrm{G}}\neg q_k$ also holds on z. Hence, we only have to show that when $\overset{\infty}{\mathrm{G}}\neg p_k$ does not hold on z, $\overset{\infty}{\mathrm{G}}\neg p_k \vee \overset{\infty}{\mathrm{G}}\neg q_k$ still holds on z. Because $\overset{\infty}{\mathrm{G}}\neg p_k$ does not hold on z, there must be some state v in z such that $p_k \in L(v)$. Note that $L(s) = L(r) = \emptyset$. 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, i.e., the assignment $\mathscr{A}$ assigns true to x_k. Hence $\neg x_k$ is false under $\mathscr{A}$. Thus, if $v_{ij} \in z$, then $l_{ij} \neq \neg x_k$ so $q_k \notin L(v_{ij})$ by the definition of L. Hence $\overset{\infty}{\mathrm{G}}\neg p_k \vee \overset{\infty}{\mathrm{G}}\neg q_k$ 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 $\bigwedge_{i=1}^{n} (\mathrm{G}\neg p_k \vee \mathrm{G}\neg q_k)$ holds on z'. Note that either p_k or q_k does not appear on the label of any state on z'. Consider the truth assignment $\mathscr{A}: x_k \to \{\mathrm{T}, \mathrm{F}\}$ as follows:

$$\mathscr{A}(x_k) = \begin{cases} \mathrm{T} & \text{if } \exists i, j [p_k \in L(v_{ij}) \text{ and } v_{ij} \in z'], \\ \mathrm{F} & \text{otherwise.} \end{cases}$$

It is quite easy to check that $\mathscr{A}$ 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 $\mathscr{A}$. Hence g is satisfiable. This completes our proof. □

Note added in proof. We refer the reader to some recent work extending our results of Section 8: H. Yen and L. Rosier, Logspace hierarchies, polynomial time and the complexity of fairness problems concerning ω-machines, *SIAM J. Comput.*, to appear.

References

[1] K. Abrahamson, Decidability and expressiveness of logics of processes, PhD Thesis, University of Washington, 1980.

[2] M. Antila, H. Erikkson, J. Ikonen, R. Kujansuu, L. Ojala and H. Tuominen, Tools and studies of formal techniques—Petri nets and temporal logic, in: H. Rudin and C. West, Eds., *Protocol Specification, Testing, and Verification III* (North-Holland, Amsterdam, 1983).

[3] K. Bartlett, R. Scantlebury and P. Wilkinson, A note on reliable full-duplex transmission over half-duplex links, *Comm. ACM* **12** (5) (1969) 260-261.

[4] E.M. Clarke and E. A. Emerson, Design and synthesis of synchronization skeletons using branching time temporal logic, *IBM Logics of Programs Workshop*, Lecture Notes in Compute Science **131** (Springer, Berlin, 1981) 52-71.

[5] E.M. Clarke, E.A. Emerson and A.P. Sistla, Automatic verification of finite state concurrent systems using temporal logic, *ACM Trans. Programming Languages and Systems* **8** (2) (1986) 244-263.

[6] E.M. Clarke and B. Mishra, Automatic verification of asynchronous circuits, *CMU Logics of Programs Workshop*, Lecture Notes in Computer Science **164** (Springer, Berlin, 1983) 101-115.

[7] E.A. Emerson and E.M. Clarke, Characterizing correctness properties of parallel programs as fixpoints, *Proc. 7th International Colloquium on Automata, Languages, and Programming*, Lecture Notes in Computer Science **85** (Springer, Berlin, 1983).

[8] E.A. Emerson and E.M. Clarke, Using branching time temporal logic to synthesize synchronization skeletons, *Sci. Comput. Programming* **2** (1982) 241-266.

[9] E.A. Emerson and J.Y. Halpern, Decision procedures and expressiveness in the temporal logic of branching time, *J. Comput. System Sci.* **30** (1) (1985) 1-24.

[10] E.A. Emerson and J.Y. Halpern, 'Sometimes' and 'not never' revisited: On branching versus linear time temporal logic, *J. ACM* **33** (1) (1986) 151-178.

[11] E.A. Emerson and A.P. Sistla, Deciding full branching time logic, *Information and Control* **61** (3) (1984) 175-201.

[12] M.J. and R.E. Ladner, Propositional dynamic logic of regular programs, *J. Comput. System Sci.* **18** (1979) 194-211.

[13] N. Francez and D. Kozen, Generalized fair termination, *Proc. 11th Annual ACM Symposium on Principles of Programming Languages* (1984) 46-53.

[14] D. Harel, A general result on infinite trees and its applications, *Proc. 16th STOC* (1984) 418-427.

[15] S. Hart and M. Sharir, Probabilistic temporal logics for finite and bounded models, *Proc. 16th STOC* (1984) 1-13.

[17] L. Lamport, 'Sometimes' is sometimes 'not never',—on the temporal logic of programs, *Proc. 7th Annual ACM Symposium on Principles of Programming Languages* (1980) 174-185.

[18] D. Lehmann, A. Pnueli and J. Stavi, Impartiality, justice and fairness: The ethics of concurrent termination, *Proc. ICALP '81*, Lecture Notes in Computer Science **115** (Springer, Berlin, 1981) 204-277.

[19] O. Lichtenstein and A. Pnueli, Checking that finite state concurrent programs satisfy their linear specification, *Proc. POPL '85* (1985) 97-107.

[20] R. McNaughton, Testing and generating infinite sequences by a finite automaton, *Information and Control* **9** (1966).

[21] C. Mead and L. Conway, *Introduction to VLSI Systems* (Addison-Wesley, Reading, MA, 1980).

[22] D.E. Muller, Infinite sequences and finite machines, *Proc. 4th Annual IEEE Symposium of Switching Theory and Logical Design* (1963) 3-16.

[23] L. Ojala, Personal Communication at *ICALP '84*, 1984.
[24] S.S. Owicki and L. Lamport, Proving liveness properties of concurrent programs, *ACM Trans. Programming Languages and Systems* **4** (3) (1982) 455-495.
[25] D. Park, On the semantics of fair parallelism, *Abstract Software Specification*, Lecture Notes in Computer Science **86** (Springer, Berlin, 1980) 504-524.
[26] A. Pnueli, The temporal logic of programs, *Proc. 19th Annual symposium on Foundations of Computer Science* (1977).
[27] A. Pnueli, On the extremely fair termination of probabilistic algorithms, *Proc. 15th Annual ACM Symposium on Theory of Computing* (1983) 278-290.
[28] A. Pnueli, Linear and branching structures in the semantics and logics of reactive systems, *Proc. 12th ICALP* (1985) 15-32.
[29] V. Pratt, Semantical considerations on Floyd-Hoare logic, *Proc. 17th FOCS* (1976) 109-121.
[30] V. Pratt, A dedicable Mu-calculus, *Proc. 22nd FOCS* (1981) 421-427.
[31] J.P. Queille and J. Sifakis, Fairness and related properties in transition systems, *Acta Informat.* **19** (1983) 195-220.
[32] M. Rabin, Decidability of second order theories and automata on infinite trees, *Trans. Amer. Math. Soc.* **141** (1969) 1-35.
[33] M. Rabin, Automata on infinite trees and the synthesis problem, Hebrew University, Technical Report no. 37, 1970.
[34] A.P. Sistla and E.M. Clarke, The complexity of propositional linear temporal logic, *J. ACM* **32** (3) (1985) 733-749.
[35] R. Streett, Propositional dynamic logic of looping and converse, *Information and Control* **54** (1982) 121-141
[36] R. Streett and E.A. Emerson, The propositional Mu-calculus is elementary, *Proc. ICALP '84* (1984) 465-472.
[37] R.E. Tarjan, Depth-first search and linear graph algorithms, *SIAM J. Comput.* **1** (1972) 146-160.
[38] M. Vardi and P. Wolper, Automata theoretic techniques of modal logics of programs, *J. Comput. System Sci.* **32** (1986) 183-221.