MODALITIES FOR MODEL CHECKING: BRANCHING TIME STRIKES BACK (Extended Abstract)

E. Allen EMERSON[1] and Chin-Laung LEI

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

1. Introduction

It is a point of continuing controversy in the computer science community as to whether branching time or linear time temporal logic is more appropriate for reasoning about concurrent programs (cf. [LA80], [EH83]). 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 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 [BSW69]) 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

[1]Work supported in part by NSF Grant MCS8302878

single occurrence of F, G, X, or U), Clarke, Emerson, and Sistla [CES83] give an algorithm that runs in time $O(|M|\cdot|p|)$ 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 ([SC82]) to be PSPACE-complete.

On the other hand, while fairness is readily handled in linear temporal logic, it is known (cf. [LA80], [EH83]) that the branching time logic CTL used in [CES83] does not permit reasoning under certain common fairness assumptions. A partial remedy to this problem is given in [CES83] 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 [LP84] 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 $O(|M|^2\cdot\exp(|p|))$ which is *quadratic* in the structure size but *exponential* in the formula length. They then claim that, in practice, the 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 it allows reasoning about types of fairness not handled by [CES83].

In this paper, we 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 present a model checking algorithm which permits efficient (actually still linear time) mechanical reasoning in a branching time framework under 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). FCTL is a branching time system which generalizes the (ordinary) CTL as used in [CES83] by having all path quantifiers relativized to a fairness assumption Φ_o specified by a boolean combination of the infinitary linear time temporal operators $\overset{\infty}{F}p$ (which abbreviates GFp and means "infinitely often p") and $\overset{\infty}{G}p$ (which abbreviates FGp and means "almost everywhere p"). Its basic modalities are thus of the form A_Φ ("for all fair paths") or E_Φ ("for some fair path") followed by a single linear time operator: F, G, X, or U. The infinitary operators of Φ_o make it possible to express and reason under a wide variety of "practical" fairness assumptions from the literature including im-

partiality [LPS81], weak fairness ([LA80]), strong fairness ([LA80]), fair reachability of predicates ([QS83]), state fairness ([PN83]), as well as technical notion of "limited looping" fairness ([AB80]).

To develop our FMCP algorithm, we will first argue that FMCP can be reduced in linear time to the *Fair State Problem* (FSP): Starting from which states does there exist some path along which Φ_o holds? Our reduction applies for any fairness specification Φ_o involving a boolean combination of the $\overset{\infty}{F}$, $\overset{\infty}{G}$ operators as above. We then show that when Φ_o is in the special *canonical form* $\bigvee_{i=1}^{n} \bigwedge_{j=1}^{n_i} (\overset{\infty}{G}p_{ij} \vee \overset{\infty}{F}q_{ij})$, then FSP (and hence also FMCP) can be solved in time linear in the size of the input specification and input structure. While any Φ_o can be translated into an equivalent Φ_o' 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 most all "practical" types of fairness considered in the literature (including all those listed above) can be directly specified using a canonical Φ_o. On the other hand, we are able to classify the complexity of FSP and FMCP for an arbitrary Φ_o: they are NP-complete.

Finally, we argue that the results discussed above strongly suggest that 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 [SC82] 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 [LP84] 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, [CES83] 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 [SC82] 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 [SC82] shows that even for the simple modality $E[FP_1 \wedge ... \wedge FP_n]$ the model checking problem is NP-hard, for the apparently closely related modalities $E[\overset{\infty}{F}P_1 \wedge ... \wedge \overset{\infty}{F}P_n]$ and $E[\overset{\infty}{G}P_1 \wedge ... \wedge \overset{\infty}{G}P_n]$ 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 $\overset{\infty}{F}P$ means "there exist infinitely many states satisfying P"; the first modality is related to the third because $\overset{\infty}{G}P$ is equivalent to FGP.) Thus, the infinitary operators $\overset{\infty}{F}P$ and $\overset{\infty}{G}P$ 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. [EC80], [HA84]) can actually simplify the problem of model checking. Indeed, we are able to obtain linear time complexity model checking algorithms under a broad class of practical fairness assumptions.

The work described so far may be viewed as extending the model checking approach originally developed for CTL to various (increasingly expressive) sublanguages of CTL*, each of which

subsumes CTL in expressive power. There is another way to think of extending model checking for CTL. View CTL not as a sublanguage of CTL*, but, rather, as a sublanguage of the *propositional Mu-calculus* (cf. [KO83], [PR81], [EC80]). The propositional Mu-calculus provides a *least fixpoint* operator (μ) and a *greatest fixpoint* operator (ν) which make it possible to give *fixpoint characterizations* of the branching time modalities. Intuitively, the Mu-Calculus makes it possible to characterize the modalities in terms of recursively defined tree-like patterns. For example, EFP = $\mu Z.P \vee EXZ$, the least fixpoint of the functional $P \vee EXZ$ where Z is an atomic proposition variable (intuitively ranging over sets of states.) Similarly, $\overset{\infty}{EF}P = \nu Z_1.\mu Z_2.EX[(P \wedge Z_1) \vee Z_2]$. (Alternating nestings of μ's and ν's as in this latter example can lead to discontinuities.) We show how model checking can be done for the Mu-calculus. In particular, for the fragment of the Mu-calculus where the depth of nesting of alternating μ's and ν's as above is bounded by k, our algorithm runs in polynomial time with the degree of the polynomial proportional to k. (This makes it possible to model check PDL-Δ ([ST81]) in quadratic 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 gives the reduction of the model checking problem for full branching time logic to that for the corresponding linear time logic. Section 5 describes how to do efficient model checking in the branching time FCTL system whenever the fairness constraint is in canonical form. The complexity of the general case is also analyzed. A variety of types of practical fairness are defined and canonically specified in Section 6. Section 7 describes how one may apply FSP for testing emptiness of finite automata on infinite strings. Our algorithm for model checking in the Mu-calculus is described in section 8. Finally, extensions to this work are considered in the concluding section 9.

2. Practical Utility 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 claim that our model checking approach transcends this dichotomy, and enjoys the best features of both categories. Our method has formal 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.

Our method also has considerable practical utility as has been *empirically* demonstrated. In particular, the model checking method as described in [CES83] 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 [OL82]. It has also been successfully applied to the verification of VLSI circuits. In particular, [CM83] describes how the EMC system was used to detect an error in a circuit from Conway and Meade's VLSI text 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 ([OJ84]; cf. [AE83]). We believe that our model checking algorithm, because of its linear complexity, may also be suitable for similar applications.

3. Syntax and Semantics of Temporal Logics

We inductively define a class of state formulae (true of false of states) which intuitively correspond to branching time logic and a class of path formulae (true or false of paths) which intuitively correspond 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 abbreviations in the usual way: $p \vee q$ for $\neg(\neg p \wedge \neg q)$, $p \Rightarrow q$ for $\neg p \vee q$, Ap for $\neg E \neg p$, Fp for *true* U p, Gp for $\neg F \neg p$, $\overset{\infty}{F} p$ for GFp, $\overset{\infty}{G} p$ for $\neg \overset{\infty}{F} \neg p$, etc.

We define the semantics of a formula with respect to a structure M = (S, R, L) where

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

A *fullpath* $(s_1,s_2,s_3,\ldots)$ 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 path and x^i denotes the suffix path $(s_i,s_{i+1},s_{i+2},\ldots)$:

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. $s \models Ep$ 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 $x \models p$
    $x \models \neg p$ iff not $(x \models p)$
P3. $x \models Xp$ iff $x^1 \models p$
    $x \models (p\ U\ q)$ iff for some $i \geq 0$, $x^i \models q$ and
      for all $j \geq 0$ $[j \leq i$ 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 a in M, $M,a \models p$. We say that state formula p is *satisfiable* if for some structure M and some state s in M, $M,a \models p$. 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).

FCTL (Fair CTL) is defined as follows: An FCTL specification (p_o, Φ_o) consists of a functional assertion p_o, which is a state formula, and an underlying fairness assumption Φ_o, which is a path formula. The functional assertion p_o 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_o, and to syntactically distinguish FCTL from CTL. A fairness constraint Φ_o is a boolean combination of the infinitary linear time operators $\overset{\infty}{F}p$ ("infinitely often p") and $\overset{\infty}{G}p$ ("almost always p"), applied to (for simplicity) propositional arguments. We now define the semantics of an FCTL specification (p_o, Φ_o). Φ_o is a path formula, in a restricted syntax specialized to describing fairness properties, so by expanding the abbreviations $M,x \models \Phi_o$ is defined by the rules S1,P1,P2,P3. We can then view a subformula such as $A_\Phi FP$ of functional assertion p_o as an abbreviation for the CTL* formula $A[\Phi_o \Rightarrow Fp]$. Similarly, $E_\Phi GP$ abbreviates $E[\Phi_o \wedge GP]$. Note that *all path quantifiers in the functional assertion are relativized to the same (single) underlying fairness constraint* Φ_o. 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_o. Thus, when we mention the *length* of p_o, we refer to the unexpanded formula.

In practice, 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. It is interesting to note from a technical standpoint, however, that we can also define a *Generalized Fair CTL (GFCTL)* as simply a sublanguage of CTL* with basic modalities such as $A[\Phi_o \Rightarrow FP]$ and $E[\Phi_1 \wedge GP]$ where each such A or E subformula is associated with a (possibly) different fairness specification Φ_i. Moreover, the arguments to the $\overset{\infty}{F}$ and $\overset{\infty}{G}$ operators can be generalized to be arbitrary GFCTL state formulae.

Remark: It is routine to give formal definitions of the syntax and semantics of CTL, FCTL, and GFCTL described in a manner similar to that used in [EH83]; these formal definitions will be given in the full paper.

4. Model Checking for the Full Branching Time Logic CTL*

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 or not $M,s \models p$ and label s with p or $\neg p$ accordingly. The *Model Checking Problem for linear time logic (LMCP)* can be similarly formulated as follows (cf. [SC82]): 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* nested path quantifiers A or E). Then determine for each state in S, whether there is a fullpath satisfying p starting at s, and label s with Ep or ¬Ep accordingly.[2] (Note that the [LP84] algorithm can be trivially modified to do this.)

Despite the superficially plausible intuition that handling, e.g., nested, alternating path quantifiers would make BMCP more difficult than LMCP, we have the following

Theorem 1: (With mild, technical conditions on its complexity,) if we are given an algorithm LMCA to solve LMCP for a linear logic (in particular, ordinary linear logic over F, G, X, and U), then we can design an algorithm BMCA to solve BMCP for the corresponding full branching time logic (in particular, CTL*) - which trivially subsumes the linear logic in expressive power - of the same order of complexity as LMCA.

proof idea: 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_o, 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_o is an atomic proposition P, then add ¬P to the label of each state s whose label does not contain P.
2. If p_o 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_o to the label of each state whose label contains both p and q.
3. If p_o is a negation ¬p of a state formula p, then recursively model check for p. Add ¬p to the label of each state not containing p.
4. If p_o is of the form Ep where p is a path formula, then let $Eq_1,...,Eq_k$ be the list of all "top level" proper E-subformulae of p. 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 or $\neg Eq_i$ as appropriate. Introduce a list of new, "fresh" atomic propositions $Q_1,...,Q_k$. Augment the labelling of each state s in the structure for each i, with Q_i if Eq_i holds at s and $\neg Q_i$ otherwise. 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 each state is labelled with Ep' or ¬Ep' as appropriate. Re-substitute Eq_i back for each Q_i so in the occurrences of Ep' to get each state labelled with Ep or ¬Ep appropriately.

Actually, in implementing the algorithm BMCA it is not necessary to introduce the auxiliary atomic propositions Q_i; rather, the corresponding Eq_i can be viewed as themselves atomic. So implemented, it is straightforward to check that if LMCA is of time complexity $O(f(|M|)\cdot g(|p|))$ for any "reasonable" functions f, g (such as polynomials or exponentials) then so is the recursive descent algorithm BMCA. In particular, the BMCP algorithm for CTL* resulting from the [LP84] algorithm for LMCP for ordinary linear temporal logic is of the same order of complexity. It is also easy to see that this reduction will work for any linear temporal formalism and its corresponding full branching temporal logic.

[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 of LMCP and then scanning all states to check whether Ap holds, we get a solution to the "alternative" formulation.

5. Model Checking for FCTL

The *Model Checking Problem* for FCTL (FMCP) is: Given a finite structure M and an FCTL specification (p_o, Φ_o), determine for each state $s \in S$ whether $M, s \models p_o'$ where p_o' is the CTL* formula that p_o abbreviates as explained in section 2. The *Fair State Problem* (FSP) is: Given a structure $M=(S, R, L)$, and a fairness constraint Φ_o, determine for each state $s \in S$ whether there is a fullpath x in M starting at s such that $M, x \models \Phi_o$ (i.e., whether $M,s \models E\Phi_o$).

5.1. Reduction of FMCP to FSP. FSP may be viewed as a special case of FMCP. However, we can generalize a method in [CES83] to reduce FMCP to FSP. The reduction yields an algorithm for for FMCP that runs in time linear in the size of the input (specification and structure) and the time to solve FSP. The reduction exploits the observation that, for any fairness constraint Φ_o and for any fullpaths x and y such that x is a suffix of y, $M, x \models \Phi_o$ iff $M, y \models \Phi_o$. We thus get the following equivalences:

(1) $M,s \models E_{\Phi}Xp$ iff $\exists(s,t)\in R[(M,t \models E\Phi_o)\wedge(M,t \models p)]$
(2) $M,s \models A_{\Phi}Xp$ iff $\forall(s,t)\in R[(M,t \models E\Phi_o)\Rightarrow(M,t \models p)]$
(3) $M,s \models E_{\Phi}[p\ U\ q]$ iff $M,s \models E[p\ U\ (q \wedge E\Phi_o)]$
(4) $M,s \models A_{\Phi}[p\ U\ q]$ iff $M,s \models \neg[E_{\Phi}(\neg q\ U\ (\neg p\wedge\neg q)) \vee E_{\Phi}G(\neg q)]$

The equivalences (1) and (2) are immediate; (3) follows from the observation above that fairness properties are oblivious to finite prefixes. To check $A_{\Phi}[p\ U\ q]$, equivalence (4) shows that we can first check whether $E_{\Phi}(\neg q\ U\ (\neg p \wedge \neg q))$ using equivalence (3). To next check $E_{\Phi}G(\neg q)$, we let M' be the substructure of M obtained by deleting all nodes where q holds (inductively, we assume nodes of M are labeled with the true subformulae). Then $E_{\Phi}G(\neg q)$ holds at a node s iff there is a finite path from s to a fair node t in M'. Detection of fair nodes is done by the algorithm for FSP. The reduction algorithm is described in detail in the full paper where we also show that if we let $T_A(M, \Phi_o)$ denote the time complexity of algorithm for FSP(M, Φ_o) then the reduction can be performed in time $O(|p_o|\cdot\max(|M|, T_A(M, \Phi_o)))$, so does the whole algorithm.

5.2. Efficient Algorithm for Fair State Problem. We will now develop a linear time algorithm for FSP when Φ_o is in the *(restricted) canonical form* $\Phi_o = \bigwedge_{i=1}^{n} (\overset{\infty}{F} p_i \vee \overset{\infty}{G} q_i)$. Since $E[p \vee q] \equiv Ep \vee Eq$, this will actually yield a linear time algorithm for FSP (and hence FMCP) when Φ_o is in the *(full) canonical form* $\bigvee_{i=1}^{n} \bigwedge_{j=1}^{n_i} (\overset{\infty}{F} p_{ij} \vee \overset{\infty}{G} q_{ij})$.

The first step is detection of *fair* components (a strongly connected component is fair if it contains a fair path). Given a strongly connected structure $C=(S,R, L)$, and a fairness constraint $\Phi_o = \bigwedge_{i=1}^{k} (\overset{\infty}{F} p_i \vee \overset{\infty}{G} q_i)$, we check if C is fair w.r.t. Φ_o as follows: if there is a fullpath in C satisfying all the $\overset{\infty}{F} p_i$ 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 nodes which do not satisfy q_j contains a component that is fair w.r.t the fairness constraint resulting from deleting the j^{th} conjunct of Φ_o. In the full paper we give a recursive implementation of this algorithm which runs in time $O(|C|\cdot|\Phi_o|)$.

Now to solve FSP, we first compute the fair components of the given structure M w.r.t. Φ_o. We then determine the fair states, i.e., those states which can reach a fair component. This can be done in time $O(|M|\cdot|\Phi_o|)$. We have thus established:

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

Proof. By the preceding remarks FSP can be solved for Φ_o in full canonical form in time $T_A = O(|M|\cdot|\Phi_o|)$. Then the reduction of FMCP to FSP solves FMCP in time $O(|p_o|\cdot\max(|M|,T_A)) = O(|p_o|\cdot|M|\cdot|\Phi_o|)$. □

Note that *any* arbitrary Φ_o can be placed in canonical form by first putting it in *Disjunctive Normal Form* (which can cause an exponential blowup) and then "padding" with $\overset{\infty}{F}$*false* or $\overset{\infty}{G}$*false* as needed. However, most all "practical" fairness specifications, as in the next section, can be massaged into canonical form with only a linear blowup.

5.3. Complexity of The General Case.

Theorem 3: FSP is NP-complete.

Proof. [NP-hardness:] We will reduce 3-SAT to FSP, with fairness constraint of the form $\bigwedge_{i=1}^{n} (\overset{\infty}{G}\neg p_i \vee \overset{\infty}{G}\neg q_i)$. Details are given in the full paper.

[Membership]: It has already been shown in [SC82] that the model checking problem for linear time temporal logic with F, and G operators can be solved in NP time, Hence FSP is in NP. □

Remark: In [SC82] it was shown that, in effect, FSP for Φ_o any arbitrary linear time formula over F, G is NP-complete. For FSP with Φ_o of the type we construct, membership in NP follows since our language of fairness constraints may be viewed as a sublanguage of linear time logic by the equivalences $\overset{\infty}{F}p \equiv GFp$ and $\overset{\infty}{G}p \equiv FGp$. But NP-hardness for Φ_o of our type does *not* follow from the proof in [SC82]. That proof involved a different reduction to a formula $Fp_1 \wedge \ldots \wedge Fp_n$. Because Fp is not expressible in our Φ_o language, such an argument cannot be applied. Since our Φ_o language has a more restricted syntax, its decision problem might be easier. Our NP-hardness argument shows that this is not the case.

Corollary 4. FMCP is NP-complete.

5.4. Handling GFCTL

We can define the model checking problem for GFCTL in the obvious way and show using the ideas above that it too can be solved in linear time.

6. Fairness Notions Expressible in FCTL

We can succinctly express the following fairness notions using our canonical form:

1. *Impartiality* [LPS81]: An infinite computation sequence is *impartial* iff every process is executed infinitely often during the computation. This notion can be expressed as $\bigwedge_{i=1}^{n} (\overset{\infty}{F}\mathit{executed}_i)$, where $\mathit{executed}_i$ is a proposition which asserts that process i is being executed.

2. *Weak Fairness* ([LA80]) (also known as *justice* [LPS81]): An infinite computation sequence is *weakly fair* iff every process enabled almost everywhere is executed infinitely often. The following FCTL formulae express weak fairness: $\bigwedge_{i=1}^{n} (\overset{\infty}{G}\mathit{enabled}_i \Rightarrow \overset{\infty}{F}\mathit{executed}_i) \equiv \bigwedge_{i=1}^{n} (\overset{\infty}{F}(\neg\mathit{enabled}_i) \vee \overset{\infty}{F}\mathit{executed}_i) \equiv \bigwedge_{i=1}^{n} (\overset{\infty}{F}(\neg\mathit{enabled}_i \vee \mathit{executed}_i))$

3. *Strong Fairness* ([LA80]) (called simply *fairness* in [LPS81]): 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: $\bigwedge_{i=1}^{n} (\overset{\infty}{F}\mathit{enabled}_i \Rightarrow \overset{\infty}{F}\mathit{executed}_i) \equiv \bigwedge_{i=1}^{n} (\overset{\infty}{G}\neg\mathit{enabled}_i \vee \overset{\infty}{F}\mathit{executed}_i)$

4. *Generalized Fairness* ([FK84]): Note that we can replace the propositions $\mathit{executed}_i$ and $\mathit{enabled}_i$ by any 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 $\mathcal{F} = ((P_1,Q_1), (P_2,Q_2), \ldots, (P_k,Q_k))$ 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 $\mathcal{F}$-fair* by $\bigwedge_{i=1}^{k} \overset{\infty}{\mathrm{F}} Q_i$, *weakly $\mathcal{F}$-fair* by $\bigwedge_{i=1}^{k} (\overset{\infty}{\mathrm{G}} P_i \Rightarrow \overset{\infty}{\mathrm{F}} Q_i)$, and *strongly $\mathcal{F}$-fair* by $\bigwedge_{i=1}^{k} (\overset{\infty}{\mathrm{F}} P_i \Rightarrow \overset{\infty}{\mathrm{F}} Q_i)$.

5. *Fair reachability of predicate P* ([QS83]): 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 $\overset{\infty}{\mathrm{F}}\mathrm{EFP} \Rightarrow \overset{\infty}{\mathrm{F}} P$. (Note: because EFP is not a pure propositional formula, this actually corresponds to a fairness specification of GFCTL rather than FCTL.)

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 a transition (i.e. traversal of an arc). To allow a precise differentiation between execution of transition actions and enabling of state conditions, we can extend the semantics of FCTL to be interpreted over PDL-like (cf. [PR76, FL79]) structures $M=(S,A_1,A_2,...,A_p,L)$ where each $A_i \subseteq S \times S$ represents (the atomic actions of) process i, and where we think of each each arc $(s_1,s_2) \in A = A_1 \cup ... \cup A_p$ as being labeled with the set $\{i: (s_1,s_2) \in A_i\}$ of processes which can cause a transition from state s_1 to state s_2. We can now extend the fairness specifications to allow atomic 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 be given a rigorous definition. It is straightforward to formalize this approach and to extend our efficient model checking algorithm to the extended semantics. Alternatively, we can encode the extended semantics with arc labels into the original semantic framework of only having node labels as is done in [PN77].

7. Finite Automata on Infinite Strings

In the full paper we describe an application of FSP to the theory of finite automata on infinite strings (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 ([VW84]). We describe how the emptiness problem for finite automata on infinite strings can be viewed as an instance of FSP. Moreover, for the common types of acceptance conditions (Buchi acceptance, pairs acceptance, and complemented pairs acceptance) the fairness condition Φ_0 for the corresponding instance of FSP is in our canonical form and the emptiness problem can be solved in linear time. (Designated subsets (Muller) acceptance can be handled in quadratic time.) Finally, we remark that our restricted canonical form corresponds to complemented pairs acceptance.

8. Model Checking in the Mu-Calculus

Formulas of the (endogenous) propositional Mu-Calculus are interpreted with respect to a structure M = (S, R, L) as in section 3. (The extension to the exogenous Mu-Calculus defined over PDL-like structures is routine). The formulae are built up using atomic proposition constants (P,Q,... etc.), atomic proposition variables (Y,Z,... etc.), the truth-functional connectives ($\wedge$, $\vee$, $\neg$), the nexttime operators (EXp, AXp where p is a subformula), and and the least fixpoint and greatest fixpoint operators (μY.p and νY.p, resp., where p is a subformula). We write p(Y) to indicate that free, atomic proposition Y is viewed as a variable ranging over PowerSet(S); p(Y) defines a mapping p': PowerSet(S) $\rightarrow$ PowerSet(S) in the obvious way. Thus, μY.p(Y) and νY.p(Y) denote the least fixpoint and greatest

fixpoint of the associated functional p', where p(Y) is required to be formally *monotonic* in variable Y, i.e., every free occurrence in Y occurs in the scope of an even number of negations ($\neg$). For example, we have the following fixpoint characterizations of CTL* modalities: $EFP = \mu Y.P \vee EXY$, $AGP = \nu Y.P \wedge AXY$, $AFP = \mu Y.P \vee AXY$, $E\overset{\infty}{F}P = \nu Y.\mu Z.EX[(P \wedge Y) \vee Z)$, and $A\overset{\infty}{G}P = \mu Y.\nu Z.AX[(P \vee Y) \wedge Z]$. We refer the reader to [EC80], [PR81], [KO83], [SE84] for additional details regarding the Mu-Calculus.

Our algorithm for model checking in the Mu-calculus operates by recursive descent and is similar to the original model checking algorithm for CTL given in [CE81]. Details are given in the full paper.

9. Extensions

While the following types of fairness can be succinctly expressed in the canonical form, it turns out that model checking for them can be done even more simply by merely modifying the algorithm.

6. *State Fairness* ([PN83]) (also called *fair choice from states* [QS83]): We say that an infinite computation x is *state fair for state s* provided that if s appears infinitely often along x, then every successor t in M of s also appears infinitely often along s. We say that x is *state fair* iff it is state fair for all s in M.

7. *"Limited Looping" Fairness* ([AB80]): 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. (Note: This fairness is closely related to but distinct from state fairness.)

Proposition 5. 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. Starting from state s, we use round-robin scheduling policy to choose the next state.

Observation 6. If fullpath x is state fair (limited looping fair), any fullpath y resulting from adding or deleting some finite prefix to/from x is still a state fair (resp., limited looping fair) path.

Due to proposition 5, FSP under the above two fairness notions becomes trivial. Furthermore, the model checking procedure for formulae of the form $A_\phi Xp$, $E_\phi Xp$, $E_\phi[pUq]$ reduce to exactly the same as the corresponding CTL formulae. To see how to do model checking for $A_\phi[pUq]$, recall that $A_\phi[pUq] \equiv \neg E_\phi G(\neg q) \vee \neg E_\phi[(\neg q)U(\neg p \wedge \neg q)]$. Hence we only have to describe how to check formulae of the form $E_\phi Gr$. 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 $E_\phi Gr$ iff there is a finite path starting from s leading to a terminal strongly connected component such that all nodes involved satisfy proposition r.

Finally, we remark that our method can be used to perform model checking for the probabilistic branching temporal logic PTL_f of [HS84] interpreted over finite Markov chains. The syntax of PTL_f is very similar to FCTL but an assertion such as $A_\phi Fp$ 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.

10. References

[AE83] Antila, M., Erikkson, H., Ikonen, J., Kujansuu, R., Ojala, L., Tuominen, H., Tools and Studies of Formal Techniques - Petri Nets and Temporal Logic, Protocol Specification, Testing, and Verification III, H. Rudin and C. West (editors), Elsevier North-Holland, IFIP, 1983.

[AB80] Abrahamson, K., Decidability and Expressiveness of Logics of Processes, PhD Thesis, University of Washington, 1980.

[BSW69] Bartlet, K., Scantlebury, R., and Wilkinson, P., A Note on Reliable Full-Duplex Transmission over Half-Duplex Links, Comm. of the ACM, vol. 12, no. 5, pp. 260-261, 1969.

[CE81] Clarke, E. M., Emerson, E.A., Design and Synthesis of Synchronization Skeletons Using Branching Time Temporal Logic, IBM Logics of Programs Workshop, Springer LNCS #131, pp. 52-71, May 1981.

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

[CM83] Clarke, E.M., Mishra, B., Automatic Verification of Asynchronous Circuits, CMU Logics of Programs Workshop, Springer LNCS #164, pp. 101-115, May 1983.

[EC80] Emerson, E. A., and Clarke, E. M., Characterizing Correctness Properties of Parallel Programs as Fixpoints. Proc. 7th Int. Colloquium on Automata, Languages, and Programming, Lecture Notes in Computer Science #85, Springer-Verlag, 1981.

[EC82] Emerson, E. A., and Clarke, E. M., Using Branching Time Temporal Logic to Synthesize Synchronization Skeletons, Tech. Report TR-208, Univ. of Texas, 1982.

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

[EH83] Emerson, E. A., and Halpern, J. Y., 'Sometimes' and 'Not Never' Revisited: On Branching versus Linear Time 10th Annual ACM Symp. on Principles of Programming Languages, January 1983.

[ES84] Emerson, E. A., and Sistla, A. P., Deciding Branching Time Logic, 16 Annual ACM Symp. on Theory of Computing, 1984.

[FL79] Fischer, M. J., and Ladner, R. E, Propositional Dynamic Logic of Regular Programs, JCSS vol. 18, pp. 194-211, 1979.

[FK84] Francez, N., and Kozen, D., Generalized Fair Termination, 11th Annual ACM Symp. on Principles of Programming Languages, 1984, pp. 46-53.

[HA84] Harel, D., A General Result on Infinite Trees and Its Applications, 16th STOC, pp. 418-427, May 84.

[HS84] Hart, S., and Sharir, M., Probabilistic Temporal Logics for Finite and Bounded Models, 16th STOC, pp. 1-13, 1984.

[KO83] Kozen, D., Results on the Propositional Mu-calculus, Theoretical Computer Science, pp. 333-354, December 83.

[LA80] Lamport, L., Sometimes is Sometimes "Not Never" - on the temporal logic of programs, 7th Annual ACM Symp. on Principles of Programming Languages, 1980, pp. 174-185.

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

[LP84] Lichtenstein, O. and Pnueli, A., Checking that Finite State Concurrent Programs Satisfy their Linear Specification, unpublished manuscript, July 84, (to appear this POPL85.)

[McN66] McNaughton, R., Testing and Generating Infinite Sequences by a Finite Automaton, Information and Control, Vol. 9, 1966.

[OJ84] Ojala, Leo, Personal Communication at ICALP84, July 1984.

[OL82] Owicki, S. S., and Lamport, L., Proving Liveness Properties of Concurrent Programs, ACM Trans. on Programming Languages and Syst., Vol. 4, No. 3, July 1982, pp. 455-495.

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

[PN83] Pnueli, A., On The Extremely Fair Termination of Probabilistic Algorithms, 15

Annual ACM Symp. on Theory of Computing, 1983, 278-290.

[PR76] Pratt, V., Semantical Considerations on Floyd-Hoare Logic, 17th FOCS, pp. 109-121, 1976.

[PR81] Pratt, V., A Decidable Mu-Calculus, 22nd FOCS, pp. 421-427, 1981.

[QS83] Queille, J. P., and Sifakis, J., Fairness and Related Properties in Transition Systems, Acta Informatica, vol. 19, pp. 195-220, 1983.

[RA69] Rabin, M., Decidability of Second order Theories and Automata on Infinite Trees, Trans. Amer. Math. Society, Vol. 141, pp. 1-35, 1969.

[RA70] Rabin, M., Automata on Infinite Trees and the Synthesis Problem, Hebrew Univ., Tech. Report no. 37, 1970.

[SC82] Sistla, A. P., and Clarke, E. M., The Complexity of Propositional Temporal Logic, 14 Annual ACM Symp. on Theory of Computing, 1982.

[ST81] Streett, R., Propositional Dynamic Logic of Looping and Converse (PhD Thesis), MIT Lab for Computer Science, TR-263, 1981. (a short version appears in STOC81)

[SE84] Streett, R., and Emerson, E. A., The Propositional Mu-Calculus is Elementary, ICALP84, pp 465 -472, July 84.

[VW84] Vardi, M. and Wolper, P., Automata Theoretic Techniques for Modal Logics of Programs, pp. 446-455, STOC84.