Reasoning about Fair Concurrent Programs
(extended abstract)

Constantin Courcoubetis
AT&T Bell Laboratories

Moshe Y. Vardi
IBM Almaden Research Center

Pierre Wolper
AT&T Bell Laboratories

Abstract

We investigate the use of branching time temporal logic to reason about fair programs. Our approach is novel in viewing fairness conditions as an intrinsic part of the computations rather than as a part of the formulas specifying the computations. Our first result is that the branching time logic of fair programs is the same as the branching time logic of probabilistic programs and as the logic of fusion- and suffix-closed programs defined by Abrahamson. Next we give decision procedures for the temporal logic of fair programs using a new type of automata on infinite trees. Matching upper and lower bounds for two different versions of branching time temporal logic are established.

1. Introduction

In concurrent programs, there is an inherent nondeterminism due to the fact that, at each moment, the next computation step can be done by any enabled process. It is convenient to imagine that this nondeterminism is embedded in a scheduler. This scheduler decides at each moment which process is going to perform the next step. While we want to make as few assumptions as possible about the scheduler, it is nevertheless reasonable to assume that it is not a pathological one. Indeed, almost all distributed protocols can be defeated by a sufficiently “evil” scheduler. Thus, it is usual to assume the scheduler to be fair, i.e., not to “discriminate” against any process. Equivalently, we can restrict our attention to fair computations, computations in which all processes have been scheduled in a fair manner. (Note that we are using here the term fairness generically; it has numerous formalizations (cf. [FK84, LFS81]).) The issue we address in this paper is the use of temporal logic to reason about fair programs.

Temporal logic was introduced by Pnueli [Pn77] and comes in two main variants. In linear temporal logic (LTL) [GSS80, Pn81], formulas are interpreted over linear sequences of program states. LTL formulas are constructed using temporal connectives such as next and until. In branching temporal logic (BTL) [BMP81, EC82, EH85, La80], on the other hand, formulas are interpreted over tree-like structures which can represent the possible computations of a concurrent or nondeterministic program. Here, we consider the branching version of temporal logic.
In BTL, formulas are built from quantifiers over paths of the branching structure and formulas describing properties of the quantified paths. A typical BTL formula is of the form $\exists \phi$, where $\phi$ is an LTL formula. $\exists \phi$ asserts that there exists a computation that satisfies $\phi$. BTL comes in many varieties, depending on the alternation between temporal connectives and quantifiers that the logic allows. One of the weakest logics is $CTL$, where there can be only one free temporal connective in the scope of a quantifier. One of the strongest logics is $CTL^*$, where there is no such restriction [EH83].

So far, temporal logic has been used to reason about fair programs by expressing the fairness constraints within the logic. That is, rather than asserting "there exists a correct computation", with the underlying semantics guaranteeing that such a computation will be fair, one asserts "there exists a fair and correct computation", with the underlying semantics enforcing no restrictions on computations [Em83]. In other words, rather than having path quantifiers range over sets of fair paths, they range over all paths with the fairness condition being part of quantified path formula. We find this solution quite unsatisfactory.

The first problem is that for this approach to work the logic has to be able to express fairness. $CTL$, for example, is not expressive enough, though it can express many correctness assertions. $CTL^*$ is expressive enough, but one pays the price for this expressiveness. While $CTL$ has a simple axiomatization and is decidable in exponential time [EH85] (which is considered to be a reasonable upper bound in the realm of propositional modal logics of programs), no axiomatization for $CTL^*$ is known, and the best known time upper bound for that logic is nondeterministic doubly exponential [Em85a, VS85]. Of course one can try to find logics between $CTL$ and $CTL^*$ that will be expressive enough to express fairness and not too expressive to render them intractable, but this is no small task (cf. [EL85a]).

Furthermore, we believe that mixing fairness and correctness is methodologically wrong. Correctness is a specification issue; it is the condition that the program has to meet in order to be useful. Fairness, on the other hand, is an implementation issue; it is a characteristic of the program being implemented. Indeed in program verification one checks that the program together with its fairness condition satisfies the correctness assertion [CES83, LP85]. To sum up, we contend that fairness is an intrinsic property of the computations of the program, independent of the correctness assertions we might make about these computations.

Consequently, we examine the interpretation of BTL over fair structures. A fair structure is essentially a state transition diagram together with a fairness condition. The fair computations are then those that satisfy the fairness condition, and one interprets path quantifiers as ranging over fair computations. A fairness condition is a condition about the limit behavior of the computation, for example, "every process is scheduled infinitely often" or "if a process is enabled infinitely often, then it is scheduled infinitely often".

A review of the literature brings up a few instances in which related approaches have been taken. Both Lamport [La80] and Clarke et al. [CES83] have looked at concurrent programs with a built-in fairness assumption, but they did not investigate the branching temporal logic of such structures. Abrahamson has considered the branching temporal logic (which he called MPL) of concurrent programs with certain built-in restrictions on the set of computations [Ab80]. He supplied an axiomatization and proved a doubly exponential time upper bound. It is not clear, however, what is the connection between his restrictions (suffix closure and fusion closure) and the fairness restriction (c.f. [Em83]). Finally, Lehman and Shelah looked at the branching temporal logic (which they called $TC$) of probabilistic programs [LS82]. Probabilistic programs can be viewed as implementing fairness through randomness. Lehman and Shelah supplied an axiomatization for $TC$, and some results (double exponential upper bound [Kr84] (see also [HS84, KL83]). Surprisingly, the axiomatization of $TC$ is identical to Abrahamson's axiomatization of his logic MPL, even though the underlying models are quite different.

Our first result relates the three classes of concurrent programs we have just mentioned: fair programs, suffix- and fusion-closed programs (à la Abrahamson), and probabilistic programs (à la Lehman and Shelah). Quite surprisingly, they have the same branching time logic. Specifically, a $CTL$ or $CTL^*$ formula is satisfiable by a fair structure if it is satisfiable by a
suffix and fusion-closed structure iff it is satisfiable by a probabilistic structure. Basically, our proof defines a notion of canonical structure and show that for each class of structures a formula is satisfiable by a structure in that class iff it is satisfiable by a canonical model.

Next, we study the complexity of the logic. For this, we use the automata-theoretic technique developed in [VW84]. Unfortunately, the type of tree automata usually used to decide standard branching time logic is unsuitable for deciding the branching time logic of fair programs. To overcome this difficulty we introduce a new type of tree automata: leftist tree automata that are designed to recognize our canonical models. Using leftist tree automata we get complexity upper bounds in a clean and simple way. For CTL* over fair programs we prove a doubly exponential time upper bound, and for CTL over fair programs we prove an exponential time upper bound. In both cases we also prove matching lower bounds. Finally, our automata also enable us to prove a finite model theorem for the branching temporal logics of fair programs.

2. Branching Time Temporal Logic

2.1. Syntax

We consider two versions of branching temporal logic: CTL and CTL*. [EH83, EH85]. To define the syntax of these logics, we distinguish between path formulas and state formulas. Path formulas are built from a set Prop of atomic propositions using Boolean and temporal connectives. For example, \( Gp \) means "\( p \) holds at all points of the computation", \( Fp \) means "\( p \) holds at some point of the computation", and \( p \lor q \) means "there is a point in the computation in which \( q \) is true, and \( p \) is true in all preceding points". In CTL path formulas are constructed from a single temporal connective, and in CTL* path formulas are constructed from any number of temporal and Boolean connectives. For example, \( GFp \supset GFq \) is a path formula in CTL* but not in CTL. In both logics, state formulas are essentially Boolean combinations of atomic propositions and existentially or universally quantified path formulas. For example, \( p \lor \exists Gq \) is a CTL state formula; it holds in a state \( s \) if either \( p \) holds in \( s \) or there is a computation that starts at \( s \) and \( q \) holds at all point of that computation. We now give a formal definition of CTL and CTL*.

The CTL and CTL* formulas built from a set of atomic propositions Prop are defined as follows:

State Formulas
- An atomic proposition \( p \in \text{Prop} \) is a state formula.
- If \( f \) and \( g \) are state formulas, so are \( f \lor g \) and \( \neg f \).
- If \( \phi \) is a path formula, then \( \exists \phi \) and \( \forall \phi \) are state formulas.

Path Formulas (CTL)
- If \( \phi \) and \( \psi \) are state formulas, then \( X \phi, F \phi, G \phi \) and \( \phi U \psi \) are path formulas.

Path Formulas (CTL*)
- A state formula is a path formula.
- If \( \phi \) and \( \psi \) are path formulas, so are \( \phi \lor \psi \) and \( \neg \phi \).
- If \( \phi \) and \( \psi \) are path formulas, then \( X \phi, F \phi, G \phi \) and \( \phi U \psi \) are path formulas.

Finally, the formulas of CTL and CTL* are their respective state formulas.

2.2. Semantics

We are interested in interpreting CTL and CTL* over three different types of structures, namely, fair structures, Abrahamson structures (suffix- and fusion-closed structures) and probabilistic structures. In all three cases, the semantics of path formulas will be identical and will be the usual semantics of linear temporal logic formulas. Given a set of states \( S \), an \( \omega \)-path over \( S \) is a function \( \sigma : \omega \rightarrow S \). We denote by \( \sigma^i \) the \( i \)th tail of \( \sigma \) (i.e. \( \sigma^i(j) = \sigma(i + j) \)). We then have the following semantics for path formulas:

- If \( \phi \) is a path formula that is also a state formula, then \( \sigma \models \phi \) iff \( \sigma(0) \models \phi \) according to the semantics of state formulas.
- \( \sigma \models \phi \lor \psi \) iff \( \sigma \models \phi \) and \( \sigma \models \psi \)
- \( \sigma \models \neg \phi \) iff \( \sigma \models \phi \)
- \( \sigma \models X \phi \) iff \( \sigma^i \models \phi \)
- \( \sigma \models F \phi \) iff for some \( i \geq 0 \), \( \sigma^i \models \phi \)

285
• \( \sigma \models G \phi \) iff for all \( i \geq 0 \), \( \sigma^i \models \phi \)

• \( \sigma \models \psi U \psi \) iff for some \( i \geq 0 \), \( \sigma^i \models \psi \) and \( \sigma^j \models \phi \) for all \( 0 \leq j < i \).

The difference between the three semantics we are considering is in how the quantification appearing in state formulas is interpreted. In fair structures, it ranges over fair paths, in Abrahamson structures, it ranges over suffix-closed and fusion-closed sets of paths, and in probabilistic structures it ranges over sets of paths with a given measure. We now give more details about the three types of structures.

### 2.2.1. Fair Structures

A fair structure \( M_F \) is a tuple \((S,R,V,P)\), where \( S \) is a nonempty denumerable set of states, \( R \subseteq S \times S \) is a total accessibility relation on \( S \), \( V : S \rightarrow 2^{2^{V(s)}} \) assigns truth values to the atomic propositions in each state, and \( F \subseteq 2^{V} \times 2^{V} \) is the fairness condition. Thus, \( F \) is a set of pairs \((X_1,Y_1)\) of subsets of \( S \). It is used to define fair paths through \( M_F \).

An \( \omega \)-path through \( M_F \) is a function \( \sigma : \omega \rightarrow S \) such that for all \( i \geq 0 \), \( (\sigma(i),\sigma(i+1)) \in R \). Given a set \( X \subseteq S \), we define the size of the intersection of \( X \) with \( \sigma \) (denoted \( |X \cap \sigma| \)) as the cardinality of the set \( \{ j \in \omega \mid \sigma(j) \in X \} \). A path \( \sigma \) is fair iff, for all pairs \((X_1,Y_1) \in F\), if \( |X_1 \cap \sigma| \) is infinite, then \( |Y_1 \cap \sigma| \) is also infinite. For technical reasons, we will only consider fair structures such that for each state \( s \in S \) there is at least one fair path starting at \( s \).

We have chosen here one of a number of possible definitions of a fairness condition. It corresponds to the notion of generalized fairness in [FK84], and it can be used to express many fairness conditions studied in the literature. For example, consider strong fairness: "if a process is eventually often enabled, then it is scheduled infinitely often" [La80].

To express this condition we let \( F \) contain a pair \((X_i,Y_i)\) for each process \( i \), where \( X_i \) is the set of states where the process is enabled and \( Y_i \) is the set of states where it is scheduled. As another example, consider state fairness: "if a computation goes through a state \( s \) infinitely often, then all transitions from \( s \) are taken infinitely often" [Pm83, Qs82]. To express this condition we let \( F \) contain a pair \((\{s\},\{t\})\) for each edge \((s,t) \in R \). (Interestingly, it follows from the results in the sequel that with regard to satisfiability of \( CTL^* \) formulas it suffices to consider fair structures where the fairness specification consists of a single pair).

We can now give the semantics of \( CTL^* \) and \( CTL^* \) state formulas over fair structures. Given a structure \( M_F \) and a state \( s \in S \), the truth of a state formula is defined as follows:

- \( M_F, s \models \phi \) iff \( \phi \in V(s) \), for an atomic proposition \( \phi \).
- \( M_F, s \models \phi \land \psi \) iff \( M_F, s \models \phi \) and \( M_F, s \models \psi \).
- \( M_F, s \models \neg \phi \) iff \( \neg M_F, s \models \phi \).
- \( M_F, s \models \forall \exists \psi \) iff for all fair paths \( \sigma \) such that \( \sigma(0)=s \), \( \sigma \models \psi \).
- \( M_F, s \models \exists s \) iff there exists a fair path \( \sigma \) such that \( \sigma(0)=s \) and \( \sigma \models g \).

Note that if we ignore the fairness condition and have the quantification range over all paths, rather than just fair paths, we get the usual semantics of \( CTL^* \) and \( CTL^* \) as defined in [EH83, EH85]. For future reference, we will call that semantics the standard semantics of \( CTL^* \).

### 2.2.2. Abrahamson Structures

An Abrahamson structure \( M_A \) is a tuple \((S,V,P)\) where \( S \) is a nonempty denumerable set of states, \( V : S \rightarrow 2^{2^{V(s)}} \) assigns truth values to the atomic propositions in each state, and \( P \subseteq S^\omega \) is a non-empty set of infinite paths that is suffix and fusion closed. A set \( P \) of paths is suffix closed if whenever \( \sigma \in P \), all the suffixes \( \sigma' \) of \( \sigma \) are also in \( P \) \((\sigma, : : \omega \rightarrow S \) is that path defined by \( \sigma'(j) = \sigma(i+j) \))

To define fusion closure, consider any two paths \( \sigma_1 \) and \( \sigma_2 \) in \( P \) that contain a common state \( s \), i.e. \( \sigma_1 \) and \( \sigma_2 \) can be respectively rewritten as \( \sigma_1 = \rho_1 \pi_1 \) and \( \sigma_2 = \rho_2 \pi_2 \). A set \( P \) of paths is then fusion closed if whenever two paths like \( \sigma_1 \) and \( \sigma_2 \) are in \( P \), the paths \( \rho_1 \pi_2 \) and \( \rho_2 \pi_1 \) are also in \( P \). In other words, once we reach a state \( s \) in a path, we can follow any other path from \( s \) and still have a path in the set.

(Interestingly, it was pointed out by Emerson [Em83] that if to the two conditions of suffix closure and fusion closure we add a third condition, limit closure, we get exactly the set of paths that can be generated from a set of states by some binary relation on \( S \). To define limit closure, consider an infinite sequence of paths in \( P \) of the form \( s_1 \pi_1, s_1 s_2 \pi_2, s_1 s_2 s_3 \pi_3 \ldots \). Then \( P \) is limit
closed, if whenever all the paths in the sequence are in $P$, then the limit path $s_1s_2s_3 \cdots$ is also in $P$.

The semantics of CTL and $CTL^*$ over Abrahamson structures are now straightforward. To interpret a formula $\psi$ or $\exists \phi$, in a state $s$, we take the quantifiers as meaning for all (for some) paths in $P$ starting with state $s$.

2.2.3. Probabilistic structures

A probabilistic structure $M_P$ is a tuple $(S, P, V)$ where $S$ is a nonempty denumerable set of states, $P : S \times S \rightarrow [0, 1]$ is a transition probability function such that $\sum_{i \in S} P(s, t) = 1$ for all $s \in S$ and $V : S \rightarrow 2^{P^n}$ assigns truth values to the propositions in each state. One can think of a probabilistic structure as a labeled Markov chain. As in the theory of Markov processes [c.f. KSK66], one can define a measure on the set of all infinite sequences generated by the probabilistic structure, starting with a given state. Specifically, given a state $s_0 \in S$, we define a probability space called the sequence space $\Omega_r = (\Omega, \Delta, \mu)$, where $\omega \subseteq S^\omega$ is the set of all infinite sequences of states starting at $s_0$. $\Delta$ is a Borel field generated by the basic cylindric sets

$$\Delta(s_0, s_1, \ldots, s_n) = \{ \sigma \in \Omega : \sigma = s_0, s_1, \ldots, s_n, \ldots \},$$

and $\mu$ is a probability distribution defined by

$$\mu(\Delta(s_0, s_1, \ldots, s_n)) = P(s_0, s_1) \cdot P(s_1, s_2) \cdots P(s_{n-1}, s_n).$$

Moreover, let $\phi$ be a linear time temporal logic and, for a given $s_0 \in S$, let $\Delta(\phi, s_0)$ be the measure of the set of infinite sequences $\{ \sigma : \sigma(0) = s_0 \land \sigma = \phi \}$. It can be shown that $\Delta(\phi, s_0)$ is a measurable set. This lets us give meaning to CTL and $CTL^*$ over probabilistic structures as follows:

- $M_P, s \models \forall \psi$ iff $\Delta(\psi, s) = 1$.
- $M_P, s \models \exists \psi$ iff $\Delta(\psi, s) > 0$.

Thus $\forall \psi$ means "almost all computation satisfy $\phi$", and $\exists \psi$ means "there is a nonnegligible likelihood that the computation will satisfy $\phi$". See [LS82] for more details.

3. Canonical Structures

In this section, we define a class of canonical structures and show that a $CTL^*$ (and hence CTL) formula has a fair, an Abrahamson, or a probabilistic model iff it has a canonical model. Our canonical structures are a specialization of tree structures. An infinite $n$-ary tree $T$ over an alphabet $\Sigma$ is a mapping $T : n^* \rightarrow \Sigma$, where $|n|$ denotes the set $\{1, \ldots, n\}$. Each element of $|n|^*$ is a node of the tree, and the root is the empty string $\lambda$. We say that a node $y$ is a successor of a node $x$ if $y = xi$ for some $i \in |n|$. A path $\sigma$ starting at a node $z \in |n|^*$ is an infinite set $x_0, x_1, \ldots$ such that $x_0 = z$ and $x_{i+1}$ is a successor of $x_i$ for all $i \geq 0$. In our canonical structures we will use special paths. A leftmost path starting at a node $z \in |n|^*$ is the set $z 1^*$. A leftist path starting at a node $z \in |n|^*$ is an infinite set $x_0, x_1, \ldots$ such that $x_0 = z$, $x_{i+1}$ is a successor of $x_i$ for all $i \geq 0$, and such that for some $j \geq 0$ and for all $i \geq j$, $x_{i+1} = z, 1$. That is, a leftist path is a path that at some point becomes a leftmost path. In other words, after some point our paths always take the leftmost branch out of a node. This is why we call them "leftist".

An $n$-ary canonical structure $M_C$ is a tree over the alphabet $2^{P^n}$. Intuitively, $M_C$ assigns truth values to the propositions in each node. Over a canonical structure, we interpret quantification in CTL and $CTL^*$ formulas as meaning for some (for all) leftist paths. More precisely, for a canonical structure $M_C : |n|^* \rightarrow 2^{P^n}$ and a node $z \in |n|^*$, we have that

- $M_C, x \models \forall \psi$ iff for all leftist paths $\sigma$ starting at $x$, we have $\sigma \models \psi$.
- $M_C, x \models \exists \psi$ iff for some leftist paths $\sigma$ starting at $x$, we have $\sigma \models \psi$.

Because they are characterized by leftist paths, we will often call our canonical structures leftist tree structures or leftist trees. We note that there is nothing special about choosing leftmost branches, what matters is that after some finite prefix, all paths follow a definite direction. Note also that there is a unique leftmost path and countably many leftist path starting at each node, whereas there is an uncountable number of arbitrary paths starting at each node. This is significant to understanding why the leftist interpretation of $CTL^*$ has different characteristics than the standard interpretation.

We can now state our theorems.

**Theorem 8.1:** A $CTL^*$ formula $\phi$ containing $n$ path quantifiers has a fair model iff it has an $n+1$-ary canonical model.
Sketch of Proof:

If: Suppose without loss of generality, that $f$ is satisfied at the node $\lambda$ of an $n+1$-ary canonical tree structure $M_G$. Then, $f$ would be satisfied at the state $\lambda$ of the fair structure $([n+1]^*, M_G, F)$ as long as $F$ ensures that exactly the leftmost paths are fair. This can be done by choosing $F = \{(X, \emptyset)\}$ where $X = \{y \in [n+1]^*: y \notin [n+1]^*1\}$. Basically, $F$ imposes that fair paths do not indefinitely often take transitions that are not leftmost.

Only if: Before giving the proof, we define the state closure of a CTL or $\text{CTL}^*$ formula $f$ $(\text{sel}(f))$ as the set of state subformulas of $f$ and their negation, where we identify $\neg \neg g$ with $g$. We also define existential state formulas as those of the form $\exists \gamma$ or $\forall \gamma$ where $\gamma$ is a path formula.

Now, suppose $f$ is satisfied in a state $x_0$ of a fair structure $M_0 = (S, R, V, F)$. We construct a canonical model for $f$ by the selective filtration technique [Ga76]. We define a leftmost tree structure $M_G([n+1]^* \rightarrow 2^{P_{\text{res}}})$ such that $M_G, \lambda \models f$ by inductively constructing a mapping $\phi: [n+1]^* \rightarrow S$ and taking $M_G(x) = V(\phi(x))$. We start with a mapping $\phi_0$ defined on the set $X_0 = 1^*$ (i.e., the leftmost path starting at $\lambda$). To define $\phi_0$, we choose some fair path $\sigma = \sigma(0)\sigma(1)\sigma(2)\ldots$ starting at $\sigma(0) = x_0$ in $M_0$ and define $\phi(1^k) = \sigma(k)$. Note that each node $x \in X_0$ has exactly one successor in $X_0$ which is its leftmost successor $x_1$.

Now, given the mapping $\phi_0$, defined on the set of nodes $X_0$, we show how to define $\phi_{i+1}$. Our construction ensures that all nodes in $X_i$ either have $n+1$ successors within $X_i$ or only have their leftmost successor in $X_i$. Consider the nodes in $X_i$ that have only a leftmost successor in $X_i$. For each such node $x$, we extend $\phi_i$ as follows. Consider all existential state formulas $E \in \text{sel}(f)$ such that $M_0, \phi(x) \models E$. There are at most $n$ such formulas, say $E_1, \ldots, E_n$, where $n$ is the number of state quantifiers in $f$, where $E_i$ is either $\exists \gamma_j$ or $\forall \gamma_j$. For each formula $E_j$, we choose a fair path $\sigma_j$ in $M$ such that $\sigma_j(0) = \phi_i(x)$ and such that $\sigma_j \models g_i$ if $E_i$ is $\exists \gamma_j$, and $\sigma_j \models \neg g_i$ if $E_i$ is $\forall \gamma_j$. We then define $\phi_{i+1}(x(j+1)) = \sigma_j(1)$ and $\phi_{i+1}(x(j+1)^k) = \sigma_j(k+1)$. If there are only $0 \leq k < n$ formulas $E_j$, for $j = k+1, \ldots, n$, we choose $\sigma_j$ to be an arbitrary fair path starting at $\phi_i(x)$.

Finally, take $\phi$ to be $\bigcup_{i=0}^{\infty} \phi_i$. We leave it to the reader to verify that for all formulas $g \in \text{sel}(f)$ and all nodes in $x \in [n+1]^*$ we have $M_G, x \models g$ if $M_G, \phi(x) \models g$. Thus in particular, $M_G, \lambda \models f$ if $M_0, \phi_0 \models f$.

Theorem 3.2: A $\text{CTL}^*$ formula $f$ containing $n$ path quantifiers has an Asp ammunition model iff it has an $n+1$-ary canonical model.

Sketch of Proof:

If: By definition, a canonical model is an Asp ammunition model.

Only if: Here, we use a construction similar to the one used in the proof of theorem 3.1. The difference is that instead of choosing fair paths when defining $\phi$, we choose paths within the Asp ammunition structure. Both the suffix closure and the fusion closure are necessary to ensure the correctness of the construction. Specifically, suffix closure is necessary to ensure the existence of the relevant paths for the construction to go through. Fusion closure is necessary to ensure that all leftmost paths of the tree structure correspond to paths in the Asp ammunition structure.

Theorem 3.3: A $\text{CTL}^*$ formula $f$ containing $n$ path quantifiers has a probabilistic model iff it has an $n+1$-ary canonical model.

If: If the formula $f$ is satisfied at the node $\lambda$ of the canonical structure $M_G([n+1]^* \rightarrow 2^{P_{\text{res}}})$, then $f$ will be satisfied in the state $\lambda$ of the probabilistic structure $M_0 = ([n+1]^* \rightarrow P, M_G)$ as long as $P$ can be chosen such that 1) all leftmost paths have positive probabilities, 2) at each state the measure of the set of leftmost paths is 1. Note that as we are dealing with a tree structure, $P$ can be viewed as a function $P : [n+1]^* \rightarrow [0, 1]$ rather than as a function $P : [n+1]^* \times [n+1]^* \rightarrow [0, 1]$. For a state $x \in [n+1]^*$, $x \neq \lambda$, $P(x)$ will be the probability of the unique edge leading to $x$. By convention, we will take $P(\lambda) = 1$.

We define the function $P$ inductively, similarly to what we did for the function $\phi$ in the proof of Theorem 3.1. We start by defining $P_0$ on the set $X_0 = 1^*$ by $P_0(1^k) = 1 - \alpha_k$ for some constants $\alpha_0, \alpha_1, \ldots$ such that $\alpha_0 = 0, 0 < \alpha_k < 1$ for all $k > 0$, and $\sum_{k>0} \alpha_k < 1$. Note that since $\prod_{k>0} (1 - \alpha_k) \geq 1 - \sum_{k>0} \alpha_k$, we have ensured that the
measure of the path \( 1^* \) is positive.

Now, given the function \( P_i \) defined on \( X_i \), we define \( P_{i+1} \). For each node \( x \in X_i \) that only has a leftmost successor in \( X_i \) we proceed as follows. For \( j \in \{2, \ldots, n+1\} \), we define \( P(xj) = (1-P(x1))/n \). Moreover, for \( k > 0 \), we choose \( P(xj1^k) = 1-\alpha_k \). It is easy to convince oneself that this assignment of probabilities makes the measure of leftist paths positive. Let us prove that it also makes the measure of the set of leftist paths starting at a given node be 1. We denote by \( \mu_{NL}(x) \) the measure of the set of non-leftist paths rooted at \( x \). Notice that for all states \( y \) that are not of the form \( x1 \), the assignment of probabilities to the subtree rooted at \( y \) is identical to the assignment of probabilities to the tree rooted at \( \lambda \) and hence that \( \mu_{NL}(y) = \mu_{NL}(\lambda) \).

For any node \( y \neq x1 \) we thus have
\[
\mu_{NL}(y) \leq (\sum_{i \geq 0} \alpha_{i+1}) \times \mu_{NL}(x1) \quad \text{which implies that} \quad \mu_{NL}(y) = 0.
\]

For a node \( y \) that is of the form \( x1 \), we have that for some \( j > 0 \) \( \mu_{NL}(y) \leq (\sum_{i \geq 0} \alpha_{i+1}) \mu_{NL}(\lambda) \) and hence we also have \( \mu_{NL}(y) = 0 \).

Only if. Again the construction is similar to the one given for Theorem 3.1. Here, when selecting the paths satisfying existential state formulas, we have to select a path that satisfies the existential formula and also all universal state formulas that are true at the point \( \phi(x) \).

We can always find such a path as universal formulas are satisfied on sets of paths of measure 1 and existential formulas on sets of paths of positive measure.

Theorems 3.1, 3.2, and 3.2, show that fair structures, Abrahamson structures, and probabilistic structures have the same branching temporal logic. The meaning of this result is that BTL is not expressive enough to distinguish between these very different kinds of structures, in the same way that first-order logic may not distinguish between two elementarily equivalent but non-isomorphic relational structures.

4. Decision Procedures

In this section, we investigate the satisfiability problem for \( \text{CTL} \) and \( \text{CTL}^* \) interpreted over canonical structures. By Theorems 3.1-3.3, our results will also apply to fair, Abrahamson and probabilistic structures.

4.1. Leftist Tree Automata

To obtain a decision procedure for \( \text{CTL} \) or \( \text{CTL}^* \) interpreted over fair structures, we use the approach advocated in [VW84]. Specifically, given a formula \( \phi \), we build a tree automaton \( A_\phi \) that recognizes precisely the leftist tree models of \( \phi \). We then check whether \( A_\phi \) accepts some tree. \( \phi \) is satisfiable if and only if \( A_\phi \) accepts some tree. Unlike the tree automata in [VW84], we need here automata that run on leftist trees; we call them leftist tree automata.

Let us first recall some standard definitions. A Buchi sequential automaton \( A = (\Sigma, S, \rho, S_0, F) \), where \( \Sigma \) is the alphabet, \( S \) is a set of states, \( \rho : S \times \Sigma \to 2^S \) is the transition function, \( S_0 \subseteq S \) is the set of initial states, and \( F \subseteq S \) is a set of designated states. A run of an automaton \( A \) over an infinite word \( w : \omega \to \Sigma \) is an infinite word \( \sigma : \omega \to S \) such that \( \sigma(0) \in S_0 \) and for all \( i > 0 \), \( \sigma(i) \in \rho(\sigma(i-1), w(i-1)) \). A run of \( \sigma \) is accepting if \( \inf \{ \sigma \} \cap F \neq \emptyset \), where \( \inf \{ \sigma \} \) is the set of states that occur infinitely often in \( \sigma \). That is, accepting runs are those in which some state in \( F \) appears infinitely often.

A Buchi n-ary tree automaton is a tuple \( A = (\Sigma, S, \rho, S_0, F) \), where \( \Sigma \) is the alphabet, \( S \) is a set of states, \( \rho : S \times \Sigma \to 2^S \) is the transition function (for each state \( s \) and letter \( \sigma_i \) it gives the possible sets of \( n \) successors), \( S_0 \subseteq S \) is the set of initial states, and \( F \subseteq S \) is a set of designated states. A run of an automaton \( A \) over a tree \( T \vdash [n]^* \to \Sigma \) is a tree \( \phi : [n]^* \to S \), where \( \phi(\lambda) \in S_0 \) and for every \( x \in [n]^* \), we have \( (\phi(x1), \ldots, \phi(xn)) \in \rho(\phi(x), T(x)) \). A run \( \phi \) of \( A \) over \( T \) is accepting if for all infinite paths \( \sigma \) starting at \( \lambda \) we have \( \inf \{ \phi(\sigma) \} \cap F \neq \emptyset \), where \( \inf \{ \phi(\sigma) \} \) denotes the set of states appearing infinitely often on path \( \sigma \) in run \( \phi \). A accepts \( T \) if it has an accepting run on \( T \). \( A \) is non-empty if \( A \) accepts some tree.

Leftist tree automata are like Buchi automata except for the notion of acceptance. Instead of defining an accepting run as one in which on all infinite paths \( \sigma \), \( \inf \{ \phi(\sigma) \} \cap F \neq \emptyset \), we only require that on all leftist paths \( \sigma_1 \), the condition \( \inf \{ \phi(\sigma_1) \} \cap F \neq \emptyset \) is satisfied.
As mentioned before, in order to use tree automata for deciding satisfiability of formulas, we have to be able to solve their emptiness problem, i.e., to decide, for a given automaton \( A \), whether \( A \) accepts some tree. Rabin has shown that the emptiness problem for Büchi tree automata is solvable in polynomial time [Ra70]. Here we prove a similar result for leftist tree automata.

**Theorem 4.1:** The emptiness problem for leftist automata is logspace complete for PTIME.

**Sketch of Proof:**

In PTIME: Consider an \( n \)-ary leftist tree automaton \( A = (\Sigma, S, \rho, \delta_0, F) \). Since we deal with nondeterministic automata, we can assume that the alphabet \( \Sigma \) consists of a single letter \( a \). For a state \( s \in S \), a finite leftmost path \( \sigma_{lm} \) from \( s \) in \( A \) is a finite sequence of states \( s_0, s_1, \ldots, s_l \) such that \( s_0 = s \) and for all \( 1 \leq i \leq k \), there exists arbitrary states \( s_{i,0}, s_{i,1}, \ldots, s_{i,n} \) such that \( (s_i, s_{i,0}, \ldots, s_{i,n}) \in \rho(s_{i-1}, a) \). We will say that a state \( s \) is a *good leftist state* if there is a finite leftmost path \( \sigma_{lm} \) from \( s \) of length \( k \geq 2 \) such that its last state, \( s_k \), is in \( F \).

The algorithm proceeds by repeatedly eliminating the states that are not good leftist states. The algorithm stops when no more states can be eliminated. Note that after a state is eliminated, the transition function \( \rho \) and the set \( F \) have to be updated accordingly. Clearly this algorithm runs in polynomial time. We leave it to the reader to verify that the leftist automaton accepts some tree iff some initial state is not eliminated.

**Hard for PTIME:** This can be shown by reduction from the path system problem in [IL77].

### 4.2. CTL^*

We first give the decision procedure for CTL^* as it is the most general and interesting. We then specialize it to CTL. The best decision procedure known for CTL^* over standard structures is nondeterministic doubly exponential time [Em85a, VS85]. The reason for the difficulty can be intuitively explained as follows. What makes CTL^* hard to decide over standard structures is that to check a formula of the form \( \forall f \), one has to check that \( f \) is satisfied on all infinite paths. The usual way to do this is to build a Büchi sequential automaton (on infinite words) that accepts the paths satisfying \( f \) [WVS83], and to run this automaton on all paths. Unfortunately, to run this automaton on all paths, it has to be deterministic, and determining finite automata on infinite strings is notably hard. In particular, to determinize Büchi sequential automata one needs to use a more powerful kind of sequential automata, such as Rabin automata [Ra72] or Streett automata [St82] (cf. [ES84, McN66]). Thus the resulting tree automata are not Büchi tree automata, and their emptiness problem is harder [Em85a, St82, VS85].

The situation is rather different when we interpret CTL^* over canonical structures. Here, we are interested in accepting leftist trees. This gives us the advantage that, to check a formula \( \forall f \), we only have to check \( f \) on leftist paths. This can be done without determining the sequential automata. Applying the classical subset construction suffices. (Note the the subset construction does not determinize automata on infinite words.)

Before giving the construction, we define the notion of a leftist Hintikka tree for a CTL^* formula \( f \). Note that, as our logic includes negation for both state and path formulas, we can assume without loss of generality that the only path quantifier being used is \( \exists \). To define Hintikka trees, we need to define the semantics of path formulas over sequences \( \sigma : \omega \rightarrow 2^{s(F)} \). These semantics are identical to those we gave in section 2, except for the following clause: if \( \phi \) is a state-formula in \( s(F) \), then we say that \( \sigma \models \phi \) if \( \phi \in \sigma(0) \). A leftist Hintikka tree for a CTL^* formula \( f \), of size \( n \), with state closure \( s(F) \), is an \( n+1 \) ary tree \( T : [n+1]^* \rightarrow 2^{s(F)} \) that satisfies the following conditions for all elements \( x \) of \([n+1]^*\):

1. \( f \in T(\lambda) \).
2. \( g \in T(x) \) iff \( \neg g \notin T(x) \).
3. \( g_{i,1} / \bar{g} \in T(x) \) iff \( g_{i,1} \in T(x) \) and \( g \notin T(x) \).

Let \( E_1, \ldots, E_n \) be the formulas of the form \( \exists y^i \) in \( s(F) \).

4. If \( E_i \in T(x) \), then the path \( x \mid 1^* \) satisfies the path formula \( g \).
5. If \( \neg \exists y \in T(x) \), then all the leftist paths from \( x \) satisfy the path formula \( \neg g \).

**Lemma 4.2:** A CTL^* formula \( f \) is satisfiable by a canonical structure iff there is a leftist Hintikka tree for
Sketch of proof:

If: given a Hintikka tree $T$, one can obtain a canonical model by restricting $T$ to $[n+1]^* \rightarrow \mathcal{P}^n$.

Only if: given a canonical model $M_G$, one can build a leftist Hintikka tree by first labeling each node $x$ of $M_G$ by the formulas $\varphi \in \text{scf}(f)$ such that $M_G, x \models \varphi$. One can then ensure that each existential formula $E_i = \exists y$ is satisfied on the path $x_i^*$ by a construction similar to the one used in the proof of Theorem 3.1.

Theorem 4.8: Given a $\text{CTL}^*$ formula $f$ of length $n$, one can build an $n+1$-ary leftist automaton of size $O(\exp^n(n))$ that is non-empty iff $f$ is satisfiable.

Sketch of Proof: We can build an automaton that recognizes the leftist Hintikka trees for $f$. The automaton is built in three parts. The local automaton, checks Hintikka conditions 1-3, the existential automaton checks Hintikka condition 4, and the universal automaton checks Hintikka condition 5. We now describe each of these automata.

The Local Automaton

The local automaton is $A_L = \{2^{\text{scf}(f)}, \cdot \cdot \cdot , \rho_L, N_f, 2^{\text{scf}(f)}\}$. The state set and the alphabet are thus the collection of all sets of formulas in $\text{scf}(f)$. For the transition relation we have that $(s_0, \ldots, s_{n+1}) \in \rho_L(s, a)$ if $a = s$ and

- $g \in s$ iff $-g \notin s$.
- $g \land g \notin s$ iff $g \in s$ and $g \notin s$.

The set of starting states $N_f$ consists of all sets $s$ such that $f \in s$.

The Existential Automaton

The existential automaton checks Hintikka condition 4. To do this, we construct an automaton for each of the formulas $E_i$, and then take the intersection of these automata. The construction of the automaton $A_{E_i}$ for a specific $E_i = \exists y$ proceeds as follows. By the result in [WVS83], we can build a Büchi sequential automaton $A_y = \{2^{\text{scf}(f)}, S, \varphi, S_0, F\}$ that accepts exactly the sequences satisfying $y$. $A_{E_i}$ is then $\{2^{\text{scf}(f)}, S_f, \cup(0), \rho_E, S_0, F\}$. We have that $(s_0, \ldots, s_{n+1}) \in \rho_E(s, a)$ whenever the following conditions are satisfied:

- if $E_i \notin s$, then, for all $2 \leq j \leq n+1$, $s_j = 0$.
- if $E_i \in s$, then $s_{n+1} = s'$ for some $s'$ such that $s' \in \rho_f(s_{n+1}, a)$ for some $a \in S_0$. Also $s_j = 0$ for all $2 \leq j \neq i + 1 \leq n + 1$.
- if $s = 0$, then $s_1 = 0$.
- if $s \neq 0$, then $s_1 = s'$ for some $s'$ such that $s' \in \rho_f(s, a)$.

The existential automaton $A_E$ is then the intersection (c.f. [VW84]) of the at most $n$ automata $A_{E_i}$. Each of these is of size $2^n$, where $n$ is the length of the formula path formula in $E_i$. Hence the automaton $A_E$ is of size $2^{2^n} = 2^n$.

The Universal Automaton

Here we build an automaton for each of the formulas $U_i$ of the form $\neg \exists y$ that are in $\text{scf}(f)$. Each of these automata checks that whenever the formula, $\neg \exists y \in T[x]$, there is no leftist path from $x$ that satisfies $\varphi$, or equivalently that all leftist paths from $x$ satisfy $\neg \varphi$. To do this, we could construct the Büchi sequential automaton for the formula $\neg \varphi$ and run it over all leftist paths starting at every node $x$ where $\neg \exists y \in x$. However, it is more economical and equivalent to proceed as follows. We run on all leftist paths from the root the sequential Büchi automaton (over the alphabet $2^{\text{scf}(f)}$) corresponding to the linear time temporal logic formula $G(\neg \exists y \supset \neg \varphi)$. Let this automaton be $A_{U_i} = \{2^{\text{scf}(f)}, S, \varphi, S_0, F\}$. The automaton on leftist trees for the formula $U_i = \neg \exists y$ is then $A_{U_i} = \{2^{\text{scf}(f)}, S_f \times S_0, \rho_{U_i}, (S_0, x) \times S_0, F\}$. We have that $(s_0, \ldots, s_{n+1}) \in \rho_{U_i}(s, a)$ whenever the following conditions are satisfied, for $s = (X, s')$:

- $s' = (X, s')$ where $s' \in \rho_f(s, a)$ and $Y = \{y \in S_f \mid y \in \rho_f(x, a), \text{ for some } x \in X\}$.
- For $2 \leq j \leq n+1, s_j = (Y, s')$ where $s' \in \rho_f(x, a)$ for some $x \in X$, and $Y = \{y \in S_f \mid y \in \rho_f(x, a), \text{ for some } x \in X\}$.

Intuitively, what our construction does is to apply the classical Rabin-Scott [RS59] construction to the automaton $A_f$ and run the resulting deterministic automaton down the tree, while simultaneously running a nondeterministic copy of $A_f$ down every leftmost path (i.e. path of the form $1^*$) that is encountered. The
universal automaton $A_U$ is then the intersection of the (at most) $n$ automata $A_U^i$. Each of these automata is of size $2^{2^n}$, where $n_i$ is the length of the path formula in $A_U^i$, and hence the automaton $A_U$ is at most of size $2^{2^n}$.

Finally, we take the intersection of the local automaton, the existential automaton and the universal automaton. The resulting automaton is of size $O(\exp(n))$.]

The lower bound proved in [VS85] for $CTL^*$ interpreted over standard structures can be adapted to $CTL^*$ interpreted over canonical structures. Thus we have:

**Theorem 4.4:** The satisfiability problem for $CTL^*$ interpreted over canonical (fair, Abrahamson, probabilistic) structures is logspace complete for deterministic doubly exponential time.]

Note that the above upper bound also follows from Theorems 3.1-3.3 and the doubly exponential time upper bounds in [Ab80,Kr84]. Nevertheless, our use of leftist automata yields a decision procedure that is substantially cleaner than Abrahamson's and Kraus'. Also, we find it very illuminating as far as explaining the difference between $CTL^*$ interpreted over fair and standard structures. We also note that our lower bound applies to the other logics in [LS82] (i.e., $CTL^*$ interpreted over finite and bounded infinite Markov chains).

### 4.3. $CTL$

For $CTL$, we can of course use the same construction as for $CTL^*$. The main difference is that the Büchi sequential automata we build for path formulas are this time of constant size rather than exponential in the length of the formula. This implies that the resulting automaton is of size $O(\exp(n))$. Precisely, we have the following:

**Theorem 4.5:** Given a $CTL$ formula $f$ of length $n$, one can build an $n+1$-ary leftist automaton of size $O(\exp(n))$ that is non-empty iff $f$ is satisfiable.

Combining Theorems 4.1 and 4.4 yields an upper bound for the satisfiability of $CTL$ over canonical structures. An exponential time lower bound for $CTL$ over probabilistic structure was proven in [Kr84].

**Theorem 4.6:** The satisfiability problem for $CTL$ over canonical (fair, Abrahamson, probabilistic) structures is logspace complete for EXPTIME.]

An exponential upper bound for $CTL$ over probabilistic structure is also proven in [Kr84]. Our proof, however, has the advantage that it is easy to extend to logics between $CTL$ and $CTL^*$. The key idea behind our exponential time upper bound is that the sequential automata for the path formulas in $CTL$ are of some fixed size. As long as this property holds, the exponential time upper bounds also holds. For example, we could allow formulas of the form $\exists(xUy)x\neq x$, where $p$, $q$, and $r$, are state formulas, and still retain the exponential time upper bound.

### 4.4. Finite Models

Another advantage of using decision procedures based on automata, is that they make it relatively easy to prove a finite model theorem for $CTL$ and $CTL^*$ interpreted over fair structures.

**Theorem 4.8:** A $CTL^*$ formula $f$ has a finite fair model if and only if it has a finite model.

The idea is that, if a leftist automaton accepts some infinite leftist tree, then there is a representation of that tree by a finite fair structure. The finite representation will be of size polynomial in the size of the automaton and will thus be exponential in the size of the formula for $CTL$ and double exponential in the size of the formula for $CTL^*$. (The theorem does not apply to probabilistic structures, since we need infinitely many different transition probabilities). The construction of a finite model from the given formula can also be viewed as a synthesis of a fair program from the correctness specification (cf. [EC82, MW84]).

### 5. Concluding Remarks

We have investigated the branching temporal logic of fair concurrent programs. The motivation for doing so is that we view the fairness conditions as an intrinsic part of the program, separate from the correctness assertions. We have shown that the branching time logic (BTL) of fair structures is identical to the BTL of Abrahamson and probabilistic structures. We believe this is a significant contribution to our understanding of the BTL of all three types of structures. Our proof is based on the existence of canonical models (which are leftist tree structures) for the three different logics. In order to
obtain decision procedures, we developed an automata theory for leftist trees. Then, using leftist automata, we obtained clean decision procedures, and provided matching lower bounds.

References


[LP85] O. Lichtenstein, A. Pnueli, "Checking that Finite-State Concurrent Programs Satisfy


