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, LPS81]).) 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) [GPSS80, 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.

[*] Address: 600 Mountain ave, Murray Hill, NJ 07974
[†] Address: Department K55/801, 650 Harry Road, San Jose, CA 95120-6099

In BTL, formulas are built from quantifiers over paths of the branching structure and formulas describing properties of the quantified paths. A typical BTL formulas is of the form $\exists\phi$, where ϕ is an LTL formula. $\exists\phi$ asserts that there exists a computations that satisfies ϕ. 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 then 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 most 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_g) of probabilistic programs [LS82]. Probabilistic programs can be viewed as implementing fairness through randomness. Lehman and Shelah supplied an axiomatization for TC_g, and Kraus a doubly exponential upper bound [Kr84] (see also [HS84,KL83]). Surprisingly, the axiomatization of TC_g 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 iff 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 pUq 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,
$p \vee \exists Gq$ 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 $p \in Prop$ is a state formula.
- If f and g are state formulas, so are $f \wedge g$ and $\neg f$.
- If ϕ is a path formula, then $\exists\phi$ and $\forall\phi$ are state formulas

Path Formulas (CTL)

- If ϕ and ψ 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 ϕ and ψ are path formulas, so are $\phi \wedge \psi$ and $\neg\phi$.
- If ϕ and ψ 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 ω-path over S is a function $\sigma:\omega \rightarrow S$. We denote by σ^i the i th tail of σ (i.e. $\sigma^i(j)=\sigma(i+j)$). We then have the following semantics for path formulas:

- If ϕ 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.
- $\sigma \models \phi \wedge \psi$ iff $\sigma \models \phi$ and $\sigma \models \psi$
- $\sigma \models \neg\phi$ iff not $\sigma \models \phi$
- $\sigma \models X\phi$ iff $\sigma^1 \models \phi$
- $\sigma \models F\phi$ iff for some $i \geq 0$, $\sigma^i \models \phi$

- $\sigma \models G\,\phi$ iff for all $i \geq 0$, $\sigma^i \models \phi$
- $\sigma \models \phi U \psi$ 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 $(S, R, V, \mathbf{F})$, where S is a nonempty denumerable set of states, $R \subseteq S \times S$ is a total accessibility relation on S, $V: S \rightarrow 2^{Prop}$ assigns truth values to the atomic propositions in each state, and $\mathbf{F} \subseteq 2^S \times 2^S$ is the *fairness condition*. Thus, $\mathbf{F}$ is a set of pairs (X_i, Y_i) of subsets of S. It is used to define fair paths through M_F.

An ω-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 σ (denoted $|X \cap \sigma|$) as the cardinality of the set $\{j \in \omega \mid \sigma(j) \in X\}$. A path σ is *fair* iff, for all pairs $(X_i, Y_i) \in \mathbf{F}$, if $|X_i \cap \sigma|$ is infinite, then $|Y_i \cap \sigma|$ 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 definition 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 infinitely often enabled, then it is scheduled infinitely often" [La80]. To express this condition we let $\mathbf{F}$ 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" [Pn83,QS82]. To express this condition we let $\mathbf{F}$ 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:

- $M_F, s \models p$ iff $p \in V(s)$, for an atomic proposition p.
- $M_F, s \models \phi \wedge \psi$ iff $M_F, s \models \phi$ and $M_F, s \models \psi$
- $M_F, s \models \neg\phi$ iff not $M_F, s \models \phi$
- $M_F, s \models \forall g$ iff for all fair paths σ such that $\sigma(0)=s$, $\sigma \models g$.
- $M_F, s \models \exists g$ iff there exists a fair path σ 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 and CTL^*.

2.2.2. Abrahamson Structures

An Abrahamson structure M_A is a tuple $(S, V, \mathbf{P})$ where S is a nonempty denumerable set of states, $V: S \rightarrow 2^{Prop}$ assigns truth values to the atomic propositions in each state, and $\mathbf{P} \subseteq S^\omega$ is a non-empty set of infinite paths that is suffix and fusion closed. A set $\mathbf{P}$ of paths is suffix closed if whenever $\sigma \in \mathbf{P}$, all the suffixes σ^i of σ are also in $\mathbf{P}$ ($\sigma^i: \omega \rightarrow S$ is that path defined by $\sigma^i(j) = \sigma(i+j)$). To define fusion closure, consider any two paths σ_1 and σ_2 in $\mathbf{P}$ that contain a common state s, i.e. σ_1 and σ_2 can be respectively written as $\sigma_1 = \rho_1 s\, \pi_1$ and $\sigma_2 = \rho_2 s\, \pi_2$. A set $\mathbf{P}$ of paths is then fusion closed if whenever two paths like σ_1 and σ_2 are in $\mathbf{P}$, the paths $\rho_1 s\, \pi_2$ and $\rho_2 s\, \pi_1$ are also in $\mathbf{P}$. 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 $\mathbf{P}$ of the form $s_1\pi_1$, $s_1 s_2 \pi_2$, $s_1 s_2 s_3 \pi_3$ .... Then $\mathbf{P}$ is limit closed, if whenever all the paths in the sequence are in **P**, then the limit path $s_1 s_2 s_3 \cdots$ is also in **P**.)

The semantics of CTL and CTL^* over Abrahamson structures are now straightforward. To interpret a formula $\forall\phi$ 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 $\sum_{t \in S} P(s,t) = 1$ for all $s \in S$ and $V : S \rightarrow 2^{Prop}$ 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 sets of of 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*** $\Psi_\Pi = (\Omega, \Delta, \mu)$, where $\Omega \subseteq S^\omega$ is the set of all infinite sequences of states starting at s_0, Δ is a Borel field generated by the ***basic cylindric sets***

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

and μ is a probability distribution defined by

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

Moreover, let ϕ 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 $\{\sigma \mid \sigma(0) = s_0 \wedge \sigma \models \phi\}$. 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:

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

Thus $\forall\phi$ means "almost all computation satisfy ϕ", and $\exists\phi$ means "there is a nonnegligible likelihood that the computation will satisfy ϕ". 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 Σ 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 λ. We say that a node y is a successor of a node x if $y = xi$ for some $i \in [n]$. A ***path*** σ starting at a node $x \in [n]^*$ is an infinite set $x_0, x_1, \ldots$ such that $x_0 = x$ 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 $x \in [n]^*$ is the set $x\,1^*$. A *leftist* path starting at a node $x \in [n]^*$ is an infinite set $x_0, x_1, \ldots$ such that $x_0 = x$, 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 $i \geq j$, $x_{i+1} = x_i 1$. 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^{Prop}. 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 $M_C : [n]^* \rightarrow 2^{Prop}$ and a node $x \in [n]^*$, we have that

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

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 3.1: A CTL^* formula f 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 λ of an $n+1$-ary canonical tree structure M_C. Then, f would be satisfied at the state λ of the fair structure $([n+1]^*, M_C, \mathbf{F})$ as long as $\mathbf{F}$ ensures that exactly the leftist paths are fair. This can be done by choosing $\mathbf{F}=\{(X,\emptyset)\}$ where $X=\{y\in[n+1]^* : y\notin[n+1]^*1\}$. Basically, $\mathbf{F}$ imposes that fair paths do not infinitely often take transitions that are not leftist.

Only if: Before giving the proof, we define the state closure of a CTL or CTL^* formula f $(scl(f))$ 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 $\exists g$ or $\neg\forall g$ where g is a path formula.

Now, suppose f is satisfied in a state s_0 of a fair structure $M_F=(S,R,V,\mathbf{F})$. We construct a canonical model for f by the ***selective filtration*** technique [Ga76]. We define a leftist tree structure $M_C:[n+1]^*\rightarrow 2^{Prop}$ such that $M_C,\lambda\models f$ by inductively constructing a mapping $\phi:[n+1]^*\rightarrow S$ and taking $M_C(x)=V(\phi(x))$. We start with a mapping ϕ_0 defined on the set $X_0=1^*$ (i.e. the leftmost path starting at λ). To define ϕ_0, we choose some fair path $\sigma=\sigma(0)\sigma(1)\sigma(2)\ldots$ starting at $\sigma(0)=s_0$ in M_F and define $\phi_0(1^k)=\sigma(k)$. 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 ϕ_i defined on the set of nodes X_i, we show how to define ϕ_{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 only have a leftmost successor in X_i. For each such node x, we extend ϕ_i as follows. Consider all existential state formulas $E\in scl(f)$ such that $M_F,\phi(x)\models E$. 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_j is either $\exists g_j$ or $\neg\forall g_j$. For each formula E_j, we choose a fair path σ_j in M_F such that $\sigma_j(0)=\phi_j(x)$ and such that $\sigma_j\models g_j$ if E_j is $\exists g_j$ and $\sigma_j\models\neg g_j$ if E_j is $\neg\forall g_j$. We then define $\phi_{i+1}(x(j+1))=\sigma_j(1)$ and $\phi_{i+1}(x(j+1)1^k)=\sigma_j(k+1)$. If there are only $0\leq k<n$ formulas E_j, for $j=k+1,\ldots,n$, we choose σ_j to be an arbitrary fair path starting at $\phi_i(x)$.

Finally, take ϕ to be $\bigcup_{i=1}^{\infty}\phi_i$. We leave it to the reader to verify that for all formulas $g\in scl(f)$ and all nodes in $x\in[n+1]^*$ we have $M_C,x\models g$ iff $M_F,\phi(x)\models g$. Thus in particular, $M_C,\lambda\models f$ iff $M_F,s_0\models f$. []

Theorem 3.2: A CTL^* formula f containing n path quantifiers has an Abrahamson model iff it has an $n+1$-ary canonical model.

Sketch of Proof:

If: By definition, a canonical model is an Abrahamson 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 ϕ, we choose paths within the Abrahamson 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 leftist paths of the tree structure correspond to paths in the Abrahamson structure. []

Theorem 3.3: A 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 λ of the canonical structure $M_C:[n+1]^*\rightarrow 2^{Prop}$, then f will be satisfied in the state λ of the probabilistic structure $M_P=([n+1]^*,P,M_C)$ as long as P can be chosen such that 1) all leftist paths have positive probabilities, 2) at each state the measure of the set of leftist 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 ϕ 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 in $x \in X_i$ 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 λ and hence that $\mu_{NL}(y)=\mu_{NL}(\lambda)$. For any node $y \neq x1$, we thus have $\mu_{NL}(y) \leq (\sum_{k \geq 0} \alpha_{k+1}) \times \mu_{NL}(y)$ which implies that $\mu_{NL}(y)=0$. For a node y that is of the form $x1$, we have that for some $j>0$ $\mu_{NL}(y) \leq (\sum_{k \geq j} \alpha_{k+1}) \mu_{NL}(\lambda)$ 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 *CTL* and *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 *CTL* or *CTL** interpreted over fair structures, we use the approach advocated in [VW84]. Specifically, given a formula ϕ, we build a tree automaton A_ϕ that recognizes precisely the leftist tree models of ϕ. We then check whether A_ϕ accepts ***some*** tree. ϕ is satisfiable if and only if A_ϕ 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 ***Büchi sequential automaton*** A is a tuple $A=(\Sigma, S, \rho, S_0, F)$, where Σ is the alphabet, S is a set of states, $\rho: S \times \Sigma \rightarrow 2^S$ 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 $w: \omega \rightarrow \Sigma$ is an infinite word $\sigma: \omega \rightarrow S$ 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 σ is ***accepting*** if $inf(\sigma) \cap F \neq \emptyset$, where $inf(\sigma)$ is the set of states that occur infinitely often in σ. That is, accepting runs are those in which some state in F appears infinitely often.

A ***Büchi n-ary tree automaton*** is a tuple $A=(\Sigma, S, \rho, S_0, F)$, where Σ is the alphabet, S is a set of states, $\rho: S \times \Sigma \rightarrow 2^{S^n}$ is the transition function (for each state and letter 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 $T:[n]^* \rightarrow \Sigma$ is a tree $\phi:[n]^* \rightarrow S$, 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 ϕ of A over T is ***accepting*** if for all infinite paths σ starting at λ we have $inf(\phi,\sigma) \cap F \neq \emptyset$, where $inf(\phi,\sigma)$ denotes the set of states appearing infinitely often on path σ in run ϕ. 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 Büchi automata except for the notion of acceptance. Instead of defining an accepting run as one in which on ***all*** infinite paths σ, $inf(\phi,\sigma) \cap F \neq \emptyset$, we only require that on all ***leftist*** paths σ_l, the condition $inf(\phi,\sigma_l) \cap F \neq \emptyset$ 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 $A=(\Sigma,S,\rho,S_0,F)$. Since we deal with nondeterministic automata, we can assume that the alphabet Σ consists of a single letter a. For a state $s\in S$, a finite leftmost path σ_{lm} from s in A is a finite sequence of states $s_0,s_1,\ldots,s_k$ such that $s_0=s$ and for all $1\leq i\leq k$, there exists arbitrary states $x_{i,2},\ldots,x_{i,n}$ such that $(s_i,x_{i,2},\ldots,x_{i,n})\in\rho(s_{i-1},a)$. We will say that a state s is a ***good leftist state*** if there is a finite leftmost path σ_{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 ρ and the set F have bo 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 [JL77]. []

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 know 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 determinizing 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 determinizing 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 $\sigma:\omega\rightarrow 2^{scl(f)}$. These semantics are identical to those we gave in section 2, except for the following clause: if ϕ is a state-formula in $scl(f)$, 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 $scl(f)$, is an $n+1$ ary tree $T:[n+1]^*\rightarrow 2^{scl}(f)$ 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) $g_1\wedge g_2\in T(x)$ iff $g_1\in T(x)$ and $g_2\in T(x)$.

Let $E_1,\ldots,E_n$ be the formulas of the form $\exists g_i$ in $scl(f)$.

4) If $E_i\in T(x)$, where $E_i=\exists g$, then the path $xi1^*$ satisfies the path formula g.

5) If $\neg\exists g\in T(x)$, 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

f .

Sketch of proof:

If: given a Hintikka tree T, one can obtain a canonical model by restricting T to $[n+1]^* \rightarrow 2^{Prop}$.

Only if: given a canonical model M_C, one can build a leftist Hintikka tree by first labeling each node x of M_C by the formulas $g \in scl(f)$ such that $M_C, x \models g$. One can then ensure that each existential formula $E_i = \exists g$ is satisfied on the path $xi1^*$ by a construction similar to the one used in the proof of Theorem 3.1. []

Theorem 4.3: Given a CTL^* formula f of length n, one can build an $n+1$-ary leftist automaton of size $O(\exp^2(n))$ that is non-empty iff f is satisfiable.

Sketch of Proof: We 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 $A_L = (2^{scl(f)}, 2^{scl(f)}, \rho_L, N_f, 2^{scl(f)})$. The state set and the alphabet are thus the collection of all sets of formulas in $scl(f)$. For the transition relation we have that $(\mathbf{s}_1, \ldots, \mathbf{s}_{n+1}) \in \rho_L(\mathbf{s}, \mathbf{a})$ iff $\mathbf{a} = \mathbf{s}$ and

- $g \in \mathbf{s}$ iff $\neg g \notin \mathbf{s}$.
- $g_1 \wedge g_2 \in \mathbf{s}$ iff $g_1 \in \mathbf{s}$ and $g_2 \in \mathbf{s}$.

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

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 $E_i = \exists g$ proceeds as follows. By the result in [WVS83], we can build a Büchi sequential automaton $A_g = (2^{scl(f)}, S_g, \rho_g, S_{0g}, F_g)$ that accepts exactly the sequences satisfying g. A_{E_i} is then $(2^{scl(f)}, S_g \cup \{0\}, \rho_{E_i}, S_{0g}, F_g)$. We have that $(\mathbf{s}_1, \ldots, \mathbf{s}_{n+1}) \in \rho_{E_i}(\mathbf{s}, \mathbf{a})$ whenever the following conditions are satisfied:

- if $E_i \notin \mathbf{a}$, then, for all $2 \leq j \leq n+1$, $\mathbf{s}_j = 0$.
- if $E_i \in \mathbf{a}$, then $\mathbf{s}_{i+1} = \mathbf{s}'$ for some $\mathbf{s}'$ such that $\mathbf{s}' \in \rho_g(\mathbf{s}_{0g}, \mathbf{a})$ for some $\mathbf{s}_{0g} \in S_{0g}$. Also $\mathbf{s}_j = 0$ for all $2 \leq j \neq i+1 \leq n+1$.
- if $\mathbf{s} = 0$, then $\mathbf{s}_1 = 0$.
- if $\mathbf{s} \neq 0$, then $\mathbf{s}_1 = \mathbf{s}'$ for some $\mathbf{s}'$ such that $\mathbf{s}' \in \rho_g(\mathbf{s}, \mathbf{a})$.

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_i}, where n_i is the length of the formula path formula in E_i. Hence the automaton A_E is of size $2^{\Sigma n_i} = 2^n$.

The Universal Automaton

Here we build an automaton for each of the formulas U_i of the form $\neg \exists g$ that are in $scl(f)$. Each of these automata checks that whenever the formula, $\neg \exists g \in T[x]$, there is no leftist path from x that satisfies g, or equivalently that all leftist paths from x satisfy $\neg g$. To do this, we could construct the Büchi sequential automaton for the formula $\neg g$ and run it over all leftist paths starting at every node x where $\neg \exists g \in x$. 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 $2^{scl}(f)$) corresponding to the linear time temporal logic formula $G(\neg \exists g \supset \neg g)$. Let this automaton be $A_g = (2^{scl(f)}, S_g, \rho_g, S_{0g}, F_g)$. The automaton on leftist trees for the formula $U_i = \neg \exists g$ is then $A_{U_i} = (2^{scl(f)}, 2^{S_g} \times S_g, \rho_{U_i}, \{S_{0g}\} \times S_{0g}, F_g)$. We have that $(\mathbf{s}_1, \ldots, \mathbf{s}_{n+1}) \in \rho_{U_i}(\mathbf{s}, \mathbf{a})$ whenever the following conditions are satisfied, for $\mathbf{s} = (X, s)$:

- $\mathbf{s}_1 = (Y, s')$ where $s' \in \rho_g(s, \mathbf{a})$ and $Y = \{y \in S_g \mid y \in \rho_g(x, \mathbf{a})$, for some $x \in X\}$.
- For $2 \leq j \leq n+1$, $\mathbf{s}_j = (Y, s')$ where $s' \in \rho_g(x, \mathbf{a})$ for some $x \in X$, and $Y = \{y \in S_g \mid y \in \rho_g(x, \mathbf{a})$, for some $x \in X\}$.

Intuitively, what our construction does is to apply the classical Rabin-Scott [RS59] construction to the automaton A_g and run the resulting deterministic automaton down the tree, while simultaneously running a nondeterministic copy of A_g 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 $2^{2^{n_i}}$, 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 $O(\exp^2(n))$ []

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 $\exists((pUq)Ur)$, 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 4.6: A CTL^* formula f has a fair model iff it has a finite fair 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

[Ab80] K. Abrahamson, "*Decidability and Expressiveness of Logics of Programs*", Ph.D. Thesis, University of Washington at Seattle, 1980.

[BMP81] M. Ben-Ari, Z. Manna, A. Pnueli, "The Temporal Logic of Branching Time", *Proc. 8th ACM Symp. on Principles of Programming Languages,* Williamsburg, January 1981, pp. 164--176.

[CES83] E.M. Clarke, E.A., Emerson, A.P. Sistla, "Automatic Verification of Finite-State Concurrent Systems Using Temporal Logics Specifications: A Practical Approach", *Proc. 10th ACM Symp. on Principles of Programming Languages,* Austin, 1983, pp. 117-126.

[EC82] E. A. Emerson, E. M. Clarke, "Using Branching Time Logic to Synthesize Synchronization Skeletons", *Science of Computer Programming,* 2(1982), pp. 241-266.

[EH85] E.A. Emerson, J.Y. Halpern,"Decision Procedures and Expressiveness in the Temporal Logic of Branching Time", *J. Computer and System Sciences,* 30(1985), pp. 1-34.

[EH83] E.A. Emerson, J.Y. Halpern, ""Sometimes" and "Not Never" Revisited: On Branching vs. Linear Time", *Proc. 10th ACM Symp. on Principles of Programming Languages,* Austin, 1983, pp. 127-140.

[EL85a] E.A. Emerson, C.L. Lei, "Temporal Model Checking under Generalized Fairness Constraints". *Proc. 18th Hawaii Int'l Conference on System Sciences,* 1985.

[EL85b] E.A. Emerson, C.L. Lei, "Modalities for Model Checking: Branching Time Strikes Back", *Proc. 12th ACM Symp. on Principles of Programming Languages,* New Orleans, 1985, pp. 84-96.

[Em83] E. A. Emerson, "Alternative Semantics for Temporal Logic", *Theoretical Computer Science,* 26(1983), pp. 120-130.

[Em85a] E. A. Emerson, "Automata, Tableau and Temporal Logic", in *Logics of Programs,* Lecture Notes in Computer Science, vol. 193, Springer-Verlag, Berlin, 1985, pp. 79-88.

[ES84] E.A. Emerson, A.P. Sistla,"Deciding Branching Time Logic", *Information and Control,* 61(1984), pp. 175-201.

[FK84] N. Francez, D. Kozen, "Generalized Fair Termination", *Proc. 11th ACM Symp. on Principles of Programming Languages,* Salt Lake City, 1983, pp. 46-53.

[Ga76] D.M. Gabbay, *"Investigation in Modal and Tense Logic"* Reidel, 1976.

[GPSS80] D. Gabbay, A. Pnueli, S. Shelah and J. Stavi, "The Temporal Analysis of Fairness", *Proc. 7th ACM Symp. on Principles of Programming Languages,* Las Vegas, 1980, pp. 163-173.

[HS84] S. Hart, M. Sharir, "Probabilistic Temporal Logics for Finite and Bounded Models", *Proc. 16th ACM Symp. on Theory of Computing,* Washington, 1984, pp. 1-13.

[JL77] N.D. Jones, W.T. Laaser, "Complete Problems in Deterministic Polynomial Time", *Theoretical Computer Science* 3(1977), pp. 105-117.

[Kr84] S. Kraus, "*The decision problem for TC*" (in Hebrew), M.Sc. Thesis, The Hebrew University in Jerusalem, 1984.

[KSK66] J.G. Kemeny, J.L. Snell, A.W. Knapp, *"Denumerable Markov Chains",* D. van Nostrand Company, 1966.

[KL83] S. Kraus, D. Lehmann, "Decision Procedures for Time and Chance", *Proc. 24th IEEE Symp. on Foundations of Computer Science,* Tucson, 1983, pp. 202-209.

[La80] L. Lamport, "Sometimes is Sometimes Not Never", *Proc. 7th ACM Symp. on Principles of Programming Languages,* Las Vegas, January 1980, pp. 174--185.

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

Their Linear Specification", *Proc. 12th ACM Symp. on Principles of Programming Languages,* New Orleans, 1985, pp. 97-107.

[LPS81] D. Lehman, A. Pnueli, J. Stavi, "Impartiality, Justice, and Fairness: The Ethics of Concurrent Termination", *Proc. 8th Int'l Colloq. on Automata, Languages, and Programming 1981,* Lecture Notes in Computer Science - Vol. 115, Springer-Verlag, pp.264-277.

[LS82] D. Lehman, S. Shelah, "Reasoning with Time and Chance", *Information and Control* 53(1982), pp. 165-198.

[McN66] R. McNaughton, "Testing and Generating Infinite Sequences by a Finite Automaton", *Information and Control* 9 (1966), pp. 521-530

[MW84] Z. Manna, P. Wolper, "Synthesis of Communicating Processes from Temporal Logic Specifications", *ACM Trans. on Programming Languages and Systems,* 6(1984), pp. 68-93.

[Pn77] A. Pnueli, "The Temporal Logic of Programs", *Proc. 8th IEEE Symp. on Foundations of Computer Science,* Providence, 1977, pp. 46-57.

[Pn81] A. Pnueli, "The Temporal Logic of Concurrent Programs", *Theoretical Computer Science* 13(1981), pp. 45-60.

[Pn83] A. Pnueli, "On the Extremely Fair Treatment of Probabilistic Algorithms", *Proc. 15th ACM Symp. on Theory of Computing,* Boston, 1983, pp. 278-290.

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

[Ra70] M.O. Rabin, "Weakly Definable Relations and Special Automata", *Proc. Symp. Math. Logic and Foundations of Set Theory* (Y. Bar-Hillel, ed.), North-Holland, 1970, pp. 1-23.

[Ra72] M.O. Rabin, "Automata on Infinite Objects and Church's Problem", *Proc. Regional AMS Conf. Series in Math.* 13(1972), pp. 1-22.

[RS59] M. O. Rabin, D. Scott, "Finite Automata and their Decision Problems", *IBM J. Res. & Dev.,* 3(1959), pp 114-125.

[St82] R.S. Streett, "Propositional Dynamic Logic of Looping and Converse", *Information and Control* 54(1982), pp. 121-141.

[VS85] M. Y. Vardi, L. Stockmeyer, "Improved Upper and Lower Bounds for Modal Logics of Programs", *Proc. ACM 17th Symposium on Theory of Computing,* Providence, May 1985, pp. 240-251.

[VW84] M. Y. Vardi, P. Wolper, "Automata Theoretic Techniques for Modal Logics of Programs", *Proc. ACM Symp. on Theory of Computing,* Washington, April 1984, pp. 446-456 (complete version to appear in *J. Computer and System Sciences*).

[WVS83] P. Wolper, M. Y. Vardi, A. P. Sistla, "Reasoning about Infinite Computation Paths", *Proc. 24th IEEE Symp. on Foundations of Computer Science,* Tucson, 1983, pp. 185-194.