Reasoning about Infinite Computations[*]

Moshe Y. Vardi[†]
IBM Almaden Research Center

Pierre Wolper[‡]
Université de Liège

Abstract

We investigate extensions of temporal logic by connectives defined by finite automata on infinite words. We consider three different logics, corresponding to three different types of acceptance conditions (finite, looping and repeating) for the automata. It turns out, however, that these logics all have the same expressive power and that their decision problems are all PSPACE-complete. We also investigate connectives defined by alternating automata and show that they do not increase the expressive power of the logic or the complexity of the decision problem.

1 Introduction

For many years, logics of programs have been tools for reasoning about the input/output behavior of programs. When dealing with concurrent or nonterminating processes (like operating systems) there is, however, a need to reason about infinite computations. Thus, instead of considering the first and last states of finite computations, we need to consider the infinite sequences of states that the program goes through. Logics to reason about such sequences include *temporal logic* [Pn77] and temporal-logic-based *process logics* [Ni80, HKP80].

In the propositional case, computations can be viewed as infinite sequences of propositional truth assignments. For reasoning about *individual* propositional truth assignments, propositional logic is a descriptively complete language, i.e., it can specify any set of propositional truth assignments. There is, however, no a priori robust notion of descriptive completeness for reasoning about sequences of propositional truth assignments.

[*]A preliminary version of this paper, authored by P. Wolper, M.Y. Vardi, and A.P. Sistla, appeared in *Proc. 24th IEEE Symp. on Foundations of Computer Science*, 1983, pp. 185–194, under the title "Reasoning about Infinite Computation Paths".

[†]Address: IBM Almaden Research K53-802, 650 Harry Rd., San Jose, CA 95120-6099,USA, vardi@almaden.ibm.com

[‡]Address: Institut Montéfiore, B28, Université de Liège, B-4000 Liège Sart-Tilman, Belgium, pw@montefiore.ulg.ac.be

In [GPSS80] propositional temporal logic (PTL) was shown to be expressively equivalent to the monadic first-order theory of $(N, <)$, the natural numbers with the less-than relation. This was taken as an indication that PTL and the process logics based on it are "descriptively complete".

The above assumes that first-order notions are all we need to reason about infinite computations. But, the very fundamental notion of regularity of sets of event sequences is not first-order; even the simple assertion that "the proposition p holds at least in every other state on a path" is not expressible in PTL [Wo83]. In fact, PTL and the first-order theory of $(N, <)$ are known to be expressively equivalent to star-free ω-regular languages [Lad77, Tho79, Tho81]. On the other hand, ω-regular sequences are a natural way of describing concurrent processes [Sh79], [Mi80]; and furthermore, the ability to describe ω-regular sequences is crucial to the task of program verification [LPZ85].

There are several different ways to extend the expressive power of PTL. We could add some non-first-order construct, such as least-fixpoint or second-order quantification, but we prefer here to add an explicit mechanism for specifying ω-regular events as was done in [Wo83]. There, PTL is extended with a temporal connective corresponding to every nondeterministic finite automaton on infinite words.[1,2] For example, if $\Sigma = \{a, b\}$ and A is the automaton accepting all infinite words over Σ having a in every other position, then A is also a binary temporal connective, and the formula $A(p, true)$ is satisfied by the paths where p holds in at least every other state.

An important point that was not considered in [Wo83] is that to define ω-regular sequences one needs to impose certain *repeating* conditions on accepting automata runs. For example, Büchi automata, which define exactly the ω-regular languages, require that some accepting state occurs infinitely often in the run [Bu62, McN66]. Using Büchi automata to define temporal connectives gives rise to an extended temporal logic that we call ETL_r.

In addition to the notion of *repeating acceptance*, we also consider two other notions of acceptance. The first notion is *finite acceptance*: an automaton accepts an infinite word if it accepts some prefix of the word by the standard notion of acceptance for finite words. The second notion is *looping acceptance*: an automaton accepts an infinite word if it has some infinite run over the word (this is the notion used in [Wo83]). Using automata with finite and looping acceptance conditions to define temporal connectives gives rise to extended temporal logics that we call ETL_f and ETL_l, respectively. We should note that our interest in finite and looping acceptance is not due to their automata-theoretic

[1]To be precise, the formalism used in [Wo83] is that of right-linear context-free grammars over infinite words.

[2]Note that here the use of automata is conceptually different from the use of automata in dynamic logic (cf. [Ha84, HS84, Pr81]). In dynamic logic automata are used to describe flowchart programs, while here automata are used to describe temporal sequences. Thus, dynamic logic automata describe regular sequences of program statements, while temporal logic automata describe regular properties of state sequences.

significance,[3] but due to their significance as specification constructs. Finite acceptance can be viewed as describing a *liveness* property, i.e., "something good will eventually happen", while looping acceptance can be viewed as a *safety* condition, i.e., "something bad will never happen".[4] Thus, finite and looping acceptance can be viewed as extensions of the "*Eventually* " and "*Always*" constructs of PTL.

The notions of finite and looping acceptance are incomparable and are strictly weaker than the notion of repeating acceptance. For example, the sequence $(ab^*)^\omega$ can be defined by repeating acceptance but not by finite or looping acceptance. Thus, we could expect the logics ETL_r, ETL_f, and ETL_l to have different expressive powers. One of the main results of this paper is that all these logics are expressively equivalent. They all have the expressive power of ω-regular expressions, which by [Bu62] is the same as that of the monadic second-order theory of $(N, <)$, usually denoted S1S. We also consider the complexity of the decision problem for ETL_f and ETL_l. It turns out that both logics have the same complexity as PTL: polynomial space (cf. [HR83, SC85]). In contrast, the decision problem for S1S is nonelementary [Me75], as is the emptiness problem for regular expressions with complement [MS73].[5]

An important contribution of this paper is to show that temporal logic formulas can be directly compiled into equivalent Büchi automata. To make the construction and its proof of correctness easier and more systematic, we first define variants of Büchi automata that are geared towards recognizing models of temporal formulas and prove that these variants can be converted to Büchi automata. We then use our construction to obtain decision procedures. Our approach is, starting with a formula, to build an equivalent Büchi automaton and then to check that this automaton is nonempty.[6] Note that the construction of Büchi automata from temporal logic formulas is not only useful for obtaining decision procedures for the logic but is also the cornerstone of synthesis and verification methods based on temporal logic [MW84, PR89, VW86b, Wo89]. This construction is also the cornerstone for applications of temporal logic to the verification of probabilistic and real-time programs (cf. [AH90, Va85b].

Finally, to explore the full power of our technique, we introduce *alternating* finite automata connectives. These can be exponentially more succinct than nondeterministic

[3]For an extensive study of acceptance conditions for ω-automata see [Ch74, Ka85, Lan69, MY88, Sta87, Tho90, Wa79]. Our three conditions corresponds to the first three conditions in [Lan69]. It is known that these three conditions exhaust all the possibilities in the Landweber classification (cf. [Wa79]).

[4]The notions of safety and liveness are due to Lamport [Lam77]. Our notion of liveness here corresponds to the notion of *guarantee* in [MP89].

[5]In [SVW87] it is shown that the decision problem for ETL_r is also PSPACE-complete. This result requires significantly more complicated automata-theoretic techniques that are beyond the scope of this paper. For other comments on the complexity of ETL_r see Section 3.

[6]The automata-theoretic approach described here can be viewed as a specialization of the automata-theoretic approach to decision problems of dynamic logic described in [VW86a] (see also [ES84, St82]). Note, however, that while the tree automata constructed in [ES84, St82, VW86a] accept only *some* models of the given formulas, the automata constructed here accept *all* models of the given formulas.

automata connectives, and one may expect their introduction to push the complexity of the logic up. We investigate both ATL_f and ATL_l, which are the alternating analogues of ETL_f and RTL_l. Surprisingly, we show that the decision problems for these logics are still PSPACE-complete.

2 Finite Automata on Infinite Words

In this section we define several classes of finite automata on infinite words and examine their nonemptiness problem. The framework developed here is a specialization of the tree-automata framework developed in [VW86a]. In view of the wide range of applications of temporal logic (cf. [BBP89]), we describe the word-automata framework in detail, in order to make the paper self-contained.

2.1 Definitions

We start by making our notation for infinite words precise. We denote by ω the set of natural numbers $\{0, 1, 2, \ldots\}$. We use the notation $[i, j]$ for the interval $\{i, i+1, \ldots, j\}$ of ω. An infinite word over an alphabet Σ is a function $w : \omega \to \Sigma$. The ith letter in an infinite word w is thus $w(i)$. We will often denote it by w_i and write $w = w_0 w_1 \ldots$.

A nondeterministic finite automaton (abbr. NFA) on infinite words is a tuple $A = (\Sigma, S, \rho, S_0, F)$ where:

- Σ is a finite alphabet of letters,
- S is a finite set of states,
- $\rho : S \times \Sigma \to 2^S$ is the transition function, mapping each state and letter to a set of possible successor states,
- S_0 is a set of initial states, and
- $F \subseteq S$ is a set of accepting states.

The automaton is deterministic if $\mid S_0 \mid = 1$ and $\mid \rho(s, a) \mid \leq 1$ for all $s \in S$ and $a \in \Sigma$.

A *finite run* of A over an infinite word $w = w_0 w_1 \ldots$, is a finite sequence $\sigma = s_0 s_1 \ldots s_{n-1}$, where $s_0 \in S_0$, $s_{i+1} \in \rho(s_i, w_i)$, for all $0 \leq i < n-1$. A *run* of A over an infinite word $w = w_0 w_1 \ldots$, is an infinite sequence $\sigma = s_0 s_1 \ldots$, where $s_0 \in S_0$, $s_{i+1} \in \rho(s_i, w_i)$, for all $i \geq 0$.

Depending on the acceptance condition we impose, we get different types of automata. In *finite acceptance automata*, a finite run $\sigma = s_0 s_1 \ldots s_{n-1}$ is *accepting* if $s_{n-1} \in F$. In *looping acceptance automata*, any run $\sigma = s_0 s_1 \ldots$ is *accepting*, i.e., no condition is imposed by the set F. In *repeating acceptance automata (Büchi automata* [Bu62]), a run

$\sigma = s_0 s_1 \ldots$ is *accepting* if there is some accepting state that repeats infinitely often, i.e., for some $s \in F$ there are infinitely many i's such that $s_i = s$.

In all three types of automata, an infinite word w is *accepted* if there is some accepting run over w. The set of infinite words accepted by an automaton A is denoted $L(A)$.

The three different types of automata recognize different classes of languages. It turns out that the class of languages accepted by repeating acceptance automata (L_{repeat}) strictly contains the class of languages accepted by looping (L_{loop}) and finite acceptance (L_{finite}) automata. These last two classes are incomparable. The languages accepted by Büchi (repeating acceptance) automata are often called the ω-regular languages. By results of Büchi [Bu62] (see also McNaughton [McN66]), this class of languages is closed under union, intersection and complementation and is equivalent to the class of languages describable in the monadic second-order theory of one successor ($S1S$) and by ω-regular expressions. The monadic second-order theory of successor is the interpreted formalism which has individual variables ranging over the natural numbers, monadic predicate variables ranging over arbitrary sets of natural numbers, the constant 0, the successor function, boolean connectives and quantification over both types of variables. ω-regular expressions are expressions of the form $\cup_i \alpha_i (\beta_i)^\omega$ where the union is finite, α and β are classical regular expressions, and ω denotes countable repetition.

To establish the relations between L_{repeat}, L_{loop} and L_{finite}, we state the following well-known facts (cf. [Ch74, Ka85, Lan69, MY88, Sta87, Tho90, Wa79]).

Lemma 2.1:

1. *The language a^ω over the alphabet $\Sigma = \{a, b\}$ is in L_{loop} but not in L_{finite}.*
2. *The language $a^\star b(a \cup b)^\omega$ over the alphabet $\Sigma = \{a, b\}$ is in L_{finite} but not in L_{loop}.*
3. *The language $(a^\star b)^\omega$ over the alphabet $\Sigma = \{a, b\}$ is in L_{repeat}, but not in L_{finite} or in L_{loop}.*

Corollary 2.2:

1. *L_{repeat} strictly contains L_{finite} and L_{loop}.*
2. *L_{finite} and L_{loop} are incomparable.*

For the rest of this section, we will only deal with Büchi automata.

2.2 The Nonemptiness Problem for Büchi Automata

An important problem we will have to solve for Büchi automata is the nonemptiness problem, which is, given a Büchi automaton, to determine whether it accepts at least one word. We are interested in the complexity of this problem as a function in the *size* of the given automaton. (We assume some standard encoding of automata, so the size of an automaton is the length of its encoding.)

Let $A = (\Sigma, S, \rho, S_0, F)$ be an automaton. We say that a state t is *reachable* from a state s if there are a finite word $w = w_1 \ldots w_k$ in $\Sigma^\star$ and a finite sequence $s_0, \ldots, s_k$ of states in S such that $s_0 = s$, $s_k = t$, and $s_{i+1} \in \rho(s_i, w_{i+1})$ for $0 \leq i \leq k-1$.

Lemma 2.3: *[TB73] A Büchi automaton accepts some word iff there is an accepting state of the automaton that is reachable from some initial state and is reachable from itself.*

We can now prove the following:

Theorem 2.4: *The nonemptiness problem for Büchi automata is logspace-complete for NLOGSPACE.*

Proof: We first prove that the problem is in NLOGSPACE. Given Lemma 2.3, to determine if a Büchi automaton accepts some word, we only need to check if there is some accepting state reachable from some initial state and reachable from itself. To do this, we nondeterministically guess an initial state s and an accepting state r; we then nondeterministically attempt to construct a path from s to r and from r to itself. To construct a path from any state x to any other state y, we proceed as follows:

1. Make x the current state.
2. Choose a transition from the current state and replace the current state by the target of this transition.
3. If the current state is y, stop. Otherwise repeat from step (2).

At each point only three states are remembered. Thus the algorithm requires only logarithmic space.

To show NLOGSPACE hardness, given Lemma 2.3, it is straightforward to construct a reduction from the graph accessibility problem, proved to be NLOGSPACE complete in [Jo75]. ∎

To solve the satisfiability problem for extended temporal logic, we will proceed as follows: build a Büchi automaton accepting the models of the formula and determine if that automaton is nonempty. The automata we will build are exponential in the size of the formula. However, as the fact that the nonemptiness problem is in NLOGSPACE indicates, it is possible to solve the satisfiability problem using only polynomial space. The argument is that it is not necessary to first build the whole automaton before applying the algorithm for nonemptiness. (A similar argument, though in a somewhat different framework, is used in [HR83, SC85, Wo83].) We now make this argument more precise.

Let Δ be some fixed finite alphabet. A *problem* P is a subset of $\Delta^\star$. An *instance* is an element x of $\Delta^\star$ for which we want to determine membership in P. Assume that we also use Δ to encode Büchi automata and their constituent elements (alphabet,states,...).

Lemma 2.5: *Let P be a problem, let f be a polynomial, let $\alpha, \beta, \gamma, \delta, \theta$ be algorithms that associate with any instance x the encoding of a Büchi automaton*

$$A_x = (\Sigma_x, S_x, \rho_x, S_0^x, F_x)$$

in the following manner:

1. *if $y \in \Sigma_x$ or $y \in S_x$, then $\mid y \mid \leq f(\mid x \mid)$,*
2. *given $y \in \Delta^\star$, α determines whether $y \in \Sigma_x$ using at most $f(\mid x \mid)$ space,*
3. *given $y \in \Delta^\star$, β determines whether $y \in S_x$ using at most $f(\mid x \mid)$ space,*
4. *given $y \in \Delta^\star$, γ determines whether $y \in S_0^x$ using at most $f(\mid x \mid)$ space,*
5. *given $y \in \Delta^\star$, δ determines whether $y \in F_x$ using at most $f(\mid x \mid)$ space,*
6. *given $u, v, w \in \Delta^\star$, θ determines whether $u \in \rho_x(v, w)$, using at most $f(\mid x \mid)$ space, and*
7. *$x \in P$ iff A_x accepts some word.*

Then there is a polynomial space algorithm for determining membership in P.

Proof: We use the algorithm described in the proof of Theorem 2.4 to check if A_x is nonempty. Given the assumptions we have made, each of the steps in the algorithm can be executed in polynomial space. Moreover, the two states being remembered are also of size polynomial in the size of the problem. So, the whole algorithm requires polynomial space. This establishes that the problem is in NPSPACE and hence PSPACE. ▮

When constructing Büchi automata from temporal logic formulas, we will often have to take the intersection of several Büchi automata. The following lemma is a special case of Theorem A.1 in [VW86a] and extends a construction of [Ch74] (see also [ES84]).

Theorem 2.6: *Let $A_0, \ldots, A_{k-1}$ be Büchi automata. There is a Büchi automaton A with $k \times \Pi_{i=0}^{k-1} |A_i|$ states such that $L(A) = L(A_0) \cap \ldots \cap L(A_{k-1})$.*

Proof: Let $A_i = (\Sigma, S^i, \rho^i, S_0^i, F^i)$. Define $A = (\Sigma, S, \rho, S_0, F)$ as follows: $S = S^0 \times \ldots \times S^{k-1} \times \{0, \ldots, k-1\}$, $S_0 = S_0^0 \times \ldots \times S_0^{k-1} \times \{0\}$, $F = F^0 \times S^1 \times \ldots \times S^{k-1} \times \{0\}$, and

> $(s_1^0, \ldots, s_1^{k-1}, j) \in \rho((s^0, \ldots, s^{k-1}, i), a)$ iff $s_1^l \in \rho(s^l, a)$, for $0 \leq l \leq k-1$, and either $s^i \notin F^i$ and $i = j$ or $s^i \in F^i$ and $j = (i+1) \bmod k$.

The automaton A consists of k copies of the cross product of the automata A_i. Intuitively, one starts by running the first copy of this cross product. One then jumps from the copy i to the copy $(i+1) \bmod k$ when a final state of A_i is encountered. The acceptance condition imposes that one goes infinitely often through a final state of A_0 in the copy 0. This forces all the components of A to visit their accepting states in a cyclic order. We leave it to the reader to formally prove that $L(A) = L(A_0) \cap \ldots \cap L(A_{k-1})$. ▮

2.3 Subword Automata

To make the construction of Büchi automata from extended temporal logic formulas easier, we define more specialized classes of automata: *subword automata* and *set-subword automata*. They are the specialization to words of *subtree automata* and *set-subtree automata* defined in [VW86b]. For completeness sake, we give a detailed treatment of this specialization.

Intuitively, a subword automaton checks that, starting at every position in an infinite word, there is a finite word accepted by some automaton. The automata accepting finite words at the various positions differ only by their initial state. The initial state corresponding to a position is determined by the symbol appearing at that position in the infinite word. Formally, a subword automaton A is a tuple $(\Sigma, S, \rho, \xi, F)$, where

- Σ is the alphabet,
- S is the state set,
- $\rho : S \times \Sigma \rightarrow 2^S$ is the transition function,
- $\xi : \Sigma \rightarrow S$ is the labeling function, and
- $F \subseteq S$ is the nonempty set of accepting states.

An infinite word $w \in \Sigma^\omega$ is accepted by A if the following two conditions hold:

- *labeling condition*: $\xi(w_{i+1}) \in \rho(\xi(w_i), w_i)$ for every $i \in \omega$
- *subword condition*: for every $i \in \omega$, there exists some $j \geq i$ and a mapping $\varphi : [i, j] \rightarrow S$ such that $\varphi(i) = \xi(w_i)$, $\varphi(j) \in F$, and for all k, $i \leq k < j$, $\varphi(k+1) \in \rho(\varphi(k), w_k)$.

The labeling condition requires the labeling ξ of the word to be compatible with the transition function of A. The subword condition, requires that from each position i in the word, there be a subword accepted by A viewed as an automaton on finite words with initial state $\xi(w_i)$.

Using a construction similar to the *flag construction* of Choueka [Ch74], a subword automaton, even without the labeling condition, can be converted to a Büchi automaton with at most an exponential increase in size. We show now that because of the labeling condition we can do this conversion with only a quadratic increase in size. Before proving this we need a technical lemma.

Lemma 2.7: *Let $A = (\Sigma, S, \rho, \xi, F)$ be a subword automaton. Then A accepts a word $w : \omega \rightarrow \Sigma$ iff*

- $\xi(w_{i+1}) \in \rho(\xi(w_i), w_i)$ *for every $i \in \omega$, and*

- *for every $i \in \omega$, there exists some $j > i$ and a mapping $\varphi : [i, j] \to S$ such that $\varphi(i) = \xi(w_i)$, $\varphi(j) \in F$, and for all k, $i \leq k < j$, $\varphi(k+1) \in \rho(\varphi(k), w_k)$.*

Proof: The only difference between the condition in the lemma and the standard condition of acceptance is the requirement that the interval $[i, j]$ contain at least two points and hence that the accepted subword $w_i \ldots w_{j-1}$ be nonempty. Thus the "if" direction is trivial. For the "only if" direction assume that A accepts w. The labeling condition clearly holds, so it remains to show the existence of the "right" subword.

Let $i \in \omega$. Then, there exists some $j \geq i$ and a mapping $\varphi_i : [i, j] \to S$ that satisfies the subword condition at i. If $i > j$, then we are done. So, let us assume $i = j$. We know that there exists some $k \geq i + 1$ and a mapping $\varphi_{i+1} : [i+1, k] \to S$ that satisfies the subword condition at point $i + 1$. We claim that the interval $[i, k]$ satisfies the subword condition at i. Indeed, because of the labeling condition, the mapping $\varphi = \varphi_i \cup \varphi_{i+1}$ satisfies the required conditions ▮

Theorem 2.8: *Every subword automaton with m states is equivalent to a Büchi automaton with $O(m^2)$ states.*

Proof: Let $A = (S, \Sigma, \rho, \xi, F)$ be a subword automaton. We now define two new transition functions $\rho_1, \rho_2 : S \times \Sigma \to 2^S$:

- $\rho_1(s, a) = \rho(s, a)$ if $s = \xi(a)$, and $\rho_1(s, a) = \emptyset$ otherwise.
- $\rho_2(s, a) = \rho(\xi(a), a)$ if $s \in F$, and $\rho_2(s, a) = \rho(s, a)$ otherwise.

These transition functions let us define two Büchi automata $B_1 = (\Sigma, S, \rho_1, S, S)$ and $B_2 = (\Sigma, S, \rho_2, S, F)$. Basically, B_1 will take care of checking the labeling condition and B_2 of checking the subword condition.

Let us show that a word $w : \omega \to \Sigma$ is accepted by A iff it is accepted by both B_1 and B_2.

Suppose first that w is accepted by B_1 and B_2. This means there are accepting runs $\sigma_1 = s_{10}, s_{11}, \ldots$ and $\sigma_2 = s_{20}, s_{21}, \ldots$ of B_1 and B_2 (resp.) on w. We verify first that the labeling property holds. Clearly, $\rho_1(s_{1i}, w_i) \neq \emptyset$ for every $i \in \omega$. Thus, $s_{1i} = \xi(w_i)$. Consequently, for every $i \in \omega$,

$$\xi(w_{i+1}) = s_{1,i+1} \in \rho_1(s_{1i}, w_i) = \rho(\xi(w_i), w_i).$$

It remains to verify the subword condition.

Let $i \geq 0$. As the run σ_2 is accepting, there is some $j \geq i$ such that $s_{2j} \in F$. For the same reason, there is a $k > j$ such that $s_{2k} \in F$. Assume without loss of generality that $s_{2l} \notin F$ for all $j < l < k$. Consider the interval $[i, k]$. We claim that it satisfies the subword condition at i.

Indeed, a suitable mapping $\varphi : [i,k] \rightarrow S$ can be defined in the following way: $\varphi(l) = \xi(w_l)$ for $l \in [i,j]$ and $\varphi(l) = s_{2l}$ for $l \in [j+1,k]$. Clearly $\varphi(k) \in F$. For $i \leq l < j$ we have

$$\varphi(l+1) = \xi(w_{l+1}) \in \rho(\xi(w_l), w_l) = \rho(\varphi(l), w_l).$$

For $l = j$ we have

$$\varphi(l+1) = s_{2l+1} \in \rho_2(s_{2l}, w_l) = \rho(\xi(w_l), w_l) = \rho(\varphi(l), w_l)$$

as $s_{2j} \in F$. Finally, for $j+1 \leq l < k$

$$\varphi(l+1) = s_{2l+1} \in \rho_2(s_{2l}, w_l) = \rho(s_{2l}, w_l) = \rho(\varphi(l), w_l)$$

as $s_{2l} \notin F$ for $j+1 \leq l < k$.

We now have to show that if w is accepted by A then it is accepted by both B_1 and B_2. Consider the run $\sigma_1 = s_{10}, s_{11}, s_{12}, \ldots$ where $s_{1i} = \xi(w_i)$. By the labeling condition, σ_1 is an accepting run of B_1 on w.

We now define a computation $\sigma_2 = s_{20}, s_{21}, s_{22}, \ldots$ of B_2 by defining it on a sequence of increasing intervals $[0,j_1], [0,j_2], \ldots$ such that for each j_i, $s_{2j_i} \in F$. The first interval is $[0,0]$ and we define $s_{20} = s$, where s is an arbitrary member of F. Suppose now that σ_2 is defined on the interval $[0,j_n]$. By induction, $s_{2j_n} \in F$. By Lemma 2.7, there exists an interval $[j_n, j_{n+1}]$, $j_{n+1} > j_n$ and a mapping $\varphi : [j_n, j_{n+1}] \rightarrow S$ such that $\varphi(j_n) = \xi(w_{j_n})$, $\varphi(j_{n+1}) \in F$, and for all k, $j_n \leq k < j_{n+1}$, $\varphi(k+1) \in \rho(\varphi(k), w_k)$. Without loss of generality, we can assume that $\varphi(k) \notin F$ for $j_n < k < j_{n+1}$. For $j_n < k \leq j_{n+1}$, we define $s_{2k} = \varphi(k)$. We show now that for every $j_n \leq k < j_{n+1}$, $s_{2,k+1} \in \rho_2(s_{2k}, w_k)$. Indeed, if $k = j_n$, then $s_{2k} = \xi(w_{j_n})$ and $s_{2k} \in F$. Consequently,

$$s_{2,k+1} = \varphi(k+1) \in \rho(\xi(w_k), w_k) = \rho_2(s_{2k}, w_k).$$

If $j_n < k < j_{n+1}$, then $s_{2k} = \varphi(k) \notin F$. Consequently,

$$s_{2,k+1} = \varphi(k+1) \in \rho(\varphi(k), w_k) = \rho_2(s_{2k}, w(k)).$$

Clearly, $\bigcup_{n=1}^{\infty}[0,j_n] = \omega$. Hence we have defined a run σ_2 of B_2. Moreover, as for each j_n, $s_{2j_n} \in F$, the run is accepting.

We have shown that w is accepted by both B_1 and B_2. Finally, by Theorem 2.6, we can construct an automaton B such that a word w is accepted by B iff it is accepted by both B_1 and B_2. It follows that the subword automaton A is equivalent to the Büchi automaton B. ∎

Subword automata are more adequate than Büchi automata for the purpose of reducing satisfiability of formulas to nonemptiness of automata. Nevertheless, to facilitate our task in the rest of the paper, we now specialize the notion of subword automata even further. The words that we shall deal with are going to consists of sets of formulas, and the states of the automata that will accept these words are also going to be sets of formulas. Thus, we consider automata where the alphabet and the set of states are the same set and have the structure of a power set. We will call these automata *set-subword automata*.

Formally, a set-subword automaton A is a pair (Ψ, ρ), where

- Ψ is a finite set of basic symbols (in fact these symbols will be just formulas of the logic). The power set 2^Ψ serves both as the alphabet Σ and as the state set $\mathbf{S}$. The empty set serves as a single accepting state. We will denote elements of 2^Ψ by $\mathbf{a}, \mathbf{b}, \ldots$ when viewed as letters from the alphabet Σ, and by $\mathbf{s}, \mathbf{s}_1, \ldots$ when viewed as elements of the state set $\mathbf{S}$. Intuitively, a letter is a set of formulas that are "alleged" to be true and a state is a set of formulas that for which the automaton tries to verify the "allegation".
- $\rho : \mathbf{S} \times \Sigma \rightarrow 2^{\mathbf{S}}$ is a transition function such that
    1. $\rho(\mathbf{s}, \mathbf{a}) \neq \emptyset$ iff $\mathbf{s} \subseteq \mathbf{a}$,
    2. $\emptyset \in \rho(\emptyset, \mathbf{a})$,
    3. if $\mathbf{s} \subseteq \mathbf{s}'$, $\mathbf{s}_1 \subseteq \mathbf{s}_1'$ and $\mathbf{s}_1 \in \rho(\mathbf{s}', \mathbf{a})$, then $\mathbf{s}_1' \in \rho(\mathbf{s}, \mathbf{a})$,
    4. if $\mathbf{s}_1' \in \rho(\mathbf{s}_1, \mathbf{a})$ and $\mathbf{s}_2' \in \rho(\mathbf{s}_2, \mathbf{a})$, then $\mathbf{s}_1' \cup \mathbf{s}_2' \in \rho(\mathbf{s}_1 \cup \mathbf{s}_2, \mathbf{a})$.

A word $w : \omega \rightarrow \Sigma$ is accepted by A if for every $i \in \omega$ and every $f \in w_i$ there exists a finite interval $[i, j]$ and a mapping $\varphi : [i, j] \rightarrow \mathbf{S}$ such that

- $\varphi(i) = \{f\}$,
- $\varphi(j) = \emptyset$, and
- for all $i \leq k < j$, $\varphi(k+1) \in \rho(\varphi(k), w_k)$.

The acceptance condition requires that for each position i in the word and for each formula f in w_i, the "right" formulas appear in the letters $w_{i+1}, w_{i+2}, \ldots$. Intuitively, the transition of the automaton are meant to capture the fact that if a certain formula appears in w_i, then certain formulas must appear in w_{i+1}. The four conditions imposed on the transition relation ρ can be explained as follows:

1. The formulas in the state of the automaton are formulas that the automaton is trying to verify. A minimal requirement is that these formulas appear in the letter of the scanned position of the word. As we will see, this condition is related to the labeling condition defined for subword automata.

2-3. These are what we call *monotonicity* conditions. A transition of the automaton is a minimum requirement on the formulas in w_{i+1} given the formulas that the automaton is trying to verify at i. Clearly if there is nothing to verify at i, then nothing is required at $i+1$ (condition (2)). Also, the transition is still legal if we try to verify fewer formulas at i or more formulas at $i+1$.

4. This is an *additivity* condition. It says that there is no interaction between different formulas that the automaton is trying to verify at position i. Thus the union of two transitions is a legal transition.

The acceptance condition requires that for each position in the word, if we start the automaton in each of the singleton sets corresponding to the members of the letter of that position, it accepts a finite subword. We will now prove that, given conditions (2), (3) and (4), it is equivalent to require that the automaton accept when started in the state identical to the letter of the node.

Theorem 2.9: *Let $A = (\Psi, \rho)$ be a set-subword automaton, let $w : \omega \to 2^{\Psi}$ be a word, and let $i \in \omega$. The following are equivalent:*

1. *There exists some $j \geq i$ and a mapping $\varphi : [i, j] \to \mathbf{S}$ such that $\varphi(i) = w_i$, $\varphi(j) = \emptyset$, and for all $i \leq k < j$, $\varphi(k+1) \in \rho(\varphi(k), w_k)$.*

2. *For every $f \in w_i$, there exists some $j_f \geq i$ and a mapping $\varphi_f : [i, j_f] \to \mathbf{S}$ such that $\varphi_f(i) = \{f\}$, $\varphi_f(j) = \emptyset$, and for all $i \leq k < j_f$, $\varphi_f(k+1) \in \rho(\varphi_f(k), w_k)$.*

Proof:
(1) $\Rightarrow$ (2). Let $f \in w_i$. We take $j_f = j$, and define φ_f as follows: $\varphi_f(i) = \{f\}$ and $\varphi_f(k) = \varphi(k)$ for $i < k \leq j_f$. We only have to show that $\varphi_f(k+1) \in \rho(\varphi_f(k), w_k)$ for $i \leq k < j_f$. But this follows, by the monotonicity condition (3) in the definition of set-subword automata, since $\varphi_f(i) \subseteq \varphi(i)$.
(2) $\Rightarrow$ (1). If $w_i = \emptyset$ take $j = i$, otherwise take $j = \max_{f \in w_i}\{j_f\}$. For every $f \in w_i$, extend φ_f to $[i, j]$ by defining $\varphi_f(k) = \emptyset$ for $j_f < k \leq j$. If $w_i = \emptyset$ we define $\varphi(i) = \emptyset$, otherwise we define $\varphi(k) = \bigcup_{f \in w_i} \varphi_f(y)$ for each $k \in [i, j]$. Because $j \geq j_f$ for each $f \in w_i$, $\varphi(j) = \emptyset$. Furthermore, by condition (4) in the definition of set-subword automata, if $i \leq k < j$, then $\varphi(k+1) \in \rho(\varphi(k), w_k)$. ∎

We can now prove that set-subword automata can be converted to subword automata without any increase in size. Thus, by Theorem 2.8, a set-subword automaton can be converted to an equivalent Büchi automaton with only a quadratic increase in size.

Theorem 2.10: *The set-subword automaton $A = (\Psi, \rho)$ is equivalent to the subword automaton $A' = (2^{\Psi}, 2^{\Psi}, \rho, \xi, \{\emptyset\})$, where ξ is the identity function.*

Proof: By Lemma 2.9, it is immediate that if a word w is accepted by the automaton A', then it is also accepted by A. Also by Lemma 2.9, if the word w is accepted by the set-subword automaton A, the subword condition of the subword automaton A' is satisfied. It remains to show that ξ satisfies the labeling condition. In other words, since ξ is the identity mapping, we have to show that for every $i \in \omega$, $w_{i+1} \in \rho(w_i, w_i)$. Since A accepts w, there exists some $j \geq i$ and a mapping $\varphi : [i, j] \to \mathbf{S}$ such that $\varphi(i) = w_i$,

$\varphi(j) = \emptyset$ and for all $i \leq k < j$, $\varphi(k+1) \in \rho(\varphi(k), w_k)$. If $j = i$, then $w_i = \emptyset$, and by the monotonicity conditions we have $w_{i+1} \in \rho(w_i, w_i)$. Otherwise, $i+1 \in [i,j]$, so $\varphi(i+1) \in \rho(w_i, w_i)$. Consider now $i+1$. If $\varphi(i+1) = \emptyset$, then clearly $\varphi(i+1) \subseteq w_{i+1}$. Otherwise, $\rho(\varphi(i+1), w_{i+1}) \neq \emptyset$, so, by condition (1) in the definition of set-subword automata, $\varphi(i+1) \subseteq w_{i+1}$. Thus by the monotonicity condition $w_{i+1} \in \rho(w_i, w_i)$. ∎

3 Temporal Logic with Automata Connectives

3.1 Definition

We consider propositional temporal logic where the temporal operators are defined by finite automata, similarly to the extended temporal logic (ETL) of [Wo83]. More precisely, we consider formulas built from a set $Prop$ of atomic propositions. The set of formulas is defined inductively as follows:

- Every proposition $p \in Prop$ is a formula.
- If f_1 and f_2 are formulas, then $\neg f_1$ and $f_1 \wedge f_2$ are formulas.
- For every nondeterministic finite automaton $A = (\Sigma, S, \rho, S_0, F)$, where Σ is the input alphabet $\{a_1, \ldots, a_n\}$, S is the set of states, $\rho : \Sigma \times S \to 2^S$ is the transition relation, $S_0 \subseteq S$ is the set of initial states, and $F \subseteq S$ is a set of accepting states, if $f_1, \ldots, f_n$ are formulas ($n = |\Sigma|$) then $A(f_1, \ldots, f_n)$ is a formula. We call A an *automaton connective*.

We assume some standard encoding for formulas. The *length* of a formula is the length of its encoding. This of course includes also the encoding of the automata connectives.

A structure for our logic is an infinite sequence of truth assignments, i.e., a function $\pi : \omega \to 2^{Prop}$ that assigns truth values to the atomic propositions in each state. Note that such a function π is an infinite word over the alphabet 2^{Prop}. We will thus use the terms word and sequence interchangeably. We now define *satisfaction* of formulas and *runs* of formulas $A(f_1, \ldots, f_n)$ over sequences by mutual induction. Satisfaction of a formula f at a position i in a structure π is denoted $\pi, i \models f$. We say that a sequence π satisfies f (denoted $\pi \models f$) if $\pi, 0 \models f$.

- $\pi, i \models p$ iff $p \in \pi(i)$, for an atomic proposition p.
- $\pi, i \models f_1 \wedge f_2$ iff $\pi, i \models f_1$ and $\pi, i \models f_2$.
- $\pi, i \models \neg f$ iff not $\pi, i \models f$.
- $\pi, i \models A(f_1, \ldots, f_n)$, where $A = (\Sigma, S, \rho, S_0, F)$, iff there is an accepting run $\sigma = s_0, s_1, \ldots$ of $A(f_1, \ldots, f_n)$ over π, starting at i.

- A run of a formula $A(f_1, \ldots, f_n)$, where $A = (\Sigma, S, \rho, S_0, F)$, over a structure π, starting at a point i, is a finite or infinite sequence $\sigma = s_0, s_1, \ldots$ of states from S, where $s_0 \in S_0$ and for all k, $0 \leq k < |\sigma|$, there is some $a_j \in \Sigma$ such that $\pi, i+k \models f_j$ and $s_{k+1} \in \rho(s_k, a_j)$.

Depending on how we define accepting runs, we get three different versions of the logic:

- ETL_f : A run σ is accepting iff some state $s \in F$ occurs in σ (finite acceptance)
- ETL_l : A run σ is accepting iff it is infinite (looping acceptance)
- ETL_r : A run σ is accepting iff some state $s \in F$ occurs infinitely often in σ (repeating or Büchi acceptance).

Every formula defines a set of sequences, namely, the set of sequences that satisfy it. We will say that a formula is *satisfiable* if this set is nonempty. The *satisfiability problem* is to determine, given a formula f, whether f is satisfiable. We are also interested in the expressive power of the logics we have defined. Our yardstick for measuring this power is the ability of the logics to define sets of sequences. Note that, by Corollary 2.2, ETL_r is at least as expressive as ETL_f and ETL_l. We can not, however, use Corollary 2.2, to infer that ETL_r is *more* expressive than ETL_f and ETL_l. Similarly, Corollary 2.2 does not give any information about the relative expressive power of ETL_f and ETL_l.

Example 3.1: Consider the automaton $A = (\Sigma, S, \rho, S_0, F)$, where $\Sigma = \{a, b\}$, $S = \{s_0, s_1\}$, $\rho(s_0, a) = \{s_0\}$, $\rho(s_0, b) = \{s_1\}$, $\rho(s_1, a) = \rho(s_1, b) = \emptyset$, $S_0 = \{s_0\}$ and $F = \{s_1\}$. If we consider finite acceptance, it accepts the language $a^\star b$. It thus defines an ETL_f connective such that $A(f_1, f_2)$ is true of a sequence iff f_1 is true until f_2 is true. Thus A_1 is equivalent to "Until" connective of [GPSS80]. It is indeed not hard to see that all the extended temporal logics (ETL_f, ETL_l, and ETL_r) are at least as expressive as PTL. ▮

Example 3.2: Consider the automaton $A = (\Sigma, S, \rho, S_0, F)$, where $\Sigma = \{a, b\}$, $S = \{s_0, s_1\}$, $\rho(s_0, a) = \{s_1\}$, $\rho(s_0, b) = \emptyset$, $\rho(s_1, a) = \emptyset$, $\rho(s_1, b) = \{s_0\}$, $S_0 = \{s_0\}$ and $F = \emptyset$. If we consider looping acceptance, it only accepts one word: $w = ababababab\ldots$. It thus defines an ETL_l connective such that $A_1(f_1, f_2)$ is true of a sequence iff f_1 is true in every even state and f_2 is true in every odd state of that sequence. It is shown in [Wo83] that this property is not expressible in PTL. ▮

3.2 Translations to Automata and Decision Procedure

As we have pointed out, the structures over which ETL is interpreted can be viewed as infinite words over the alphabet 2^{Prop}. It is not hard to show that our logics are translatable into S1S, and hence, by [Bu62] and [McN66], they define ω -regular sets of words. That is, given a formula in one of the logics ETL_f, ETL_l, or ETL_r, one can build a Büchi automaton that accepts exactly the words satisfying the formula. Nevertheless, since negation in front of automata connectives causes an exponential blow-up, we might have expected the complexity of the translation to be nonelementary, as is the translation from S1S to Büchi automata [Bu62]. Not so; for each of the logics we have defined, there is an exponential translation to Büchi automata. This translation also yields a PSPACE decision procedure for the satisfiability problem. In this section, we will give the translation for ETL_f and ETL_l. The translation for ETL_r is given in [SVW87]. We start with the translations for ETL_f and then outline the differences that occur when dealing with ETL_l.

We first need to define the notion of the closure of an ETL_f formula g, denoted $cl(g)$. It is similar in nature to the closure defined for PDL in [FL79]. From now on we identify a formula $\neg\neg g'$ with g'. Given an automaton $A = (\Sigma, S, \rho, S_0, F)$, for each $s \in S$ we define A_s to be the automaton $(\Sigma, S, \rho, \{s\}, F)$. The closure $cl(g)$ of an ETL_f formula g is then defined as follows:

- $g \in cl(g)$.
- $g_1 \wedge g_2 \in cl(g) \rightarrow g_1, g_2 \in cl(g)$.
- $\neg g_1 \in cl(g) \rightarrow g_1 \in cl(g)$.
- $g_1 \in cl(g) \rightarrow \neg g_1 \in cl(g)$.
- $A(g_1, \ldots, g_n) \in cl(g) \rightarrow g_1, \ldots, g_n \in cl(g)$.
- $A(g_1, \ldots, g_n) \in cl(g) \rightarrow A_s(g_1, \ldots, g_n) \in cl(g)$, for all $s \in S$.

Intuitively, the closure of g consists of all subformulas of g and their negations, assuming that $A_s(g_1, \ldots, g_n)$ is considered to be a subformula of $A(g_1, \ldots, g_n)$. Note that the cardinality of $cl(g)$ can easily be seen to be at most $2l$, where l is the length of g.

A structure $\pi : \omega \rightarrow 2^{Prop}$ for an ETL_f formula g can be extended to a sequence $\Pi : \omega \rightarrow 2^{cl(g)}$ in a natural way. With each point $i \in \omega$, we associate the formulas in $cl(f)$ that are satisfied at that point. Sequences on $2^{cl(g)}$ corresponding to models of a formula satisfy some special properties.

A *Hintikka sequence* for an ETL_f formula g is a sequence $\Pi : \omega \rightarrow 2^{cl(g)}$ that satisfies conditions 1–5 below. To state these conditions, we use a mapping that, for an automaton connective A, associates states of A to elements of $2^{cl(g)}$. Precisely, given $A(g_1, \ldots, g_n) \in cl(g)$ with $A = (\Sigma, S, \rho, S_0, F)$, we define a mapping $\sigma_A : 2^{cl(g)} \rightarrow 2^S$ such that

$$\sigma_A(\mathbf{a}) = \{s \in \rho(s_0, a_l) \ : \ s_0 \in S_0, a_l \in \Sigma, \text{ and } g_l \in \mathbf{a}\}.$$

In the following conditions, whenever $A(g_1, \ldots, g_n) \in cl(g)$ is mentioned, we assume that $A = (\Sigma, S, \rho, S_0, F)$. The conditions are then:

1. $g \in \Pi_0$,

   and for all $i \in \omega$

2. $h \in \Pi_i$ iff $\neg h \notin \Pi_i$,

3. $h \wedge h' \in \Pi_i$ iff $h \in \Pi_i$ and $h' \in \Pi_i$,

4. if $A(g_1, \ldots, g_n) \in \Pi_i$, then either $S_0 \cap F \neq \emptyset$ or there exists some $j > i$ and a mapping $\varphi : [i, j] \rightarrow 2^{cl(f)}$ such that:

   - $\varphi(k) \subseteq \Pi_k$ for $i \leq k \leq j$,
   - $A_{s_0}(g_1, \ldots, g_n) \in \varphi(i)$ for some $s_0 \in S_0$,
   - $\varphi(j) = \emptyset$,
   - for all k, $i \leq k < j$, if $A_s(g_1, \ldots, g_n) \in \varphi(k)$, then either
     - $s \in F$, or
     - there is some $t \in \sigma_{A_s}(\Pi_k)$ such that $A_t(g_1, \ldots, g_n) \in \varphi(k+1)$,

5. if $A(g_1, \ldots, g_n) \notin \Pi_i$, then:

   - $S_0 \cap F = \emptyset$ and
   - $A_s(g_1, \ldots, g_n) \notin \Pi_{i+1}$ for all $s \in \sigma_A(\Pi_i)$.

Hintikka conditions 2 and 3 are intended to capture the semantics of propositional connectives, while Hintikka conditions 4 and 5 are intended to capture the semantics of automata connectives. Note that for each propositional connective we need one condition using a double implication ("iff"), while we need two conditions for automata connectives due to the use of a single implication ("if ... then"). The reason for doing this is that Hintikka condition 5 is weaker than the converse of Hintikka condition 4. This makes it easier to construct an automaton that checks these conditions.

Proposition 3.3: *An ETL_f formula g has a model iff it has a Hintikka sequence.*

Proof: *Only if:* Let g be an ETL_f formula and let π be a model of g. We define a Hintikka sequence Π for g as follows: for all $i \geq 0$, $\Pi_i = \{h \in cl(g) \ : \ \pi, i \models h\}$. We now have to show that Π is indeed a Hintikka sequence. By the definition of a model, $\pi, 0 \models g$; this implies Hintikka condition 1. That Hintikka conditions 2, 3, 4, and 5 hold follows immediately from the semantic definition of ETL_f.

If: Let g be an ETL_f formula and let Π be a Hintikka sequence for g. Consider the structure π such that $\pi(i) = \{p \in Prop \ : \ p \in \Pi_i\}$. We now show that $\pi, 0 \models g$. For this, we show by induction that for all $g' \in cl(g)$ and $i \in \omega$, we have that $g' \in \Pi_i$ iff $\pi, i \models g'$. For the base case ($g' \in Prop$), this is immediate by construction. The inductive step for formulas of the form $\neg h$ and $h \wedge h'$ follows directly from the Hintikka conditions

2 and 3, respectively. It remains to prove the inductive step for formulas of the form $A(g_1, \ldots, g_n)$.

Suppose first that $A(g_1, \ldots, g_n) \in \Pi_i$. By Hintikka condition 4 and the inductive hypothesis, there is a finitely accepting run of A over π starting at i, so $\pi, i \models A(g_1, \ldots, g_n)$.

Suppose now that $A(g_1, \ldots, g_n) \notin \Pi_i$ but $\pi, i \models A(g_1, \ldots, g_n)$. Then there is a finitely accepting run of A over π starting at i. That is, there are finite sequences $s_0, \ldots, s_k$, and $j_0, \ldots, j_k$, $k \geq 0$, such that $s_0 \in S_0$, $s_k \in F$, and if $0 \leq l < k$, then $\pi, i + l \models g_{j_l}$ and $s_{l+1} \in \rho(s_l, a_{j_l})$. By Hintikka condition 5 and by induction on k it follows that $A_{s_l}(g_1, \ldots, g_n) \notin \Pi_{i+l}$ for $0 \leq l \leq k$. In particular, $A_{s_k}(g_1, \ldots, g_n) \notin \Pi_{i+k}$. But $s_k \in F$, so Hintikka condition 5 is violated. ▮

We now build a Büchi automaton over the alphabet $2^{cl(g)}$ that accepts precisely the Hintikka sequences for g. We do that by building two automata, A_L and A_E, such that $L(A_L) \cap L(A_E)$ is the set of Hintikka sequences for g. The first automaton A_L, called the *local* automaton, checks the sequence locally, i.e., it checks Hintikka conditions 1-3 and 5. This automaton is a Büchi automaton. The second automaton A_E, called the *eventuality* automaton, is a set-subword automaton that checks Hintikka condition 4. This automaton ensures that for all *eventualities* (i.e., formulas of the form $A(g_1, \ldots, g_n)$), there is some finite word for which condition Hintikka 4 is satisfied. Finally, we convert the *eventuality* automaton to a Büchi automaton and combine it with the local automaton.

The Local Automaton

The local automaton is $A_L = (2^{cl(g)}, 2^{cl(g)}, \rho_L, N_g, 2^{cl(g)})$. The state set is the collection of all sets of formulas in $cl(g)$.

For the transition relation ρ_L, we have that $\mathbf{s}' \in \rho_L(\mathbf{s}, \mathbf{a})$ iff $\mathbf{a} = \mathbf{s}$ and:

- $h \in \mathbf{s}$ iff $\neg h \notin \mathbf{s}$,
- $h \wedge h' \in \mathbf{s}$ iff $h \in \mathbf{s}$ and $h' \in \mathbf{s}$,
- if $\neg A(g_1, \ldots, g_n) \in \mathbf{s}$, the $S_0 \cap F = \emptyset$, and for all $s \in \sigma_A(\mathbf{a})$ we have that $\neg A_s(g_1, \ldots, g_n) \in \mathbf{s}'$.

The set of starting states N_g consists of all sets $\mathbf{s}$ such that $g \in \mathbf{s}$. Clearly, A_L accepts precisely the sequences that satisfy Hintikka conditions 1-3 and 5.

The Eventuality Automaton

The eventuality automaton is a set-subword automaton $A_E = (cl(g), \rho_E)$. For the transition relation ρ_E, we have that $\mathbf{s}' \in \rho_E(\mathbf{s}, \mathbf{a})$ iff:

- $\mathbf{s} \subseteq \mathbf{a}$, and
- If $A(g_1, \ldots, g_n) \in \mathbf{s}$, then, either $S_0 \cap F \neq \emptyset$ or there is a state $s \in \sigma_A(\mathbf{a})$ such that $A_s(g_1, \ldots, g_n) \in \mathbf{s}'$.

It is immediate to check that conditions (1-4) of the definition of set-subword automata are satisfied for A_E. Furthermore, A_E accepts precisely the sequences that satisfy Hintikka condition 4.

We now have:

Proposition 3.4: *Let g be an ETL_f formula and $\Pi : \omega \to 2^{cl(g)}$ be a sequence, then Π is a Hintikka sequence for f iff $\Pi \in L(A_L) \cap L(A_E)$.*

By Theorems 2.8 and 2.10, we can build a Büchi automaton that accepts $L(A_E)$. This automaton will have $2^{2cl(g)}$ states. Then by using Theorem 2.6, we can build a Büchi automaton accepting $L(A_L) \cap L(A_E)$. This automaton will have $2^{3cl(g)+1}$ states.[7]

The automaton we have constructed, accepts words over $2^{cl(g)}$. However, the models of f are defined by words over 2^{Prop}. So, the last step of our construction is to take the projection of our automaton on 2^{Prop}. This is done by mapping each element $b \in 2^{cl(g)}$ into the element $a \in 2^{Prop}$ such that $b \cap Prop = a$. Equivalently, the automaton runs over a word over 2^{Prop}, guesses a corresponding word over $2^{cl(g)}$, and verify that the latter is a Hintikka sequence for g. To summarize, we have the following:

Theorem 3.5: *Given an ETL_f formula g built from a set of atomic propositions $Prop$, one can construct a Büchi automaton A_g (over the alphabet 2^{Prop}), whose size is $2^{O(|g|)}$, that accepts exactly the sequences satisfying g.*

We can now give an algorithm and complexity bounds for the satisfiability problem for ETL_f. To test satisfiability of an ETL_f formula g, it is sufficient to test if $L(A_g) \neq \emptyset$. By Theorem 2.4, this can be done in nondeterministic logarithmic space in the size of A_g. Moreover, it is not hard to see that the construction of A_g satisfies the conditions of Lemma 2.5. Combining this with the fact that satisfiability for propositional temporal logic (a restriction of ETL_f) is PSPACE-hard [SC85], we have:

Theorem 3.6: *The satisfiability problem for ETL_f is logspace complete for PSPACE.*

Let us now consider ETL_l. The construction proceeds similarly to the one for ETL_f. The first difference is that Hintikka conditions 4 and 5 are replaced by the following:

4′) if $A(g_1, \ldots, g_n) \in \Pi_i$, then there is some $s \in \sigma_A(\Pi_i)$ such that $A_s(g_1, \ldots, g_n) \in \Pi_{i+1}$.

5′) if $\neg A(g_1, \ldots, g_n) \in \Pi_i$, then there exists some $j \geq i$ and a mapping $\varphi : [i, j] \to 2^{cl(g)}$ such that:

- $\varphi(k) \subseteq \Pi_k$ for $i \leq k \leq j$,

[7]This construction contains some redundancy, since the work done by the component that checks for the labeling condition (from the proof of Theorem 2.7) is subsumed by the work done by the local automaton. By eliminating this redundancy we can build an equivalent automaton with $2^{2cl(g)}$ states.

- $\neg A_{s_0}(g_1, \ldots, g_n) \in \varphi(i)$ for all $s_0 \in S_0$,
- $\varphi(j) = \emptyset$,
- for all k, $i \leq k < j$, if $\neg A_s(g_1, \ldots, g_n) \in \varphi(k)$, then $\neg A_t(g_1, \ldots, g_n) \in \varphi(k+1)$ for all $t \in \sigma_{A_s}(\Pi_k)$.

The analogue of Proposition 3.3 holds:

Theorem 3.7: *An ETL_l formula g has a model iff it has a Hintikka sequence.*

Proof: As in Proposition 3.3, one direction is immediate.

Let g be an ETL_l formula and let Π be a Hintikka sequence for g. Consider the structure π such that $\pi(i) = \{p \in Prop : p \in \Pi_i\}$. We show by induction that for all $g' \in cl(g)$ and $i \in \omega$, we have that $g' \in \Pi_i$ iff $\pi, i \models g'$. We show the inductive step that corresponds to automata connectives.

Suppose first that $A(g_1, \ldots, g_n) \in \Pi_i$. By Hintikka condition 4′, there are infinite sequences $s_0, s_1, \ldots$ and $j_0, j_1, \ldots$ such that $s_0 \in S_0$ and if $l \geq 0$ then $s_{l+1} \in \rho(s_l, a_{j_l})$ and $g_{j_l} \in \Pi_{i+l}$. By the induction hypothesis $\pi, i+l \models g_{j_l}$ for all $l \geq 0$, so $\pi, i \models A(g_1, \ldots, g_n)$.

Suppose now that $A(g_1, \ldots, g_n) \notin \Pi_i$. Then, by Hintikka condition 2, we have that $\neg A(g_1, \ldots, g_n) \in \Pi_i$. If $\pi, i \models A(g_1, \ldots, g_n)$, there is an infinite run $\sigma = s_0, s_1, \ldots$ of $A(g_1, \ldots, g_n)$ over π starting at i. More explicitly, we have that $s_0 \in S_0$ and for all $k \geq 0$ there is some $a_j \in \Sigma$ such that $\pi, i+k \models g_j$ and $s_{k+1} \in \rho(a_j, s_k)$. By the induction hypothesis, if $\pi, i+k \models g_j$, then $g_j \in \Pi_{i+k}$. But then, Hintikka condition 5′ cannot hold.
∎

We again construct an automaton to recognize Hintikka sequences in two parts: the local automaton and the eventuality automaton. This time the local automaton deals with Hintikka conditions 1–4 and the eventuality automaton deals with condition 5. The local and eventuality automata only differ from those for ETL_f by their transition functions. They are the following:

The Local Automaton

We have that $\mathbf{s}' \in \rho_L(\mathbf{s}, \mathbf{a})$ iff $\mathbf{a} = \mathbf{s}$ and:

- $h \in \mathbf{s}$ iff $\neg h \notin \mathbf{s}$,
- $h \wedge h' \in \mathbf{s}$ iff $h \in \mathbf{s}$ and $h' \in \mathbf{s}$,
- if $A(g_1, \ldots, g_n) \in \mathbf{s}$, the for some $s \in \sigma_A(\mathbf{s})$ we have that $A_s(g_1, \ldots, g_n) \in \mathbf{s}'$.

The Eventuality Automaton

We have that $\mathbf{s}' \in \rho_E(\mathbf{s}, \mathbf{a})$ iff:

- $\mathbf{s} \subseteq \mathbf{a}$, and

- If $\neg A(g_1, \ldots, g_n) \in \mathbf{s}$, then for all $s \in \sigma_A(\mathbf{a})$ we have that $\neg A_s(g_1, \ldots, g_n) \in \mathbf{s}'$.

We now have the analogues of Proposition 3.4 and Theorems 3.5 and 3.6:

Proposition 3.8: *Let g be an ETL_l formula and $\Pi : \omega \to 2^{cl(g)}$ be a sequence, then Π is a Hintikka sequence for g iff $\Pi \in L(A_L) \cap L(A_E)$.*

Theorem 3.9: *Given an ETL_l formula g built from a set of atomic propositions Prop, one can construct a Büchi automaton A_g (over the alphabet 2^{Prop}), whose size is $2^{O(|g|)}$, that accepts exactly the sequences satisfying f.*

Theorem 3.10: *The satisfiability problem for ETL_l is logspace complete for PSPACE.*

The construction described above simultaneously takes care of running automata in parallel, when we have an automata connective nested within another automata connective, and complementing automata, when we have a negated automata connective. Thus, the construction can be viewed as combining the classical subset construction of [RS59] and Choueka's "flag construction" in [Ch74]. This should be contrasted with the treatment of ETL_r in [SVW87], where a special construction is needed to complement Büchi automata. As a result, the size of the automaton A_g constructed in [SVW87] for an ETL_r formula g is $2^{O(|g|^2)}$. Using results by Safra [Sa88], this can be improved to $2^{O(|g| \log |g|)}$, which is provably optimal. Thus, while the construction given here for ETL_f and ETL_l as well as the construction given in [SVW87] for ETL_r are exponential, the exponent for ETL_f and ETL_l is linear, while it is nonlinear for ETL_r.

It is interesting the compare our technique here to Pratt's model construction technique [Pr79]. There one starts by building a maximal model, and then one eliminates states whose eventualities are not satisfied. Our local automata correspond to those maximal models. However, instead of eliminating states, we combine the local automata with the eventuality automata checking the satisfaction of eventualities. This construction always yields automata whose size is exponential in the size of the formula. We could also construct our automata using the *tableau* technique of [Pr80]. This technique can sometimes be more efficient than the maximal model technique.

A major feature of our framework here is the use of set-subword automata to check for eventualities. A direct construction of a Büchi automaton to check for eventualities would have to essentially use the constructions of Theorems 2.8 and 2.9. To appreciate the simplicity of our construction using set-subword automata, the reader is encouraged to try to directly construct a Büchi automaton that checks Hintikka condition 4 for ETL_f or checks Hintikka condition 5' for ETL_l. There are, however, other possible approaches to the translation of formulas to automata. Streett [St90] suggested using so-called *formula-checking automata*. His approach eliminates the need for distinction between the local automaton and the eventuality automaton. Another approach, using *weak alternating automata* is described in [MSS88]. In that approach not only there is no distinction between the local automaton and the eventuality automaton, but the automaton is constructed by a simple induction on the structure of the formula. We believe, however, that the distinction between the local automaton, which checks local properties, and the eventuality automaton that checks global properties is fundamental, even if it is avoidable for ETL_f and ETL_l. For example, for ETL_r or the temporal μ-calculus, the construction of the local automaton is straightforward, and the main difficulty lies in the construction of the "global" automaton [SVW87, Va88]. Unlike our approach, the "unitary" approaches of [MSS88, St90] do not generalize to these logics.

4 Translations Among the Logics

The results of the previous section show that the set of sequences describable by ETL_f, ETL_l or ETL_r formulas are expressible by Büchi automata. In the case of ETL_r, the converse is also clearly true. Thus, ETL_r has exactly the same expressive power as Büchi automata. Since, by Lemma 2.1, the notions of finite and looping acceptance are weaker than the notions of repeating acceptance, it would be conceivable for ETL_f and ETL_l to be less expressive than ETL_r and hence Büchi automata. We show that this is not the case as there is a translation of ETL_r formulas to ETL_f and ETL_l. As we will see, these translations involves an exponential increase in the length of the formula. Before giving these constructions, which are based on nontrivial automata-theoretic results from [Sa88], we show that there are straightforward translations between ETL_f and ETL_l (which also involves a single exponential increase in the length of the formulas).

We start by going from ETL_f to ETL_l.

Theorem 4.1: *Given an ETL_f formula g of length m, one can construct an ETL_l formula g' of length $2^{O(m)}$, that is satisfied by exactly the same sequences as g.*

Proof: To prove our theorem, we need to show how an ETL_f formula $A(f_1, \ldots, f_n)$, where A is defined by a nondeterministic finite acceptance automaton can be translated into ETL_l. The idea of this translation is that the sequences accepted by a finite acceptance automaton are those rejected by a closely related looping acceptance automaton, if the automaton is deterministic. To determinize finite acceptance automata, we can use the classical subset construction of [RS59].

The subset construction alone, however, does not assure sufficient determinism. What we want is that, given a structure, there is only one run of the deterministic automaton over that structure. Now, even though the automaton A is deterministic, a formula $A(f_1, \ldots, f_n)$ could be nondeterministic because states in a structure might satisfy more than one formula f_i. To overcome this type of nondeterminism, we replace $A = (\Sigma, S, \rho, S_0, F)$ by an automaton over 2^Σ. The automaton is $A' = (2^\Sigma, S, \rho', S_0, F)$, where the transition relation ρ' is defined as follows: $s_i \in \rho'(s_j, X)$ iff $s_i \in \rho(s_j, a)$ for some $a \in X$. Now, $A(f_1, \ldots, f_n)$ is equivalent to $A'(g_1, \ldots, g_{2^n})$, where the formula g_i corresponding to a set $X_i \subseteq \Sigma$ is $(\bigwedge_{a_j \in X_i} f_j) \wedge (\bigwedge_{a_j \notin X_i} \neg f_j)$. Clearly, in a given state exactly one g_i is satisfied.

Our next step, is to apply the subset construction to the automaton A'. This yields an automaton $A'' = (2^\Sigma, 2^S, \rho'', \{S_0\}, F'')$ where

- $\mathrm{s}' \in \rho''(\mathbf{a}, \mathrm{s})$ iff $\mathrm{s}' = \{s' \in S \mid \exists s \in \mathrm{s}$ such that $s' \in \rho(\mathbf{a}, s)\}$.
- $F'' = \{\mathrm{s} \in 2^S | \mathrm{s} \cap F \neq \emptyset\}$.

Now, for a given structure, there is a unique computation of $A''(g_1, \ldots, g_{2^n})$ on that structure and that computation is accepting iff it reaches a state in F''. In other words, the computation is nonaccepting if it is a looping computation of the automaton $A''' = (2^\Sigma, 2^S - F'', \rho'', \{S_0\}, -)$[8]. Hence our translation of $A(f_1, \ldots, f_n)$ into ETL_l is

$$\neg A'''(g_1, \ldots, g_{2^n}) \tag{1}$$

The size of (1) is clearly exponential in the size of $A(f_1, \ldots, f_n)$ and hence the translation of a formula of length m will be of length $2^{O(m)}$. ∎

We now turn to the translation from ETL_l to ETL_f.

Theorem 4.2: *Given an ETL_l formula g of length m, one can construct an ETL_f formula g' of length $2^{O(m)}$, that is satisfied by exactly the same sequences as g.*

Proof: The proof is identical to that of Theorem 4.1 except that this time the automaton A''' is the finite acceptance automaton defined by $A''' = (2^\Sigma, 2^S, \rho'', \{S_0\}, \emptyset)$. ∎

To translate from ETL_r to ETL_f and ETL_l, we will also use the determinization of the automaton defining a connective. However, here the automata are Büchi automata and they cannot be determinized by the subset construction. It is possible to build a deterministic finite-state automaton corresponding to a Büchi automaton, but this automaton will have to use a more general type of accepting condition than the infinite repetition of some state in a set F [McN66, Ch74]. Rather than using this more general type of automaton, we will rely on a construction that partially determinizes Büchi automata.

Lemma 4.3: *[Sa88]: Given a Büchi automaton $A = (\Sigma, S, \rho, S_0, F)$, where $|S| = m$, one can construct $k \leq m$ nondeterministic automata on finite words A_i each of size at most $O(m)$ and k deterministic Büchi automata B_i, each of size at most $2^{O(m)}$, such that the language $L(A)$ accepted by A satisfies $L(A) = \bigcup_{1 \leq i \leq k} L(A_i)L(B_i)$.*

We can now give our translations. We first deal with ETL_f.

Theorem 4.4: *Given an ETL_r formula g of length m, one can construct an ETL_f formula g' of length $2^{O(m)}$, that is satisfied by exactly the same sequences as f.*

[8] Remember that for looping automata the set of designated states is irrelevant.

Proof: For convenience in the proof, we temporarily use the following definition. Given a set of ETL_r formulas $\{f_1, \ldots, f_n\}$, a set of infinite words L over an alphabet $\Sigma = \{a_1, \ldots, a_n\}$ is satisfied by a sequence π iff there is a word $w_1 w_2 \ldots$ in L such that $\pi, i \models f_j$ if $w_i = a_j$. Note that this definition coincides with the definition of automata connectives when the language L is the language accepted by an automaton.

To prove our theorem, we need to show how an ETL_r formula $A(f_1, \ldots, f_n)$, where A is defined by a nondeterministic Büchi automaton can be translated into ETL_f. The main difficulty is to express in ETL_f the condition imposed by the repetition set of the nondeterministic Büchi automaton. By using Lemma 4.3, we will replace the nondeterministic Büchi automaton A by the deterministic automata B_i. We will then show how repetition for deterministic Büchi automata can be expressed in ETL_f.

However, similarly to the proof of Theorem 4.1, we will want the deterministic automata B_i to be deterministic over the structure. This implies that we have to ensure that only one argument of the automaton connective is true in each state of the structure. Thus, we first use the same construction as in Theorem 4.1 and replace the automaton $A = (\Sigma, S, \rho, S_0, F)$ by the automaton $A' = (2^\Sigma, S, \rho', S_0, F)$. The formula $A(f_1, \ldots, f_n)$ is then equivalent to $A'(g_1, \ldots, g_{2^n})$.

Now, we will use Lemma 4.3 on the automaton A'. Lemma 4.3 enables us to express the language accepted by A' as

$$L(A') = \bigcup_{1 \le i \le k} L(A'_i) L(B'_i)$$

Each of the terms of the union defines a language on infinite strings. Similarly to automata connectives, each of these languages is satisfied by a given set of sequences. Clearly, a sequence satisfies $A'(g_1, \ldots, g_{2^n})$ iff it satisfies one of the languages $L(A'_i)L(B'_i)$. Hence, if we build an ETL_f formula f_i corresponding to each of the terms of the union, the ETL_f formula corresponding to $A'(g_1, \ldots, g_{2^n})$ will be

$$\bigvee_{1 \le i \le k} f_i.$$

We now give the translation for each of the terms of the union. We start with the infinite parts of the term, B'_i.

Given a deterministic Büchi automaton $B'_i = (2^\Sigma, S'_i, \rho'_i, {S'}^0_i, F'_i)$, we want to construct an ETL_f formula f'_i such that a sequence satisfies f'_i iff it satisfies $B'_i(g_1, \ldots, g_{2^n})$. We use the fact that given a infinite sequence, there is exactly one run of $B'_i(g_1, \ldots, g_{2^n})$ over that sequence. That run is accepting if it goes through some state in F'_i infinitely often. Similarly, it is nonaccepting if after some point, it never goes through a state in F'_i. We will express the latter condition and then negate it.

To do this, we express that B'_i eventually reaches a state s after which it never goes through a state in F'_i. The sequences that cause B'_i never to go through a state in F'_i from the state s are those rejected by the finite-acceptance automaton $B'_{i,s} =$

$(2^\Sigma, S'_i, \rho'_i, \{s\}, F'_i)$. They are hence also those satisfying the formula $\varphi_s = \neg B'_{i,s}(g_1, \ldots, g_{2^n})$. To express that B'_i eventually reaches s and satisfies φ_s, we use the finite-acceptance automaton $B''_{i,s} = (2^\Sigma \cup \{a\}, S'_i \cup \{e\}, \rho''_{i,s}, S'^0_i, \{e\})$, where a is a new letter and e a new state. The transition relation $\rho''_{i,s}$ is ρ'_i extended with $\rho''_{i,s}(s,a) = \{e\}$. The formula $f''_{i,s} = B''_{i,s}(g_1, \ldots, g_{2^n}, \varphi_s)$ then expresses that the automaton B'_i eventually reaches state s and from then on never goes through any state in F'_i. Thus, to state that the computation of the automaton B'_i is accepting, we need to state for all states $s \in S'_i$, the formula $f''_{i,s}$ does not hold. In other words,

$$f'_i = \bigwedge_{s \in S'_i} \neg f''_{i,s}.$$

Now, to construct f_i, we need to express that there is some computation of A'_i after which f'_i holds. Le A'_i be the automaton $(2^\Sigma, T'_i, \delta'_i, T'^0_i, G'_i)$. We build the automaton $A''_i = (2^\Sigma \cup \{a\}, T'_i \cup \{e\}, \delta''_i, T'^0_i, \{e\})$ where a is a new letter, e is a new state and the transition relation δ''_i is δ'_i augmented with $\delta''_i(a,s) = \{e\}$ for $s \in G'_i$. We then finally have that

$$f_i = A''_i(g_1, \ldots, g_{2^n}, f'_i).$$

The size of each B'_i is exponential in the size of the original automaton A whereas the size of each A'_i is linear. The formula f_i contains at most one copy of A'_i and $2 \times |B'_i|$ copies of B'_i. It is thus exponential in the size of A. Hence the formula $\bigvee_{1 \le i \le k} f_i$ is also exponential in the size of $A(f_1, \ldots, f_n)$. Thus when translating an ETL_r formula of length l to ETL_f, the result will be of length $2^{O(m)}$. ∎

We now consider the translation to ETL_l. The result is similar to the one for ETL_f.

Theorem 4.5: *Given an ETL_r formula g of length m, one can construct an ETL_l formula f' of length $O(2^{O(m)})$, that is satisfied by exactly the same sequences as g.*

Proof: The proof is along the same lines as that of Theorem 4.4 However, we have to use looping acceptance automata instead of finite acceptance automata. We outline how this is done for each of the finite acceptance automata appearing in the proof of Theorem 4.4 The first such automaton is $B'_{i,s}$. We use instead the looping automaton

$$B'_{i,s,loop} = (2^\Sigma, S'_i - F'_i, \rho'_i, \{s\}, -).$$

That is, the automaton $B'_{i,s,loop}$ is the automaton $B'_{i,s}$ with the set of designated states removed and the transition relation updated accordingly. The formula φ_s then becomes $\varphi_s = B'_{i,s,loop}(g_1, \ldots, g_{2^n})$.

The second finite acceptance automaton used is $B''_{i,s}$ We replace it by

$$B''_{i,s,loop} = (2^\Sigma \times \{0,1\}, S'_i, \rho''_{i,s,loop}, S'^0_i, -),$$

where the transition relation $\rho''_{i,s,loop}$ is defined as follows:

- if $\sigma \neq s$, then for all letters $\alpha \in 2^\Sigma$, $\rho''_{i,s,loop}((\alpha,0),\sigma) = \rho''_{i,s,loop}((\alpha,1),\sigma) = \rho'_i(\alpha,\sigma)$.
- if $\sigma = s$, then for all letters $\alpha \in 2^\Sigma$, $\rho''_{i,s,loop}((\alpha,0),\sigma) = \rho'_i(\alpha,\sigma)$ and $\rho''_{i,s,loop}((\alpha,1),\sigma) = \emptyset$

We then have

$$f''_{i,s} = \neg B''_{i,s,loop}(g_1 \wedge \neg\varphi_s, \dots, g_{2^n} \wedge \neg\varphi_s, g_1 \wedge \varphi_s, \dots, g_{2^n} \wedge \varphi_s)$$

Intuitively, we have constructed the operator $B''_{i,s,loop}$ so that it is strictly deterministic and is true only if the formula φ_s is false each time B'_i goes through the state s. Thus its negation will be satisfied if at some point B'_i goes through the state s with φ_s true.

The last finite acceptance operator appearing in the proof of Theorem 4.4 is A''_1. As we have shown in Theorem 4.1, it is always possible to replace an ETL_f operator by an ETL_l operator at the cost of an exponential increase in size. This is what we do here. It does not affect the exponential overall complexity of our translation as the automaton A''_i is of size linear in the length of the original formula. ▮

5 Alternating Temporal Logic

The results of the preceding sections show that our technique is applicable to automata connectives and to the negation of automata connectives. This suggests that this technique can also deal with alternation.

Given a set S of states, let us denote by B_S the set of all Boolean formulas that use the states in S as variables. Members of B_S can be viewed as Boolean-valued functions on 2^S. Let $\varphi \in B_S$ and $S' \subseteq S$. Then $\varphi(S')$ is the Boolean value of φ when the states in S' are assigned 1 and the states in $S - S'$ are assigned 0. Formulas of the form s or $\neg s$, where $s \in S$, are called *atomic* formulas.

An *alternating finite automaton* [BL80, CKS81] (abbr. AFA) A is a tuple $A = (\Sigma, S, \rho, \varphi_0, F)$, where Σ is the input alphabet, S is the set of states $\{s_1, \dots, s_m\}$, $\rho : S \times \Sigma \to B_S$ is the transition function that associates with each state and letter a Boolean formula in B_S, $\varphi_0 \in B_S$ is the start formula, and $F \subseteq S$ is the set of accepting states. We can extend ρ to $B_S \times \Sigma$: $\rho(\varphi, a)$ is obtained by substituting $\rho(s_j, a)$ in φ for each s_j, $1 \leq j \leq m$. For example, if $\rho(s_1, a) = s_3 \vee s_4$ and $\rho(s_2, a) = s_3 \wedge \neg s_4$, then $\rho(\neg s_1 \vee s_2, a) = (\neg s_3 \wedge \neg s_4) \vee (s_3 \wedge \neg s_4)$. We define an auxiliary mapping $\alpha : 2^S \to 2^{B_S}$ by

$$\alpha(T) = \{s \ : \ s \in T\} \cup \{\neg s \ : \ s \in S - T\}.$$

We first consider acceptance of finite words by AFA. The run of A on a word $w = a_1, \dots, a_l$ is the sequence $\varphi_0, \dots, \varphi_l$ of formulas from B_S, where $\varphi_i = \rho(\varphi_{i-1}, a_i)$. A accepts w if $\varphi_l(F) = 1$. An equivalent way of defining acceptance of finite words by AFA is in terms of *finite run forests*. A finite run forest of A on w is a collection of finite trees labeled by atomic formulas satisfying the following conditions:

- The length of all branches is l.
- There is a set $S' \subseteq S$ such that $\varphi_0(S') = 1$ and for all $\varphi \in \alpha(S')$ there is a tree in the forest whose root is labeled by φ.
- Let x be an internal node of *depth* j, $0 \leq j < l$, labeled by φ_x. Then there is a set $S'' \subseteq S$ such that $\rho(\varphi_x, a_j)(S'') = 1$ and for all $\varphi \in \alpha(S'')$ there is a child y of x labeled by φ.

A node x labeled by φ_x is accepting if $\varphi_x(F) = 1$. The run forest is accepting if x is accepting whenever x is a leaf. A accepts w if it has an accepting run forest on w.

Afa's define regular languages. Nevertheless, it follows from the results in [Le81], [CKS81] that they can be exponentially more succinct than NFA's. That is, given any n-state AFA, one can construct an 2^n-state NFA that accepts the same language. Furthermore, for each n there is an n-states AFA A, such that the language defined by A is not definable by any NFA with less than 2^n states.

The notion of alternation can also be extended to automata on infinite words [MH84]. Finite acceptance is defined by means of finite acceptance forests. A finite run forest of A on an infinite word $w = a_1a_2\ldots$ is a collection of finite trees labeled by atomic formulas satisfying the following conditions:

- There is a set $S' \subseteq S$ such that $\varphi_0(S') = 1$ and for all $\varphi \in \alpha(S')$ there is a tree in the forest whose root is labeled by φ.
- Let x be an internal node of *depth* j labeled by φ_x. Then there is a set $S'' \subseteq S$ such that $\rho(\varphi_x, a_j)(S'') = 1$ and for all $\varphi \in \alpha(S'')$ there is a child y of x labeled by φ.

Again, the run forest is accepting if x is accepting whenever x is a leaf. A accepts w if it has an accepting run forest on w. Note that, as opposed to the definition used in the case of finite words, all branches of the run forest are not required to have the same length.

Looping acceptance is defined by means of infinite run forests. An infinite run forest of A on an infinite word $w = a_1a_2\ldots$ is a collection of infinite trees labeled by atomic formulas satisfying the following conditions:

- All branches are infinite.
- There is a set $S' \subseteq S$ such that $\varphi_0(S') = 1$ and for all $\varphi \in \alpha(S')$ there is a tree in the forest whose root is labeled by φ.
- Let x be an internal node of *depth* j labeled by φ_x. Then there is a set $S'' \subseteq S$ such that $\rho(\varphi_x, a_j)(S'') = 1$ and for all $\varphi \in \alpha(S'')$ there is a child y of x labeled by φ.

A accepts w if it has a run forest on w.

Repeating acceptance is also defined by means of infinite run forests. The condition is that along any branch of the run forest there are infinitely many accepting nodes.

As is the case with finite words, alternation does not add any expressive power beyond nondeterminism. More precisely, finite acceptance (resp., looping acceptance, repeating acceptance) AFA has the same expressive power as finite acceptance (resp., looping acceptance, repeating acceptance) NFA [MH84]. Thus the only gain in using alternation is in succinctness.

We first deal with the alternating analog of ETL_f. ATL_f is defined analogously to ETL_f, with AFA connectives replacing NFA connectives. For an AFA $A = (\Sigma, S, \rho, \varphi_0, F)$ and a formula $\varphi \in B_S$, we define A_φ to be the AFA $(\Sigma, S, \rho, \varphi, F)$. The semantics of ATL_f are defined as follows:

- $\pi, i \models p$ iff $p \in \pi(i)$, for an atomic proposition p.
- $\pi, i \models f_1 \wedge f_2$ iff $\pi, i \models f_1$ and $\pi, i \models f_2$.
- $\pi, i \models \neg f$ iff not $\pi, i \models f$.

To define the semantics of automata connectives we first adapt the definition of *finite run forests*. A finite run forest of $A(f_1, \ldots, f_n)$, where $A = (\Sigma, S, \rho, \varphi_0, F)$, on a structure π starting at i is a collection of finite trees labeled by atomic formulas satisfying the following conditions:

- There is a set $S' \subseteq S$ such that $\varphi_0(S') = 1$ and for all $\varphi \in \alpha(S')$ there is a tree in the forest whose root is labeled by φ.
- Let x be an internal node of *depth* j labeled by φ_x. Then there are an $a_l \in \Sigma$ and a set $S'' \subseteq S$ such that $\pi, i + j \models f_l$, $\rho(\varphi_x, a_l)(S'') = 1$ and for all $\varphi \in \alpha(S'')$ there is a child y of x labeled by φ.

The run forest is accepting if x is accepting whenever x is a leaf. A accepts w if it has an accepting run forest on w.

Notice the difference between the definitions of run forests for AFA and AFA formulas. In a run forest of an AFA all "spawned" automata read the same input. This is not the case in run forests of AFA formulas, since it is possible that more than one argument of the AFA connective holds at a point i of the structure π. Essentially, AFA read complete description of states (i.e., the input letter), while AFA formulas only read partial description of states (i.e., the input formula). We believe that this definition is the appropriate one in our context. Furthermore, as is shown in [HRV90], using the standard definition of run forests for AFA causes an exponential increase in the complexity of the logic.

We can now define satisfaction for automata connectives:

- $\pi, i \models A(f_1, \ldots, f_n)$ if and only if $A(f_1, \ldots, f_n)$ has an accepting run forest on π starting at i.

Clearly, ATL_f is at least as expressive as ETL_f. Indeed, if we restrict the formulas in B_S to be positive disjunctions then ATL_f reduces to ETL_f. Also, there is an exponential translation from ATL_f to ETL_f. The interest in this logic is twofold. First, since the translation from ATL_f to ETL_f causes a one exponential blow-up, one may expect ATL_f to be of higher complexity then ETL_f. Surprisingly, it has the same complexity. Also the study of AFA connectives reveals the full power of our techniques.

Theorem 5.1:

1. *Given an ATL_f formula g built from a set of atomic propositions Prop, one can construct a Büchi automaton A_g (over the alphabet 2^{Prop}), whose size is $2^{O(|g|)}$, that accepts exactly the sequences satisfying g.*

2. *The satisfiability problem for ATL_f is complete in PSPACE.*

Proof: We prove here the first claim, and the second claim follows as in Section 3. The proof parallels those given in Section 3. We are given a formula g, and we construct an automaton, which is the cross product of the local automaton and the eventuality automaton. The notion of closure is similar to that in Section 3. For an ATL_f formula g, $cl(g)$ is defined as follows:

- $g \in cl(g)$
- $g_1 \wedge g_2 \in cl(g) \rightarrow g_1, g_2 \in cl(g)$
- $\neg g_1 \in cl(g) \rightarrow g_1 \in cl(g)$
- $g_1 \in cl(g) \rightarrow \neg g_1 \in cl(g)$
- $A(g_1, \ldots, g_n) \in cl(g) \rightarrow g_1, \ldots, g_n \in cl(g)$
- $A(g_1, \ldots, g_n) \in cl(g) \rightarrow A_s(g_1, \ldots, g_n) \in cl(g)$, for all $s \in S$.
- $A(g_1, \ldots, g_n) \in cl(g) \rightarrow A_{\neg s}(g_1, \ldots, g_n) \in cl(g)$, for all $s \in S$.

Hintikka sequences for ATL_f formulas are defined similarly to those used in Section 3, except for conditions 4 and 5. We again need a technical definition to state these conditions. Given $A(g_1, \ldots, g_n) \in cl(g)$ with $A = (\Sigma, S, \rho, \varphi_0, F)$, we define a mapping $\sigma_A : 2^{cl(g)} \rightarrow 2^{2^S}$ such that

$$\sigma_A(\mathbf{a}) = \{T \ : \ a_l \in \Sigma, g_l \in \mathbf{a}, \text{ and } \rho(\varphi_0, a_l)(T) = 1\}.$$

In the following conditions, whenever $A(g_1, \ldots, g_n) \in cl(g)$ is mentioned, we assume that $A = (\Sigma, S, \rho, \varphi_0, F)$. The new conditions 4 and 5 are then:

4″) If $A(g_1, \ldots, g_n) \in \Pi_i$, then there exists some $j \geq i$ and a mapping $\psi : [i, j] \rightarrow 2^{cl(g)}$ such that:

- $\psi(j) = \emptyset$,
- there is a set $S' \subseteq S$ such that $\varphi_0(S') = 1$, and for all $\theta \in \alpha(S')$ we have $A_\theta(g_1, \ldots, g_n) \in \psi(i)$,
- for all k, $i \leq k < j$, if $A_\varphi(g_1, \ldots, g_n) \in \psi(k)$, then either
  - $\varphi(F) = 1$, or
  - there is some $S'' \in \sigma_A(\Pi_k)$ such that for all $\theta \in S''$ we have $A_\theta(g_1, \ldots, g_n) \in \psi(k+1)$.

5$''$) if $\neg A(g_1, \ldots, g_n) \in \Pi_i$, then $\varphi_0(F) = 0$ and for all $S' \in \sigma_A(\Pi_i)$ there exists some $\theta \in \alpha(S')$ such that $\neg A_\theta(g_1, \ldots, g_n) \in \Pi_{i+1}$.

The Local Automaton

The local automaton is $L = (2^{cl(g)}, 2^{cl(g)}, \rho_L, N_g, 2^{cl(g)})$. For the transition relation ρ_L, we have that $\mathbf{s}' \in \rho_L(\mathbf{s}, \mathbf{a})$ 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 $g_2 \in \mathbf{s}$,
- if $A(g_1, \ldots, g_n) \in \mathbf{s}$, then there is $S' \subseteq S$ such that $\varphi_0(S') = 1$ and for all $\theta \in S'$ we have $A_\theta(g_1, \ldots, g_n) \in \mathbf{s}$.
- if $\neg A(g_1, \ldots, g_n) \in \mathbf{s}$, then $\varphi_0(F) = 0$ and for all $S' \in \sigma_A(\mathbf{a})$ there exists some $\theta \in \alpha(S')$ such that $\neg A_{s'}(g_1, \ldots, g_n) \in \mathbf{s}'$.

Finally, the set of starting states N_g consists of all sets $\mathbf{s}$ such that $g \in \mathbf{s}$.

The Eventuality Automaton

The eventuality automaton is a set-subword automaton $A_E = (cl(g), \rho_E)$. For the transition relation ρ_E, we have that $\mathbf{s}' \in \rho_E(\mathbf{s}, \mathbf{a})$ iff:

- $\mathbf{s} \subseteq \mathbf{a}$, and
- If $A(g_1, \ldots, g_n) \in \mathbf{s}$, where φ_0 is atomic, then either $\varphi_0(F) = 1$ or there is some $S' \in \sigma_A(\mathbf{a})$ such that for all $\theta \in \alpha(S')$ we have $A_\theta(g_1, \ldots, g_n) \in \mathbf{s}'$.

▮

We now deal with the alternating analog of ETL_l. ATL_l is defined analogously to ETL_l, with AFA connectives replacing NFA connectives. We now define the semantics of automata connectives in ATL_l. We first need the following definition: An infinite run forest of $A(f_1, \ldots, f_n)$, where $A = (\Sigma, S, \rho, \varphi_0, F)$, on a structure π starting at i is a collection of infinite trees labeled by states of S or negation of states of S satisfying the following conditions:

- All branches are infinite.
- There is a set $S' \subseteq S$ such that $\varphi_0(S') = 1$ and for all $\varphi \in \alpha(S')$ there is a tree in the forest whose root is labeled by φ.
- Let x be a node of *depth* j labeled by φ_x. Then there are an $a_l \in \Sigma$ and a set $S'' \subseteq S$ such that $\pi, i+j \models f_l$, $\rho(\varphi_x, a_l)(S'') = 1$ and for all $\varphi \in \alpha(S'')$ there is a child y of x labeled by φ.

We can now define satisfaction for automata connectives in ATL_l.

- $\pi, i \models A(f_1, \ldots, f_n)$ if and only if $A(f_1, \ldots, f_n)$ has an infinite run forest on π starting at i.

Clearly, ATL_l is at least as expressive as ETL_l. Indeed, if we restrict the formulas in B_S to be positive disjunctions, then ATL_l reduces to ETL_l. Also, there is an exponential translation from ATL_f to ETL_f. The proof of the following theorem is similar to the proof of Theorem 5.1 and is left to the reader.

Theorem 5.2:

1. *Given an ATL_l formula g built from a set of atomic propositions Prop, one can construct a Büchi automaton A_g (over the alphabet) 2^{Prop}, whose size is $2^{O(|g|)}$, that accepts exactly the sequences satisfying g.*
2. *The satisfiability problem for ATL_l is complete in PSPACE.*

Finally, we describe the alternating analog of ETL_r. Recall that an accepting node in a run forest is a node x labeled by φ_x such that $\varphi_x(F) = 1$. In ATL_r, an infinite run forest is accepting if every branch has infinitely many accepting nodes. Satisfaction for automata connectives in ATL_r is defined by:

- $\pi, i \models A(f_1, \ldots, f_n)$ if and only if $A(f_1, \ldots, f_n)$ has an accepting infinite run forest on π starting at i.

In [MH84] it is shown that alternation can be eliminated from repeating acceptance automata by an exponential construction. We conjecture that this construction can be combined with the techniques of [SVW87] to prove the analogue of Theorems 5.1 and 5.2 (though with a nonlinear exponent in the exponential bound) for ATL_r.

6 Concluding Remarks

There are other approaches to extending PTL. In [HP85] the language is extended by regular operators corresponding to concatenation and the Kleene star. This, however, pushes the decision problem for their language to nonelementary complexity. Furthermore, it does not show the fine interplay between the different acceptance conditions that we have considered. In [Si83, SVW87], PTL is extended with quantifiers over propositions. For this extension, the complexity of the decision procedure is again nonelementary. The extension of PTL closest to the one discussed in this paper is probably that of the *temporal μ-calculus*, in which PTL is extended with fixpoint operators. It was introduced in [EC80]; see also [BKP85, BKP86]. It has the same expressive power as ETL and, as is shown in [Va88], also has a PSPACE-complete decision problem (see also [BB89]). It is interesting to note however that this last result was obtained using an extension of the automata-theoretic techniques presented in this paper.

The relation between the various types of acceptance criteria for finite automata can also be presented within the framework of propositional dynamic logic (PDL) [FL79]. Since we are reasoning here about computation paths, we consider PDL with one deterministic atomic program ($1DPDL$), where the structures are infinite sequences. In this framework, the finite acceptance condition corresponds to the *diamond* construct [FL79], the looping acceptance condition corresponds to the *loop* construct [HP79], and the repeating acceptance condition corresponds the *repeat* construct [HP79]. It follows that adding to $1DPDL$ looping, repeating, fixpoint, and even quantification over propositions does not change the expressive power of the language and does not render it undecidable. This in sharp contrast with what happens for propositional dynamic logic in general. It is known that PDL is less expressive than $PDL + loop$ (Pratt, see [HS82]), which is less expressive than $PDL + repeat$ [HS82], which is less expressive than $PDL + fixpoint$ [Ko83], which can be shown to be less expressive than $PDL + quantification$. Also, the last language can even be shown to be highly undecidable (Π^1_1- complete).

It should be noted that the automata-theoretic techniques developed here are also applicable to propositional dynamic logic; see [Va85a, VW86a]. Also, the temporal logics discussed here can be combined with dynamic logic to yield expressive process logics which are also amenable to automata-theoretic methods [VW83].

In conclusion, we note that our results have an interesting interpretation from a purely automata-theoretic point of view. The ability to have an automaton operator nested within another automaton operator is essentially equivalent to the ability of an automaton to consult an oracle. That is, when the automaton reaches certain states it asks the oracle whether it accepts the rest of the word, and its next move depends on the answer. Consider now the following hierarchy. Let A_0 be the class of nondeterministic finite automata, let A_i be the class of nondeterministic finite automata with oracles from A_{i-1}, and let A be $\cup_{i\geq 0}A_i$. If C is a class of automata, then $\overline{C}$ denotes the class of languages defined by automata in C.

Now we have three hierarchies A^f, A^l, and A^r, depending whether we use finite, looping, or repeating acceptance, respectively. Our results show that not only do these hierarchies collapse but that they are also equivalent: $\overline{A^f} = \overline{A^l} = \overline{A^r} = \overline{A_1^f} = \overline{A_2^l} = \overline{A_0^r}$. However, since each complementation causes at least an exponential blow-up, we might have expected the complexity of the emptiness problem for automata in A^f, A^l, and A^r to be nonelementary. Our results show that the problem is logspace complete for PSPACE for the three types of automata. This should be contrasted with the nonelementariness of the emptiness problem for regular expressions with complement [MS73]).

Acknowledgements.

We wish to thank R. Fagin, J. Halpern, R. Rosner, and L. Stockmeyer for helpful comments. We are also grateful to R. Streett and an anonymous referees for their careful reviews of the paper.

References

[AH90] R. Alur and T. Henzinger, "Real-time logics: complexity and expressiveness", *Proc. 5th IEEE Symp. on Logic in Computer Science*, 1990, pp. 390–401.

[BB89] B. Banieqbal and H. Barringer, "Temporal Logic with Fixed Points", in *Proc. Temporal Logic in Specification*, B. Banieqbal, H. Barringer, and A. Pnueli, eds., Lecture Notes in Computer Science 398, Springer-Verlag, 1989, pages 62–74.

[BKP85] H. Barringer, R. Kuiper, and A. Pnueli, "A Compositional Temporal Approach to a CSP-like Language", in *Formal Methods of Programming*, E. J. Neuhold and G. Chroust, eds., North Holland, 1985, pp. 207–227.

[BKP86] H. Barringer, R. Kuiper, and A. Pnueli, "A Really Abstract Concurrent Model and its Temporal Logic", *Proc. 13th ACM Symp. on Principles of Programming Languages*, St. Petersburgh, 1986, pp. 173–183.

[BBP89] B. Banieqbal, H. Barringer, and A. Pnueli, eds., "*Temporal Logic in Specification*", Lecture Notes in Computer Science 398, Springer-Verlag, 1989.

[BL80] J.A. Brzozowski and E. Leiss, "On Equations for Regular Languages, Finite Automata, and Sequential Networks", *Theoretical Computer Science* 10(1980), pp. 19–35.

[Bu62] J. R. Büchi, "On a Decision Method in Restricted Second Order Arithmetic", *Proc. Internat. Congr. Logic, Method. and Philos. Sci. 1960,* Stanford University Press, 1962, pp. 1–12.

[Ch74] Y. Choueka, “Theories of Automata on ω-Tapes: A Simplified Approach”, *J. Computer and System Sciences* 8(1974), pp. 117–141.

[CKS81] A.K. Chandra, D.C. Kozen, and L.J. Stockmeyer, “Alternation”, *J. ACM* 28(1981), pp. 114–133.

[EC80] E. A. Emerson and E. M. Clarke, “Characterizing Correctness Properties of Parallel Programs as Fixpoints”, *Proc. 7th Int. Colloquium on Automata, Languages and Programming,* Lecture Notes in Computer Science 85, Springer-Verlag, 1981, pp. 169–181.

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

[FL79] M. Fischer and R. Ladner, “Propositional Dynamic Logic of Regular Programs”, *J. Computer and System Sciences* 18(2), 1979, pp. 194–211.

[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.

[Ha79] D. Harel, *First Order Dynamic Logic*, Lecture Notes in Computer Science 68, Springer-Verlag, 1979

[Ha84] D. Harel, “Dynamic Logic”, in *Handbook of Philosophical Logic*, vol. 2, (D. Gabbay and F. Guenther, eds.), 1984, pp. 497–604.

[HP79] D. Harel and V. Pratt, “Nondeterminism in Logics of Programs”, *Proc. 5th ACM Symp. on Principles of Programming Languages,* Tuscon, 1978, pp. 203–213.

[HKP80] D. Harel, D. Kozen, and R. Parikh, “Process Logic: Expressiveness, Decidability, Completeness”, *J. Computer and System Science* 25(2), 1982, pp. 144–170

[HP85] D. Harel and D. Peleg, “Process logic with Regular Formulas”, *Theoretical Computer Science* 38(1985), pp. 307–322.

[HRV90] D. Harel, R. Rosner, and M.Y. Vardi, “On the Power of Bounded Concurrency III: Reasoning about Programs”, *Proc. 5th IEEE Symp. on Logic in Computer Science*, 1990, pp. 478-488.

[HR83] J. Y. Halpern and J. H. Reif, “The Propositional Dynamic Logic of Deterministic, Well-Structured Programs”, *Theoretical Computer Science* 27(1983), pp. 127–165.

[HS82] D. Harel and R. Sherman, "Looping vs. Repeating in Dynamic Logic", *Information and Control* 55(1982), pp. 175–192.

[HS84] D. Harel and R. Sherman, "Propositional Dynamic Logic of Flowcharts", *Information and Control* 64(1985), pp. 119–135.

[Jo75] N. D. Jones, "Space-bounded Reducibility among Combinatorial Problems", *J. Computer and System Science*, **11** (1975), pp. 68–75.

[Ka85] M. Kaminski, "A Classification of ω-Regular Languages", *Theoretical Computer Science* 36(1985), pp. 217–229.

[Ko83] D. Kozen, "Results on the Propositional μ-Calculus", *Theoretical Computer Science* 27(1983), pp. 333–354.

[Lad77] R. E. Ladner, "Application of Model-Theoretic Games to Discrete Linear Orders and Finite Automata", *Information and Control*, 33(1977), pp. 281–303.

[Lam77] L. Lamport, "Proving the Correctness of Multiprocess Programs", *IEEE Trans. on Soft. Eng.* SE-7(1977).

[Lan69] L. H. Landweber, "Decision Problems for ω-Automata", *Math. Systems Theory* 4(1969), pp. 376-384.

[Le81] E. Leiss, "Succinct Representation of Regular Languages by Boolean Automata", *Theoretical Computer Science* 13(1981), pp. 323–330.

[LPZ85] O. Lichtenstein, A. Pnueli, and L. Zuck, "The Glory of the Past", *Proc. Workshop on Logics of Programs*, Brooklyn, Springer-Verlag, Lecture Notes in Computer Science 193, 1985 pp. 196–218.

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

[Me75] A. R. Meyer, "Weak Monadic Second Order Theory of Successor is not Elementary Recursive", *Proc. Logic Colloquium,* Lecture Notes in Mathematics 453, Springer-Verlag, 1975, pp. 132–154.

[MH84] S. Miyano and T. Hayashi, "Alternating Automata on ω-Words", *Theoretical Computer Science* 32(1984), pp. 321-330.

[Mi80] R. Milner, "*A Calculus of Communicating Systems*", Lecture Notes in Computer Science 92, Springer-Verlag, 1980.

[MSS88] D.E. Muller, A. Saoudi, and P.E. Schupp, "Weak Alternating Automata Give a Simple Explanation of Why Most Temporal and Dynamic Logic are Decidable in Exponential Time", *Proc. 3rd IEEE Symp. on Logic in Computer Science*, 1988, pp. 422–427.

[MS73] A.R. Meyer and L.J. Stockmeyer, "Non-elementary Word Problems in Automata and Logic", *Proc. AMS Symp. on Complexity of Computation,* April 1973.

[MP89] Z. Manna and A. Pnueli, "The Anchored Version of the Temporal Framework", in *Linear Time, Branching Time, and Partial Order in Logics and Models for Concurrency* (J.W. de Bakker, W. P. de Roever, and G. Rozenberg, eds.), Lecture Notes in Computer Science 354, Springer-Verlag, 1989, pp. 201-284.

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

[Mu63] D.E. Muller, "Infinite Sequences and Finite Machines", *Proc. 4th IEEE Symp. on Switching Circuit Theory and Logical Design,* New York, 1963, pp. 3–16.

[MY88] T. Moriya and H. Yamusaki, "Accepting Conditions for Automata on ω-Languages", *Theoretical Computer Science* 61(1988) pp. 137–147.

[Ni80] H. Nishimura, "Descriptively Complete Process Logic", *Acta Informatica* 14(1980), pp. 359–369.

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

[Pr76] V.R. Pratt, "Semantical Considerations on Floyd-Hoare Logic", *Proceedings 17th IEEE Symposium on Foundations of Computer Science*, October 1976, pp. 109-121.

[Pr79] V.R. Pratt, "Models of program logics", *Proc. 20th IEEE Symp. on Foundation of Computer Science,* San Juan, 1979, pp. 115–122.

[Pr80] V.R. Pratt, "A Near-Optimal Method for Reasoning about Action", *J. Computer and System Sciences* 20(1980), pp. 231–254.

[Pr81] V.R. Pratt, "Using Graphs to Understand PDL", *Proc. Workshop on Logics of Programs,* (D. Kozen, ed.), Yorktown-Heights, Lecture Notes in Computer Science 131, Springer-Verlag, 1982, pp. 387-396.

[PR89] A. Pnueli and R. Rosner, "On the Synthesis of a Reactive Module", *Proc. 16th ACM Symp. on Principles of Programming Languages*, Austin, 1989, pp. 179–190.

[RS59] M. O. Rabin and D. Scott, "Finite Automata and their Decision Problems", *IBM Journal of Research and Development* 3(1959), pp. 114–125.

[Sa88] S. Safra, "On the Complexity of ω-Automata", *Proc. 29th IEEE Symp. on Foundation of Computer Science*, 1988, pp. 319–327.

[SC85] A. P. Sistla and E. M. Clarke, "The Complexity of Propositional Linear Temporal Logic", *J. ACM* 32(1985), pp. 733–749.

[Sh79] A.C. Shaw, "*Software Specification Languages Based on Regular Expressions*", Technical Report, ETH Zurich, June 1979.

[Si83] A. P. Sistla, "*Theoretical Issues in The Design and Analysis of Distributed Systems*", PhD Thesis, Harvard University, 1983.

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

[St90] R. Streett, personal communication.

[Sta87] L. Staiger, "Research in the Theory of ω-Languages", *Electron. Inf. Verarbeit. Kybernetic* 23(1987), pp. 415–439.

[SVW87] A.P. Sistla, M.Y. Vardi, and P. Wolper, "The Complementation Problem for Büchi Automata with Applications to Temporal Logic", *Theoretical Computer Science* 49(1987), pp. 217–237.

[TB73] B. A. Trakhtenbrot and Y. M. Barzdin, "*Finite Automata: Behavior and Synthesis*", North-Holland, 1973.

[Tho79] W. Thomas, "Star-Free Regular Sets of ω-Sequences", *Information and Control* 42(1979), pp. 148–156.

[Tho81] W. Thomas, "A Combinatorial Approach to the Theory of ω-Automata", *Information and Control* 48(1981), pp. 261–283.

[Tho90] W. Thomas, "Automata on Infinite Objects", in *Handbook of Theoretical Computer Science*, Vol. B. (J. v. Leeuwen, ed.), Elsevier, 1990, pp. 135–191.

[VW86a] M.Y. Vardi and P. Wolper. "Automata-Theoretic Techniques for Modal Logic of Programs", *J. Computer and System Sciences*, 32(1986), pp. 183–221.

[VW86b] M.Y. Vardi and P. Wolper, "An Automata-Theoretic Approach to Automatic Program Verification", *Proc. 1st IEEE Symp. on Logic in Computer Science*, Boston, 1986, pp. 332–334.

[Va85a] M.Y. Vardi, "The Taming of the Converse: Reasoning about 2-way Computations", *Proc. Workshop on Logics of Programs*, Brooklyn, Springer-Verlag Lecture Notes in Computer Science 193, 1985, pp. 413–424.

[Va85b] M.Y. Vardi, "Automatic verification of probabilistic concurrent finite-state programs", *Proc. 26th IEEE Symp. on Foundations of Computer Science*, Portland, Oct. 1985, pp. 327-338.

[Va88] M. Y. Vardi, "A Temporal Fixpoint Calculus", *Proc. 15th ACM Symp. on Principles of Programming Languages*, San Diego, 1988, pp. 250–259.

[VW83] M. Y. Vardi and P. Wolper, "Yet Another Process Logic", *Proc. Workshop Logic of Programs*, Lecture Notes in Computer Science 164, Springer-Verlag, 1983, pp. 501–512.

[Wa79] K. Wagner, "On ω-Regular Sets", *Information and Control* 43(1979), pp. 123–177.

[Wo83] P. Wolper, "Temporal Logic Can Be More Expressive", *Information and Control* 56(1983), pp. 72–99.

[Wo82] P. Wolper, "*Synthesis of Communicating Processes from Temporal Logic Specifications*", Ph. D. Thesis, Stanford University, 1982.

[Wo89] P. Wolper, "On the relation of programs and computations to models of temporal logic", in *Proc. Temporal Logic in Specification*, B. Banieqbal, H. Barringer, and A. Pnueli, eds., Lecture Notes in Computer Science 398, Springer-Verlag, 1989, pages 75–123.