An Automata Theoretic Approach to Temporal Logic

Gjalt G. de Jong *
Eindhoven University of Technology
Department of Electrical Engineering
P.O. Box 513, 5600 MB Eindhoven, The Netherlands
Tel. +31 40 473345, Fax: +31 40 464527
Email: gjalt@es.ele.tue.nl

Abstract

A syntax directed mapping is presented from Propositional Temporal Logic (PTL) formulae to Müller type finite automata. This is a direct and much more elegant and easier to implement approach than previously described methods. Most of these methods are based on tableau methods for satisfiability checking, and after that a Büchi type of automaton is extracted. Büchi and Müller automata are equally expressive. However, Müller automata have nicer properties than Büchi automata, for instance deterministic Müller automata are expressive as non-deterministic ones, while this is not true for Büchi automata. Also deterministic Büchi automata are not closed under complement. This transformation is the first step in a decision procedure, since the resulting Müller automaton represents the models of the temporal logic formula, and on which further verification and analysis can be performed.

1. Introduction

Temporal logic has proven to be a well suited formalism for program verification [1] as well as hardware specification and verification [2, 3]. The theory of finite automata is also very well-known and a very suitable formalism to describe and analyze systems in. Since the semantics of temporal logic is based on a state transition graph, these two formalisms can be linked together. It is shown in [4, 5] that propositional temporal logic is contained in the class of ω-regular expressions. This is based on a tableau kind decision procedure from which then a Büchi automaton is extracted. From then on, papers have appeared to show how certain properties, described as temporal logic formulae, can be stated as Büchi automata, for instance [6, 7]. However, transformations from temporal logic formulae

* This research is part of the ASCIS project sponsored by the European Community under contract BRA 3281.

onto Büchi automata all suffer from the drawback that deterministic Büchi automata are not closed under complementation [8], which then gives a burden on the complexity [9].

This approach gives us alternative ways to check satisfiability and tautology, but also allows us to mix freely temporal logic and state-transition based formalisms for specification, verification [3] as well as manipulation.

This paper is organized as follows. First we will discuss temporal logic and give its semantics. In the next chapter we will give a short overview of finite automata, and two types of accepting conditions for automata accepting languages of infinitary-length words: Büchi and Müller acceptance condition. After that, we will present a syntax directed transformation of temporal logic formulae onto Müller automata, which describe the models of the temporal logic formulae.

2. Temporal Logic

Temporal logic is the natural extension of propositional logic to include time related information. Whereas propositional logic can be said to express facts to be true or false at a certain moment, or state, temporal logic allows one to express the relation between such facts at several moments or states. Each state then announces particular atomic propositions to be true and others to be false. The language of propositional temporal logic (PTL) may be regarded as a superset of the language of propositional logic adding the operators $\Box$ (*always*), $\Diamond$ (*sometime*), U (*until*) and $\bigcirc$ (*next*). $\bigcirc f$ intuitively means that f is true in the **next** state; $\Box f$ that f is true in **all** future states; $\Diamond f$ that f is true in **some** future state; fUg that f is true for all states until g becomes valid in a state.

Formally, the semantics of a temporal logic formula (with a set P of propositional variables) is defined with respect to a triple $M = (S, N, p_i)$, where S is a finite set of states, $N : S \rightarrow S$ a total successor function giving for each state a unique next state and $p_i : S \times P \rightarrow \{True, False\}$ a truth-assignment giving a truth value to each propositional variable in each state.

The truth of a PTL formula is inductively defined relative to a structure M and a state s by:

$$<M, s> \models p \text{ iff } p_i(s,p) = True, \; p \in P$$

$$<M, s> \models \neg f \text{ iff not } <M, s> \models f$$

$$<M, s> \models f \vee g \text{ iff } <M, s> \models f \text{ or } <M, s> \models g$$

$$<M, s> \models f \wedge g \text{ iff } <M, s> \models f \text{ and } <M, s> \models g$$

$$<M, s> \models \bigcirc f \text{ iff } <M, N(s)> \models f \qquad (1)$$

$<M, s> \models \Diamond f$ *iff* $\exists_{i \geq 0} <M, N^i(s)> \models f$

$<M, s> \models \Box f$ *iff* $\forall_{i \geq 0} <M, N^i(s)> \models f$

$<M, s> \models fUg$ *iff* $\exists_{i \geq 0} (<M, N^i(s)> \models g$ *and* $\forall_{0 \leq j < i} <M, N^j(s)> \models f)$

where $N^i(s)$ *denotes the* i^{th} *successor of* s.

An interpretation or model for a PTL formula consists of a structure M with a designated set of states S, truth-assignment p_i, successor function N and an initial state s_0.

Because the set of states S is finite and the successor relation is a total function, any infinite sequence of occurrences of states, may be represented in a finite way by a ω-regular string over the alphabet S, i.e. it consists of a certain possible empty prefix sequence followed by an endless repetition of a cycle of 1 or more states.

A PTL formula is satisfiable, i.e. can be made true, if we can find a model $<M, s_0>$ such that $<M, s_0> \models f$ holds. If a formula is true in a model we also say that the model, or sequence of states with associated truth-assignment satisfies the formula.

A formula is said to be valid, or is a tautology, iff it is true in every appropriate model, notation: $\models f$. A formula that cannot be satisfied by any model is a contradiction. Two formulas f and g are said to be equivalent, notated $f = g$, when $\models (f \leftrightarrow g)$ holds. Note that a formula is unsatisfiable if and only if its negation is a tautology and conversely a formula is valid iff its negation is unsatisfiable.

3. Finite Automata

A finite automaton M is a five tuple $(\Sigma, Q, \delta, I, F)$, where Σ is an alphabet of symbols, Q a finite set of states, δ a mapping $Q \times \Sigma \rightarrow 2^Q$ that represent the state transitions labeled by a symbol, $I \subset Q$ a set of initial states and $F \subset Q$ a set of final states. An automaton M accepts a word $w = \sigma_1 .. \sigma_n$ ($\in \Sigma^*$) whenever there exists a sequence $q_0 \ldots q_n$ ($q_i \in Q$) such that $q_0 \in I$, $q_i \in \delta(q_{i-1}, \sigma_i)$ and $q_n \in F$. The language that an automaton M accepts is the set of all words that are accepted by the automaton. The class of languages which can be accepted by finite automata is the class of regular languages.

δ can be extended in the natural way to a mapping of $Q \times \Sigma^* \rightarrow 2^Q$. The symbol $\varepsilon \notin \Sigma$ denotes the empty word. An automaton M is called deterministic, when it has only one initial state, and δ is a mapping $Q \times \Sigma \rightarrow Q$. Non-deterministic automata can be converted to deterministic automata by the subset method [10].

Finite automata can also be used to accept the class of ω-regular languages, which is a class of languages consisting of infinitary-length words. A language is ω-regular if it can be written as UV^ω where U and V are regular languages. Two major types of finite

automata exist which are equally expressive, both able to recognize the class of ω-regular languages: Müller and Büchi automata. These automata differ only in their accepting condition.

A Büchi automaton is a five tuple $(\Sigma, Q, \delta, I, F)$ as above. However, an infinitary length word w is accepted by a Büchi automaton if $INF(w) \cap F \neq \emptyset$, where $INF(w)$ is defined as the set of states that are 'visited' infinitely many times.

A Müller automaton is a five tuple $(\Sigma, Q, \delta, I, A)$ with Σ, Q, δ and I as before. But the accepting condition is defined by $A \subset 2^Q$. A word w is accepted by a Müller automaton if $INF(w) \in A$. More intuitively: if in the long run, a word stays in a particular subset of the state space.

The class of Müller and Büchi automata is closed under union, intersection. Both types of automata are also closed under the operation of prefixing with normal finite automata. Only deterministic Müller automata are closed under complementation. These operations are defined for Müller automata as: (where $fa_i = (\Sigma, Q_i, \delta_i, I_i, A_i)$ and *dfa* is a deterministic automaton.)

$$Union(fa_1, fa_2) = (\Sigma, Q_1 \cup Q_2, \delta_1 \cup \delta_2, I_1 \cup I_2, A_1 \cup A_2)$$

$$Complementation(dfa) = (\Sigma, Q, \delta, I, 2^Q - A)$$

By DeMorgan, intersection, or product, is then also defined. But this may also be written as:

$Intersection(fa_1, fa_2) = (\Sigma, Q_1 \times Q_2, \delta, I_1 \times I_2, A_1 \times A_2)$
where $(q_i \times q_{i'}, \sigma, q_j \times q_{j'}) \in \delta$ if $(q_i, \sigma, q_j) \in \delta_1$ and $(q_{i'}, \sigma, q_{j'}) \in \delta_2$.

$Concatenation(fa_1, fa_2) = (\Sigma, Q_1 \cup Q_2, \delta_1 \cup \delta_2 \cup \delta_{fi}, I_1, A_2)$,
where fa_1 is a normal finite automaton $(\Sigma, Q_1, \delta_1, F)$, and $\delta_{fi} = \{(q_{f_1}, \varepsilon, q_{i_j}) \mid q_{fi} \in F_1 \wedge q_{i_j} \in I_2\}$

These operations are similar in the case of Büchi automata, except for the complement operation.

Determinization of Müller automata can also be done by the subset method, however it is then necessary that there does **not** exist a path going out of an accepting component. So it may be necessary to duplicate state sets, and its transitions. Duplication of an accepting set $A_k \in A$ is defined as:

$$M' = (\Sigma, Q', \delta', I, A') \tag{2}$$
where
$Q' = Q \cup \{q'_i \mid q_i \in A_k\}$
$\delta' = \delta \cup \{(q_i, \sigma, q'_j) \mid (q_i, \sigma, q_j) \in \delta \wedge q_i \in A_k \wedge q_j \in A_k\}$

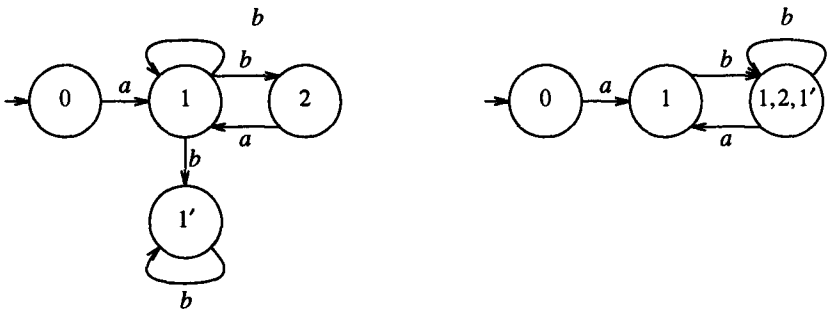
$$\cup \{(q'_i, \sigma, q'_j) | (q_i, \sigma, q_j) \in \delta \wedge q_i \in A_k \wedge q_j \in A_k\}$$
$$A' = \{A'_k\}, A'_k = \{q'_i | q_i \in A_k\}$$

The accepting sets of the deterministic automaton after the subset method are:
$A' = \{A_k\}, A_k = \{q \in P(Q) | \exists A_k \in A : q \subset A_k \neq \emptyset\}$ (3)
Note that only those sets that are strongly connected contribute to accepting paths.

In Fig. 1 and Fig. 2 it is illustrated that duplication of states is indeed necessary.

$A = \{\{1\}\}$ $A = \{\{<1>,<1,2>\}\}$

Figure 1. Incorrect determinization with the subset method



$A = \{\{1'\}\}$ $A = \{\{<1,2,1'>\}\}$

Figure 2. Determinization with the subset method after duplication of accepting sets

Theorem 1: Duplication of accepting sets of a Müller automaton $M = (\Sigma, Q, \delta, I, A)$ automaton results in an equivalent Müller automaton M' according to (2).

Proof: Since Müller automata are closed under union, let, without loss of generality, $A = \{a\}$. Let ρ be a run for an ω-word accepted by M. Since there are no transitions removed, every path in M also exists in M'. At some instant, ρ on M enters a (at state q_0) and from then on it will stay in a. The corresponding ρ' on M' also enters a (at the same state q_0) on M'. The next move on M will be from one state in a to a next state q_1 in a. M' can make the corresponding move to state q'_1 in a', because for every move from a state $q_0 \in a$ towards $q_1 \in a$ a transition is made from $q_0 \in a$ towards $q_1 \in a'$. All subsequent moves on M within a have corresponding moves on M' within a'. Thus $L(M) \subset L(M')$.

The converse is analogous. Let ρ' be a run for an ω-word accepted by M'. At some time, ρ' moves on M' from a state $q_0 \notin a'$ to a state $q_1 \in a'$. Because every transition towards a state $q' \in a'$ on M' corresponds to a transition towards $q \in a$ on M, it is clear, that M can enter its accepting set too. The only possible subsequent transitions on M' are all within a' and because of the duplication, all these transitions have corresponding transitions within a on M. Thus $L(M') \subset L(M)$.

From Fig. 1 and Fig. 2 it is clear that the subset method does not result in a correct automaton when transitions exist between states of accepting sets as defined in (3) and there does not exist a corresponding transition, i.e. with the same label, within the original accepting set. In the following it is proven that such transitions do not exist in the deterministic automaton when all the accepting sets are duplicated according to (2).

Theorem 2: The subset method applied to a nondeterministic Müller automaton $M = (\Sigma, Q, \delta, I, A)$ with each accepting state set A_k duplicated according to (2) yields an equivalent Müller automaton M' with A' according to (3).

Proof: We only need to show that no illegal transition exist, i.e. that no transitions exist between accepting states if does not exist an corresponding transition within the accepting set of the NFA after duplication.
Because Müller automata are closed under union, let, without loss of generality, $M = (\Sigma, Q, \delta, I, A)$ be the NFA, where A is the singleton $\{a\}$ which is the result of duplication.
Let $M' = (\Sigma, Q', \delta', I', A')$ the DFA, constructed via the subset-construction out of M. Now take any $(Q_1, \sigma, Q_2) \in \delta'$, with $Q_1 \in a'$ and $Q_2 \in a'$. Q_1 is the 'super'state of M' consisting of states in Q, which are all reachable in M from some state, by the same word. If $Q_1 \cap a = \emptyset$, then $Q_1 \notin A'$ and this implies a contradiction. Thus it follows $Q_i \cap a \neq \emptyset$. If $\exists q_0 \in Q_1 \cap a : \delta(q_0, \sigma) \neq \emptyset$ then of course $Q_2 \in A'$ and the transition is legal, because it has its corresponding transition within the accepting set of M. Else $\forall q_0 \in Q_1 \cap a : \delta(q_0, \sigma) = \emptyset$. In that case, if $\forall q \in Q_1 : \delta(q, \sigma) \cap a = \emptyset$, then $Q_2 \notin A'$, again a contradiction. Now take $q_1 \in Q_1$ and $q_1 \notin a$ and $(q_1, \sigma, q_2) \in \delta$, with $q_2 \in a$. This implies, that this transition will occur in the accepting set a' of the DFA M', while it was not in the accepting set a of the NFA M. However, the duplication ensures, that the only transitions pointing towards a state $q_2 \in a$ must come from a state $q_1 \in a$ (contradiction) or else, an equivalent transition will exist within the accepting set, i.e. $\exists q_3 \in a : (q_3, \sigma, q_2) \in \delta$ and this q_3 exists because it is the duplication of q_1. It also follows from the duplication, that q_1 and q_3 are reachable from the initial states of M, by the same word. That however would finally imply, that there is a corresponding transition for (Q_1, σ, Q_2) within the original accepting set a. This completes the proof, that there can be no 'new' transition introduced in the accepting set of the DFA, when the duplication method is used, before the subset-construction.

It is proven that Büchi and Müller automata are equally expressive, since they both define the class of ω-regular languages [11]. It is also proven that deterministic and non-

deterministic Müller automata are equally expressive, while deterministic Büchi automata are less expressive [8]. Since also the complementation operation on Müller automata is much more efficient that on Büchi automata [9], we find the type Müller automata more convenient for our purposes.

4. Temporal Logic to Müller automata

For temporal logic, the above described types of automata are extended to be able to model propositional logic formulae as the labels on the transitions, instead of symbols out of an alphabet Σ. So a Müller type of automaton is then described by the five tuple: (P, Q, δ, I, A) with P the set of propositional variables, and δ is a mapping $Q \times PL \rightarrow 2^Q$ where PL is a propositional logic formula which can be seen as element of 2^P. All previous defined operations can be extended in the natural way on this type of automaton. Note that in fact a boolean algebra on P is defined of which the conventional type of transitions is a special case.

In fact, such an automaton can be seen as a compact representation of all the models $<M, s>$ with respect to which a PTL formula f is defined. Only the states and transitions have changed roles, because in a model for a PTL formula f each state has a set of propositions which are valid in that state. In our case, these sets of valid propositions are on the transitions. But these two types of state transition graphs are each others dual and can therefore be transformed into each other.

Now a transformation of a PTL formula f to a Müller automaton M is given, for which the set of all *accepting* paths t are all the models m of f where

$$M = (P, Q, \delta, I, A) \tag{4}$$
$$t = <s_0, s_1, \ldots> \text{ with } s_0 \in I \text{ and } (s_i, p, s_{i+1}) \in \delta$$
$$m = <s'_0, s'_1, \ldots> \text{ with } p_i(s_i) = p, (s_i, p, s_{i+1}) \in \delta$$

This transformation $FA{:}PTL \rightarrow MFA$, where MFA is the type of Müller automata, is defined inductively as:

— Case $f = p \in P$:
$FA(f) = (P, \{q_0, q_1\}, \{(q_0, p, q_1)(q_1, True, q_1)\}, \{q_0\}, \{\{q_1\}\})$

— Case $f = \neg f_1$:
$FA(f) = Complement(FA(f_1))$

— Case $f = f_1 \wedge f_2$:
$FA(f) = Intersection(FA(f_1), FA(f_2))$

— Case $f = f_1 \vee f_2$:
$FA(f) = Union(FA(f_1), FA(f_2))$

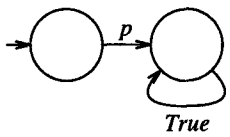


Figure 3. Automaton of a propositional formula p

— Case $f = \bigcirc f_1$:
$FA(f) = Concatenation(FA_{\bigcirc}, FA(f_1))$
where
$FA_{\bigcirc} = (P, \{q_0, q_1\}, \{(q_0, True, q_1)\}, \{q_0\}, \{q_1\})$
This is illustrated in Fig. 4 .

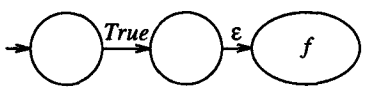


Figure 4. Automaton of $\bigcirc f$

— Case $f = \langle\rangle f_1$:
$FA(f) = Concatenation(FA_{\langle\rangle}, FA(f_1))$
where
$FA_{\langle\rangle} = (P, \{q_0\}, \{(q_0, True, q_0)\}, \{q_0\}, \{q_0\})$
This is illustrated in Fig. 5 .

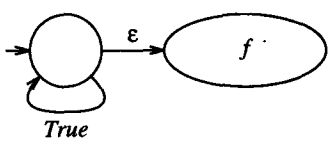


Figure 5. Automaton of $\langle\rangle f$

— Case $f = \Box f_1$:
Since $\Box f = \neg\langle\rangle \neg f$, the automaton for $\Box f$ can be constructed with the previous operations.

— Case $f = f_1 \; U \; f_2$:
Since $f \; U \; g = \langle\rangle g \wedge (g \vee (f \wedge (f \; U \; g)))$, the automaton for $f \; U \; g$ can be constructed as the automaton for $\langle\rangle g \wedge \Box(X = g \vee (f \wedge \bigcirc X))$, so by introducing an auxiliary variable which can be hided later.

Now we proof that the above transformation results in a Müller automaton M for a PTL formula f which accepts the same language as the PTL formula f, or equivalently represents all models of f as defined in (4). The proof is based on language equivalence. So we first define the following two functions L and M:

$$L{:}MFA \rightarrow R^{\omega} \tag{5}$$

where R^{ω} is the type of ω-regular languages. The alphabet may be seen as the set of propositional variables P which have the value *True* assigned. This function is the classical mapping of automata to (ω-)regular languages.

$$M{:}PTL \rightarrow R^{\omega} \tag{6}$$

M is the mapping of models to ω-regular strings as defined in section 2.

Theorem 3: $L(FA(f)) = M(f)$, or equivalently: the models of f are just the accepting paths of M as defined in (4).

Proof: The proof is by induction on the length of PTL formula f.

— Case $f = p \in P$:
The models for this formula are the sequences $<s_0, s_1, s_2, ...>$ in which $p_i(s_0) = p$ and $p_i(s_i) = True$ for all $i>0$. It is trivial to check that this is equivalent with $FA(f)$.

— Case $f = \neg f_1$:
According to (1), $M(f) = \Sigma^{\omega} - M(f_1)$. By induction, $L(FA(f_1)) = M(f_1)$. Also $L(FA(f)) = L(FA(\neg f_1)) = \Sigma^{\omega} - L(FA(f_1))$. So $L(FA(f)) = M(f)$.

— Case $f = f_1 \vee f_2$:
According to (1), $M(f) = M(f_1) \cup M(f_2)$. By induction, $L(FA(f_i)) = M(f_i)$. Also $L(FA(f)) = L(FA(f_1 \vee f_2)) = L(FA(f_1)) \cup L(FA(f_2))$. So $L(FA(f)) = M(f)$.

— Case $f = f_1 \wedge f_2$:
According to (1), $M(f) = M(f_1) \cap M(f_2)$. By induction, $L(FA(f_i)) = M(f_i)$. Also $L(FA(f)) = L(FA(f_1 \wedge f_2)) = L(FA(f_1)) \cap L(FA(f_2))$. So $L(FA(f)) = M(f)$.

— Case $f = \bigcirc f_1$:
According to (1), $M(f) = True \,.\, M(f_1)$, which again is trivial to check that this is equivalent with $L(FA(f))$.

— Case $f = \langle\rangle f_1$:
According to (1), models for f consists of a prefix in which any truth assignment is satisfactory, prefixed to models of f_1. So $M(f) = True^{*} \,.\, M(f_1)$ to which $L(FA(f))$ is equivalent.

— The cases $f = \Box f_1$ and $f = f_1 U f_2$ are a composition of the other cases.

The subset method to make automata deterministic may cause an exponential blow-up in the number of states. When implementing this transformation, it is clear that this approach is only practical when determinizations of Müller automata have to be done as few times as possible. Also the enumeration of the accepting sets in case of complementation may cause an exponential blow-up.

Complement and intersection are determinism preserving operations, only the union operation is not determinism preserving. Note that also the construction of arbitrary propositional formulae is determinism preserving. This is even the case for PTL formulae

with only the O as temporal operator. So these type of formulae can be dealt with as special simple cases. Examination of the transformation function *FA* even shows that the other temporal cases can also be simplified when one or both arguments is a propositional variable, or a PTL formula with only O operators.

The resulting automaton can be used as a model for the PTL formula, and on which other verifications and analyses can be performed. For example, a test on satisfiability, or model checking [12]. It is clear that satisfiability checking will become a trivial decision procedure, because when the PTL formula is satisfiable, it will result in a Müller automaton which accepts a non-empty 'language'. And an unsatisfiable PTL formula, i.e. a contradiction, results in an empty automaton.

Acknowledgements

I would like to thank the anonymous reviewers for their helpful and stimulating remarks. Also I wish to thank my colleague Geert-Leon Janssen with whom I had innumerable discussions on temporal logic and the various ways for finite and efficient representations of models.

The transformation method presented in this paper has been implemented on top of a finite automaton manipulation package. Interested readers are invited to write to the author to obtain a copy of the program and the finite automaton package.

References

[1] PNUELI, A., "Applications of Temporal Logic to the Specification and Verification of Reactive Systems: A Survey of Current Trends," *Current Trends in Concurrency: Overviews and Tutorials*, ed. J. W. de Bakker, W.-P. de Roever and G. Rozenberg, Lecture Notes in Computer Science 224, Springer Verlag, Berlin, pp. 510-584.

[2] V. BOCHMANN, G., "Hardware Specification with Temporal Logic: An Example," *IEEE Trans. on Computers*, vol. C-31, no. 3, March 1982, pp. 223-231.

[3] JANSSEN, G. L. J. M., "Hardware verification using Temporal Logic: A Practical View," *Formal VLSI Correctness Verification, VLSI Design Methods-II, Proc. of the IMEC-IFIP WG10.2 WG 10.5 International Workshop on Applied Formal Methods for Correct VLSI Design*, ed. L. J. M. Claesen, North-Holland, 1990, pp. 159-168.

[4] WOLPER, P., "Temporal Logic Can Be More Expressive," *Information and Control*, vol. 56, 1983, pp. 72-99.

[5] WOLPER, P., M. Y. VARDI, AND A. P. SISTLA, "Reasoning about Infinite Computation Paths," *Proc. 24th Ann. Symp. on Foundations of Computer Science*, Tucson, AZ, November 7-9,1983, pp. 185-193.

[6] MANNA, Z. AND A. PNUELI, "Specification and Verification of Concurrent Programs by ∀-Automata," *Proc. 14th ACM Symp. on Principles of Programming Languages*, Munich, January 21-23, 1987, pp. 1-12.

[7] ALPERN, B. AND F. B. SCHNEIDER, "Verifying Temporal Properties without Temporal Logic," *ACM Trans. on Programming Languages and Systems*, vol. 11, no. 1, January 1989, pp. 147-167.

[8] CHOUEKA, Y., "Theories of Automata on ω-Tapes: a Simplified Approach," *J. Comput. System Sci.*, vol. 8, 1974, pp. 117-141.

[9] SISTLA, A. P., M. Y. VARDI, AND P. WOLPER, "The Complementation Problems for Büchi Automata with Applications to Temporal Logic," *Proc. 12th Int. Colloquium on Automata, Languages and Programming (ICALP'85)*, Lecture Notes in Computer Science 194, Springer Verlag, Berlin, Napflion, Greece, July 1985, pp. 465-474.

[10] RABIN, M. O. AND D. SCOTT, "Finite Automata and their Decision Problems," *IBM J. Res. Develop.*, vol. 3, 1959, pp. 114-125.

[11] MCNAUGHTON, R., "Testing an Generating Infinite Sequences by a Finite Automaton," *Information and Control*, vol. 9, 1966, pp. 521-530.

[12] LICHTENSTEIN, O. AND A. PNUELI, "Checking That Finite State Concurrent Programs Satisfy Their Linear Specification," *Proc. 12th ACM Symp. on Principles of Programming Languages*, New Orleans, January 14-16, 1985, pp. 97-107.