

Alternating the Temporal Picture for Safety

Zohar Manna and Henny B. Sipma *

Computer Science Department
Stanford University
Stanford, CA. 94305-9045
{`manna,sipma`}@`cs.stanford.edu`

Abstract. We use alternating automata on infinite words to reduce the verification of linear temporal logic (LTL) safety properties over infinite-state systems to the proof of first-order verification conditions. This method generalizes the traditional deductive verification approach of providing verification rules for particular classes of formulas, such as invariances, nested precedence formulas, etc. It facilitates the deductive verification of arbitrary safety properties without the need for explicit temporal reasoning.

1 Introduction

Temporal logic is a powerful language for specifying properties of reactive systems. However, a specification language should be accompanied by verification methods to be useful in practice. For finite-state systems model checking provides such a verification method: it is (largely) automatic and applicable to arbitrary temporal properties. For infinite-state systems the situation is different. Although complete proof systems have been proposed for CTL [GF96] and LTL [MP91], the proof system presented for LTL requires the formula to be in a canonical form for the rules to be applicable. Transforming a formula into canonical form is expensive, the formula may grow exponentially, and worse, it may result in a formula that is so different from the original that the user's intuition proves useless for constructing the invariants and intermediate assertions necessary to complete the proof.

In this paper we present a verification rule `SAFE` that reduces the proof of LTL safety formulas to the proof of first-order validities. The rule `SAFE` constructs an *alternating automaton* [Var96,Var97] for the formula to be proven. This automaton may have to be strengthened by the user. First-order verification conditions are then generated based on the structure and labeling of this automaton.

The approach resembles that of *verification diagrams* [BMS95] and *assertion graphs* [BBM97], which also reduce the proof of temporal properties to first-order

* This research was supported in part by the National Science Foundation under grant CCR-98-04100 and CCR-99-00984 ARO under grants DAAH04-96-1-0122 and DAAG55-98-1-0471, ARO under MURI grant DAAH04-96-1-0341, by Army contract DABT63-96-C-0096 (DARPA), and by Air Force contract F33615-99-C-3014.

verification conditions. In principle verification diagrams can be generated automatically (apart from the strengthening and refinement necessary for fairness), using an algorithm similar to the tableau construction for LTL, thus reducing a temporal property to first-order verification conditions. However, verification diagrams are based on (nondeterministic) ω -automata, and the size of the resulting diagram can, in the worst case, be exponential in the size of the formula, giving rise to a number of first-order verification conditions of the same order of magnitude, which clearly is undesirable. Alternating automata have an advantage over the regular ω -automata that they are linear in the size of the formula, thus making the number of verification conditions generated also proportional to it.

The remainder of the paper is organized as follows. Section 2 provides the preliminaries: it presents our computational model of transition systems, and specification language of linear temporal logic (LTL). Alternating automata and their models are introduced in Section 3. In Section 4 we give an algorithm to construct an alternating automaton for future LTL formulas, and we prove that the language accepted by the constructed automaton is precisely the set of sequences of states that satisfy the formula. Section 5 proposes the verification rules B-SAFE and SAFE that reduce the verification of future safety formulas to first-order verification conditions, and it is shown that the special verification rules of [MP95] are subsumed by this rule. In Section 6 we give an algorithm to construct an alternating automaton for LTL formulas involving past operators, and we propose a verification rule for such formulas. Finally, in Section 7 we discuss some of the limitations of these rules and give some ideas on how they could be overcome.

2 Preliminaries

2.1 Computational Model: Fair Transition Systems

The computational model used for reactive systems is that of a *transition system* [MP95] (TS), $\mathcal{S} = \langle V, \Theta_{\mathcal{S}}, \mathcal{T} \rangle$, where V is a finite set of variables, $\Theta_{\mathcal{S}}$ is an initial condition, and \mathcal{T} is a finite set of transitions. A *state* s is an interpretation of V , and Σ denotes the set of all states. A transition $\tau \in \mathcal{T}$ is a function $\tau : \Sigma \mapsto 2^{\Sigma}$, and each state in $\tau(s)$ is called a τ -successor of s . We say that a transition τ is *enabled* on s if $\tau(s) \neq \emptyset$, otherwise τ is *disabled* on s . Each transition τ is represented by a *transition relation* $\rho_{\tau}(s, s')$, an assertion that expresses the relation between the values of V in s and the values of V (referred to by V') in any of its τ -successors s' .

A *run* of \mathcal{S} is an infinite sequence of states such that the first state satisfies $\Theta_{\mathcal{S}}$ and any two consecutive states satisfy a ρ_{τ} for some $\tau \in \mathcal{T}$. A state s is called *\mathcal{S} -accessible* if it appears in some run of \mathcal{S} . The set of all runs of \mathcal{S} is denoted by $\mathcal{L}(\mathcal{S})$.

2.2 Specification Language: Linear Temporal Logic

The specification language studied in this paper is *linear temporal logic*. We assume an underlying *assertion language* which is a first-order language over

interpreted symbols for expressing functions and relations over some concrete domains. We refer to a formula in the assertion language as a *state formula* or *assertion*. A *temporal formula* is constructed out of state formulas to which we apply the boolean connectives and the temporal operators shown below.

Temporal formulas are interpreted over a *model*, which is an infinite sequence of states $\sigma : s_0, s_1, \dots$. Given a model σ , a state formula p and temporal formulas φ and ψ , we present an inductive definition for the notion of a formula φ holding at a position $j \geq 0$ in σ , denoted by $(\sigma, j) \models \varphi$.

For a state formula:

$$(\sigma, j) \models p \quad \text{iff} \quad s_j \models p, \quad \text{that is, } p \text{ holds on state } s_j.$$

For the boolean connectives:

$$\begin{aligned} (\sigma, j) \models \phi \wedge \psi & \quad \text{iff} \quad (\sigma, j) \models \phi \text{ and } (\sigma, j) \models \psi \\ (\sigma, j) \models \phi \vee \psi & \quad \text{iff} \quad (\sigma, j) \models \phi \text{ or } (\sigma, j) \models \psi \\ (\sigma, j) \models \neg \phi & \quad \text{iff} \quad (\sigma, j) \not\models \phi . \end{aligned}$$

For the future temporal operators:

$$\begin{aligned} (\sigma, j) \models \bigcirc \phi & \quad \text{iff} \quad (\sigma, j+1) \models \phi \\ (\sigma, j) \models \square \phi & \quad \text{iff} \quad (\sigma, i) \models \phi \text{ for all } i \geq j \\ (\sigma, j) \models \diamond \phi & \quad \text{iff} \quad (\sigma, i) \models \phi \text{ for some } i \geq j \\ (\sigma, j) \models \phi \mathcal{U} \psi & \quad \text{iff} \quad (\sigma, k) \models \psi \text{ for some } k \geq j, \\ & \quad \text{and } (\sigma, i) \models \phi \text{ for every } i, j \leq i < k \\ (\sigma, j) \models \phi \mathcal{W} \psi & \quad \text{iff} \quad (\sigma, j) \models \phi \mathcal{U} \psi \text{ or } (\sigma, j) \models \square \phi . \end{aligned}$$

For the past temporal operators:

$$\begin{aligned} (\sigma, j) \models \ominus \phi & \quad \text{iff} \quad j > 0 \text{ and } (\sigma, j-1) \models \phi \\ (\sigma, j) \models \odot \phi & \quad \text{iff} \quad j = 0 \text{ or } (\sigma, j-1) \models \phi \\ (\sigma, j) \models \square \phi & \quad \text{iff} \quad (\sigma, i) \models \phi \text{ for all } 0 \leq i \leq j \\ (\sigma, j) \models \diamond \phi & \quad \text{iff} \quad (\sigma, i) \models \phi \text{ for some } 0 \leq i \leq j \\ (\sigma, j) \models \phi \mathcal{S} \psi & \quad \text{iff} \quad (\sigma, k) \models \psi \text{ for some } k \leq j, \\ & \quad \text{and } (\sigma, i) \models \phi \text{ for every } i, k < i \leq j \\ (\sigma, j) \models \phi \mathcal{B} \psi & \quad \text{iff} \quad (\sigma, j) \models \phi \mathcal{S} \psi \text{ or } (\sigma, j) \models \square \phi . \end{aligned}$$

An infinite sequence of states σ *satisfies* a temporal formula ϕ , written $\sigma \models \phi$, if $(\sigma, 0) \models \phi$. The set of all sequences that satisfy a formula φ is denoted by $\mathcal{L}(\varphi)$, the *language* of φ .

We say that a formula is a future (past) formula if it contains only state formulas, boolean connectives and future (past) temporal operators. We say that a formula is a general safety formula if it is of the form $\square \varphi$, for a past formula φ .

A state formula p is called *\mathcal{S} -state valid* if it holds over all \mathcal{S} -accessible states. A temporal formula φ is called *\mathcal{S} -valid* (valid over system \mathcal{S}), denoted by

$$\mathcal{S} \models \varphi ,$$

if it holds over all runs of \mathcal{S} .

3 Alternating Automata

Alternating automata are a generalization of nondeterministic automata. Nondeterministic automata have an existential flavor: a word is accepted if it is accepted by *some* path through the automaton. On the other hand \forall -automata [MP87] have a universal flavor: a word is accepted if it is accepted by *all* paths. Alternating automata combine the two flavors by allowing choices along a path to be marked as either existential or universal.

Example Consider the two automata shown in Figure 1. An arc between two edges denotes an “*and*” choice: both paths have to be accepting. The absence of an arc denotes the (regular) “*or*” choice.

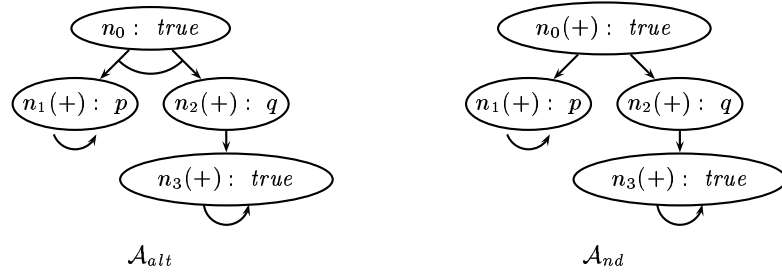


Fig. 1. An alternating automaton \mathcal{A}_{alt} and a nondeterministic automaton \mathcal{A}_{nd} .

Automaton \mathcal{A}_{alt} accepts only sequences of the form

$$\langle -, - \rangle, \langle p, q \rangle, \langle p, - \rangle, \langle p, - \rangle, \langle p, - \rangle, \dots$$

where a “ $-$ ” denotes a “don’t care”. For a sequence to be accepted, it has to be accepted by both branches. Automaton \mathcal{A}_{nd} , on the other hand, accepts sequences of the form

$$\langle -, - \rangle, \langle p, - \rangle, \langle p, - \rangle, \langle p, - \rangle, \langle p, - \rangle, \dots$$

and sequences of the form

$$\langle -, - \rangle, \langle -, q \rangle, \langle -, - \rangle, \langle -, - \rangle, \langle -, - \rangle, \dots$$

□

Automata are usually defined with input symbols labeling the edges. However, for our purposes it is more convenient to have them label the nodes. Therefore our definition of alternating automata is somewhat different from those found in [Var96, Var97].

Definition 1 (Alternating Automaton) An *alternating automaton* \mathcal{A} is defined recursively as follows:

$\mathcal{A} ::= \epsilon_{\mathcal{A}}$		empty automaton
$\langle \nu, \delta, f \rangle$		single node
$\mathcal{A} \wedge \mathcal{A}$		conjunction of two automata
$\mathcal{A} \vee \mathcal{A}$		disjunction of two automata

where ν is a state formula, δ is an alternating automaton expressing the next-state relation, and f indicates whether the node is accepting (denoted by $+$) or rejecting (denoted by $-$). We require that the automaton be finite.

The set of nodes of an alternating automaton \mathcal{A} , denoted by $\mathcal{N}(\mathcal{A})$ is formally defined as

$$\begin{aligned} \mathcal{N}(\epsilon_{\mathcal{A}}) &= \emptyset \\ \mathcal{N}(\langle \nu, \delta, f \rangle) &= \langle \nu, \delta, f \rangle \cup \mathcal{N}(\delta) \\ \mathcal{N}(\mathcal{A}_1 \wedge \mathcal{A}_2) &= \mathcal{N}(\mathcal{A}_1) \cup \mathcal{N}(\mathcal{A}_2) \\ \mathcal{N}(\mathcal{A}_1 \vee \mathcal{A}_2) &= \mathcal{N}(\mathcal{A}_1) \cup \mathcal{N}(\mathcal{A}_2) \end{aligned}$$

We denote with $\mathcal{N}_{rej}(\mathcal{A})$ the set of nodes of \mathcal{A} that are rejecting, that is,

$$\mathcal{N}_{rej}(\mathcal{A}) = \{n \in \mathcal{N}(\mathcal{A}) \mid f(n) = -\} .$$

Example The automata shown in Figure 1 can be written as follows:

$$\mathcal{A}_{alt} : \langle true, \mathcal{A}_1 \wedge \langle q, \mathcal{A}_2, + \rangle, + \rangle \quad \text{and} \quad \mathcal{A}_{nd} : \langle true, \mathcal{A}_1 \vee \langle q, \mathcal{A}_2, + \rangle, + \rangle$$

where

$$\mathcal{A}_1 = \langle p, \mathcal{A}_1, + \rangle \quad \text{and} \quad \mathcal{A}_2 = \langle true, \mathcal{A}_2, + \rangle .$$

□

A path through a regular ω -automaton is an infinite sequence of nodes. A “path” through an alternating ω -automaton is, in general, a tree. To define the language of an alternating automaton, we first define a tree.

Definition 2 A *tree* is defined recursively as follows:

$T ::= \epsilon_T$		empty tree
$T \cdot T$		composition
$\langle node, T \rangle$		single node with child tree

A tree may have both finite and infinite branches.

Definition 3 (Run) Given an infinite sequence of states $\sigma : s_0, s_1, \dots$, a tree T is called a *run* of σ in \mathcal{A} if one of the following holds:

$\mathcal{A} = \epsilon_{\mathcal{A}}$	and	$T = \epsilon_T$
$\mathcal{A} = n$	and	$T = \langle n, T' \rangle$ and $s_0 \models \nu(n)$ and T' is a run of s_1, s_2, \dots in $\delta(n)$
$\mathcal{A} = \mathcal{A}_1 \wedge \mathcal{A}_2$	and	$T = T_1 \cdot T_2$, T_1 is a run of \mathcal{A}_1 and T_2 is a run of \mathcal{A}_2
$\mathcal{A} = \mathcal{A}_1 \vee \mathcal{A}_2$	and	T is a run of \mathcal{A}_1 or T is a run of \mathcal{A}_2

Definition 4 (Accepting run) A run T is *accepting* if every infinite branch contains infinitely many accepting nodes.

Example A run of the sequence

$$\sigma : \langle p, q \rangle, \langle p, q \rangle, \langle p, \neg q \rangle, \langle p, q \rangle, \dots$$

in the automaton \mathcal{A}_{alt} of Figure 1 is shown in Figure 2. The run is clearly accepting since both branches contain infinitely many accepting nodes. \square

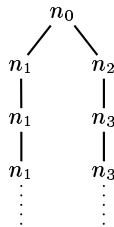


Fig. 2. Run of $\sigma : \langle p, q \rangle, \langle p, q \rangle, \langle p, \neg q \rangle, \langle p, q \rangle, \dots$ in \mathcal{A}_{alt}

Definition 5 (Model) An infinite sequence of states σ is a *model* of an alternating automaton \mathcal{A} if there exists an accepting run of σ in \mathcal{A} .

The set of models of an automaton \mathcal{A} , also called the *language of \mathcal{A}* , is denoted by $\mathcal{L}(\mathcal{A})$.

4 Linear Temporal Logic: Future Formulas

It has been shown that for every LTL formula φ there exists an alternating automaton \mathcal{A} such that $\mathcal{L}(\varphi) = \mathcal{L}(\mathcal{A})$ and the size of \mathcal{A} is linear in the size of φ [Var97]. In [Var97] a construction method is given for such an automaton with propositions labeling the edges. Since we prefer to label the nodes with propositions (or, in our case, state formulas), we present a slightly different procedure. In the remainder of this paper we assume that all negations have been pushed in to the state level (a full set of rewrite rules to accomplish this is given in [MP95]), that is, no temporal operator is in the scope of a negation.

Given an LTL formula φ , an alternating automaton $\mathcal{A}(\varphi)$ is constructed, as follows.

For a state formula p :

$$\mathcal{A}(p) = \langle p, \epsilon_{\mathcal{A}}, + \rangle .$$

For temporal formulas φ and ψ :

$$\begin{aligned}
\mathcal{A}(\varphi \wedge \psi) &= \mathcal{A}(\varphi) \wedge \mathcal{A}(\psi) \\
\mathcal{A}(\varphi \vee \psi) &= \mathcal{A}(\varphi) \vee \mathcal{A}(\psi) \\
\mathcal{A}(\bigcirc \varphi) &= \langle \text{true}, \mathcal{A}(\varphi), + \rangle \\
\mathcal{A}(\square \varphi) &= \langle \text{true}, \mathcal{A}(\square \varphi), + \rangle \wedge \mathcal{A}(\varphi) \\
\mathcal{A}(\diamond \varphi) &= \langle \text{true}, \mathcal{A}(\diamond \varphi), - \rangle \vee \mathcal{A}(\varphi) \\
\mathcal{A}(\varphi \mathcal{U} \psi) &= \mathcal{A}(\psi) \vee (\langle \text{true}, \mathcal{A}(\varphi \mathcal{U} \psi), - \rangle \wedge \mathcal{A}(\varphi)) \\
\mathcal{A}(\varphi \mathcal{W} \psi) &= \mathcal{A}(\psi) \vee (\langle \text{true}, \mathcal{A}(\varphi \mathcal{W} \psi), + \rangle \wedge \mathcal{A}(\varphi))
\end{aligned}$$

The constructions are illustrated in Figure 3.

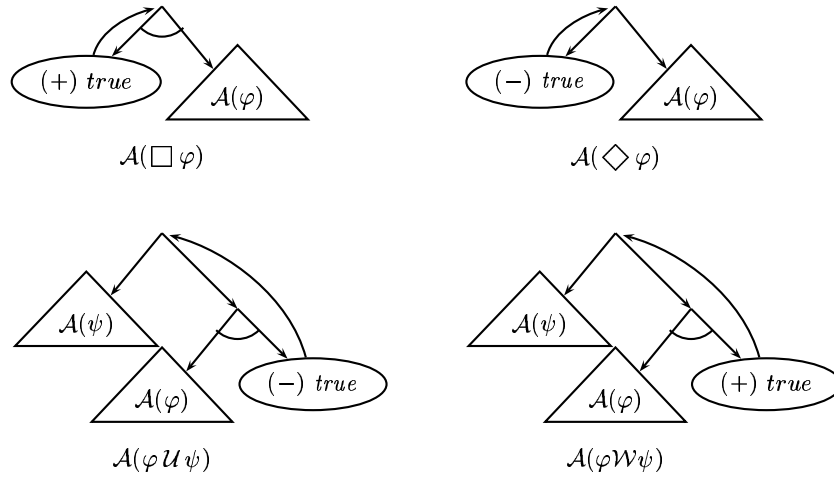


Fig. 3. Alternating automata for the temporal operators \square , \diamond , \mathcal{U} , \mathcal{W}

Note the close resemblance between these constructions and the expansion congruences [MP95]:

$$\begin{aligned}
\square \varphi &\approx \varphi \wedge \bigcirc \square \varphi \\
\diamond \varphi &\approx \varphi \vee \bigcirc \diamond \varphi \\
\varphi \mathcal{U} \psi &\approx \psi \vee (\varphi \wedge \bigcirc (\varphi \mathcal{U} \psi)) \\
\varphi \mathcal{W} \psi &\approx \psi \vee (\varphi \wedge \bigcirc (\varphi \mathcal{W} \psi))
\end{aligned}$$

Alternating automata represent these congruences completely, while in addition capturing the acceptance condition, not encoded in them.

Example The automaton for the wait-for formula, for state formulas p , q , and r :

$$\varphi : \square(p \rightarrow q\mathcal{W}r)$$

is shown in Figure 4. A run for the sequence

$$\sigma : \langle \neg p, -, - \rangle, \langle p, q, \neg r \rangle, \langle \neg p, q, \neg r \rangle, \langle p, \neg q, r \rangle, \langle \neg p, -, - \rangle, \langle \neg p, -, - \rangle, \dots$$

is shown in Figure 5.

The automaton for the formula

$$\varphi : \Box(p \rightarrow q \mathcal{U} r)$$

is identical to that of Figure 4 except that node n_4 is rejecting. \square

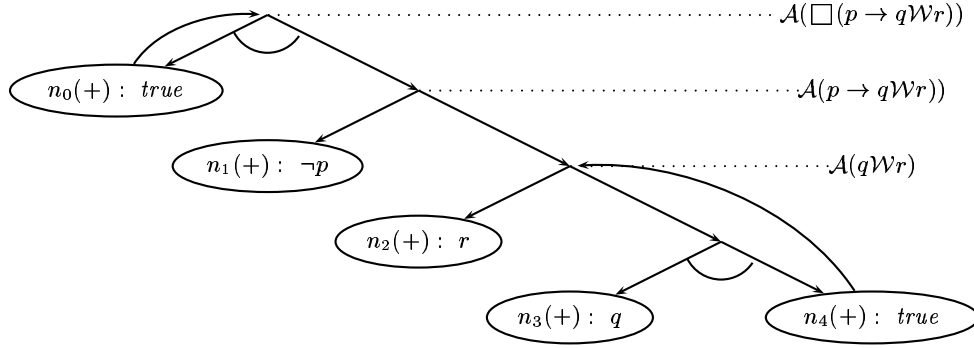


Fig. 4. Alternating automaton for $\Box(p \rightarrow q \mathcal{U} r)$

It is easy to see that, without changing the language of the automaton, we can make the following simplifications to the construction, for state formula p :

$$\begin{aligned} \mathcal{A}(\Box p) &= \langle p, \mathcal{A}(\Box p), + \rangle \\ \mathcal{A}(p \mathcal{U} \psi) &= \mathcal{A}(\psi) \vee \langle p, \mathcal{A}(p \mathcal{U} \psi), - \rangle \\ \mathcal{A}(p \mathcal{W} \psi) &= \mathcal{A}(\psi) \vee \langle p, \mathcal{A}(p \mathcal{W} \psi), + \rangle \end{aligned}$$

In this case the automaton for $\Box(p \rightarrow q \mathcal{U} r)$ becomes the one shown in Figure 6.

Theorem 1. For a future temporal formula φ , $\mathcal{L}(\varphi) = \mathcal{L}(\mathcal{A}(\varphi))$.

Before we prove this theorem, we state a supporting lemma, which we will use without mention in the proof below.

Lemma 1. For two automata \mathcal{A}_1 and \mathcal{A}_2 ,

- (1) $\mathcal{L}(\mathcal{A}_1) \cap \mathcal{L}(\mathcal{A}_2) = \mathcal{L}(\mathcal{A}_1 \wedge \mathcal{A}_2)$
- (2) $\mathcal{L}(\mathcal{A}_1) \cup \mathcal{L}(\mathcal{A}_2) = \mathcal{L}(\mathcal{A}_1 \vee \mathcal{A}_2)$

Proof This follows directly from the definition of a run. \square

Proof of Theorem 1 The proof is by induction on the structure of the formula. In each of the cases we assume an arbitrary sequence of states $\sigma : s_0, s_1, \dots$, and denote the sequence s_i, s_{i+1}, \dots by σ^i .

- φ is a state formula. In this case $\mathcal{A} = \langle \varphi, \epsilon_{\mathcal{A}}, + \rangle$. \mathcal{A} accepts all sequences whose first state satisfies φ , which are exactly the sequences that satisfy φ .
- $\varphi = \varphi_1 \wedge \varphi_2$; $\varphi = \varphi_1 \vee \varphi_2$. These follow directly from Lemma 1.
- $\varphi = \bigcirc \psi$. In this case $\mathcal{A}(\varphi) = \langle true, \mathcal{A}(\psi), + \rangle$. We prove the two directions separately.
 - Assume $\sigma \in \mathcal{L}(\mathcal{A}(\varphi))$. Then σ has an accepting run T in $\mathcal{A}(\varphi)$ such that $T = \langle n, T' \rangle$, and T' is a run of σ^1 in $\mathcal{A}(\psi)$. Clearly if T is accepting, so is T' , and thus $\sigma^1 \in \mathcal{L}(\mathcal{A}(\psi))$. Then, by the inductive hypothesis, $\sigma^1 \models \psi$, and thus, by the definition of \bigcirc , $\sigma \models \varphi$.
 - Assume $\sigma \models \bigcirc \psi$. Then, $\sigma^1 \models \psi$, and, by the inductive hypothesis, $\sigma^1 \in \mathcal{L}(\mathcal{A}(\psi))$, and, by the definition of a run, $\sigma \in \mathcal{L}(\langle true, \mathcal{A}(\psi), + \rangle)$.
- $\varphi = \square \psi$. In this case $\mathcal{A}(\varphi) = \langle true, \mathcal{A}(\varphi), + \rangle \wedge \mathcal{A}(\psi)$. We prove the two directions separately.
 - Assume $\sigma \in \mathcal{L}(\mathcal{A}(\varphi))$, and for the sake of contradiction, $\sigma \not\models \square \psi$. Then there must be some $i \geq 0$ such that $\sigma^i \not\models \psi$. But then, by the inductive hypothesis, $\sigma^i \notin \mathcal{L}(\mathcal{A}(\psi))$, and, by the definition of $\mathcal{A}(\square \psi)$ (and by Lemma 1), $\sigma^i \notin \mathcal{L}(\mathcal{A}(\square \psi))$. For $i = 0$ this contradicts the assumption that $\sigma \in \mathcal{L}(\mathcal{A}(\varphi))$. For $i > 0$ we have that $\sigma^{i-1} \notin \mathcal{L}(\langle true, \mathcal{A}(\square \psi), + \rangle)$ and thus (again by Lemma 1 and the definition of $\mathcal{A}(\square \psi)$) that $\sigma^{i-1} \notin \mathcal{L}(\mathcal{A}(\square \psi))$. By downwards induction on i we can conclude that $\sigma \notin \mathcal{L}(\mathcal{A}(\square \psi))$, a contradiction. Therefore $\sigma \models \square \psi$.
 - Assume $\sigma \models \square \psi$. Then, by the semantics of \square , for all $i \geq 0$, $\sigma^i \models \psi$, and therefore, by the induction hypothesis, for all $i \geq 0$, $\sigma^i \in \mathcal{L}(\mathcal{A}(\psi))$. Let n_0 be $\langle true, \mathcal{A}(\square \psi), + \rangle$. We construct a tree T as shown in Figure 7, where T_{σ^i} is an accepting run of σ^i in $\mathcal{A}(\psi)$, which exists since for all $i \geq 0$, $\sigma^i \in \mathcal{L}(\mathcal{A}(\psi))$. We claim that T is an accepting run of σ in $\mathcal{A}(\square \psi)$ and therefore $\sigma \in \mathcal{L}(\mathcal{A}(\varphi))$. Indeed, consider an infinite branch in T . This branch is either n_0, n_0, n_0, \dots or it ends in a T_{σ^i} . In any case it contains an infinite number of accepting nodes.
- $\varphi = \diamond \psi$. In this case $\mathcal{A}(\varphi) = \langle true, \mathcal{A}(\varphi), - \rangle \vee \mathcal{A}(\psi)$.
 - Assume $\sigma \in \mathcal{L}(\mathcal{A}(\varphi))$ and let T be a run of σ in $\mathcal{L}(\mathcal{A}(\varphi))$. Let n_0 be $\langle true, \mathcal{A}(\varphi), - \rangle$. Then T has to be of the form n_0, \dots, n_0, T' , where T' is an accepting run of $\mathcal{A}(\psi)$. (Note that it cannot be n_0, n_0, \dots , because n_0 is rejecting.) Therefore there exists a suffix σ^i of σ such that $\sigma^i \in \mathcal{L}(\mathcal{A}(\psi))$. By the induction hypothesis, $\sigma^i \models \psi$ and therefore $\sigma \models \diamond \psi$.
 - Assume $\sigma \models \diamond \psi$. Then, by the semantics of \diamond , there exists $i \geq 0$ such that $\sigma^i \models \psi$, and by the inductive hypothesis, $\sigma^i \in \mathcal{L}(\mathcal{A}(\psi))$, and thus (by the definition of $\mathcal{A}(\diamond \psi)$ and Lemma 1) $\sigma^i \in \mathcal{L}(\mathcal{A}(\diamond \psi))$. If $i = 0$ we are done. If $i > 0$ then we have that $\sigma^{i-1} \in \mathcal{L}(\langle true, \mathcal{A}(\diamond \psi), - \rangle)$, and thus $\sigma^{i-1} \in \mathcal{L}(\mathcal{A}(\diamond \psi))$. By downwards induction on i we can conclude that $\sigma \in \mathcal{L}(\mathcal{A}(\diamond \psi))$.
- $\varphi = \varphi_1 \mathcal{U} \varphi_2$; $\varphi = \varphi_1 \mathcal{W} \varphi_2$. The proof of these cases proceeds along the same lines as those for $\square \psi$ and $\diamond \psi$, and is omitted.

□

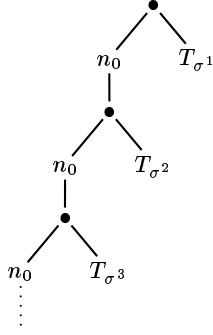


Fig. 7. Tree for σ in $\mathcal{A}(\Box \psi)$

5 Temporal Verification Rule for Future Safety Formulas

Alternating automata can be used to automatically reduce the verification of an arbitrary safety property specified by a future formula to first-order verification conditions, where a safety property is defined to be a property φ , such that if a sequence σ does not satisfy φ , then there is a finite prefix of σ such that φ is false on every extension of this prefix.

We define the *initial condition* of an alternating automaton \mathcal{A} , denoted by $\theta_{\mathcal{A}}(\mathcal{A})$, as follows:

$$\begin{aligned}
 \theta_{\mathcal{A}}(\epsilon_{\mathcal{A}}) &= \text{true} \\
 \theta_{\mathcal{A}}(\langle \nu, \delta, f \rangle) &= \nu \\
 \theta_{\mathcal{A}}(\mathcal{A}_1 \wedge \mathcal{A}_2) &= \theta_{\mathcal{A}}(\mathcal{A}_1) \wedge \theta_{\mathcal{A}}(\mathcal{A}_2) \\
 \theta_{\mathcal{A}}(\mathcal{A}_1 \vee \mathcal{A}_2) &= \theta_{\mathcal{A}}(\mathcal{A}_1) \vee \theta_{\mathcal{A}}(\mathcal{A}_2)
 \end{aligned}$$

Intuitively, the initial condition of an automaton characterizes the set of initial states of sequences accepted by the automaton. For example, the initial condition of the automaton shown in Figure 4 is

$$\theta_{\mathcal{A}}(\mathcal{A}) = \neg p \vee q \vee r .$$

Basic Rule

Following the style of verification rules of [MP95] we can now present the basic temporal rule B-SAFE, shown in Figure 8. In the rule we use the *Hoare triple* notation $\{p\} \tau \{q\}$, which stands for $p \wedge \rho_{\tau} \rightarrow q'$. The notation $\{p\} \mathcal{T} \{q\}$ stands for $\{p\} \tau \{q\}$ for all $\tau \in \mathcal{T}$.

Premise T1, the *Initiation Condition*, requires that the initial condition of \mathcal{S} implies the initial condition of the automaton $\mathcal{A}(\varphi)$. Premise T2, the *Consecution Condition*, requires that for all nodes, $n \in \mathcal{N}(\mathcal{A}(\varphi))$, and for all transitions $\tau \in \mathcal{T}$, τ , if enabled, leads to the initial condition of the next-state automaton of n .

For a future safety formula φ and TS $\mathcal{S} : \langle V, \Theta_{\mathcal{S}}, \mathcal{T} \rangle$,		
T1.	$\Theta_{\mathcal{S}} \rightarrow \theta_{\mathcal{A}}(\mathcal{A}(\varphi))$	
T2.	$\{\nu(n)\} \mathcal{T} \{\theta_{\mathcal{A}}(\delta(n))\}$	for $n \in \mathcal{N}(\mathcal{A}(\varphi))$
$\mathcal{S} \models \varphi$		

Fig. 8. Basic temporal rule B-SAFE

General Rule

As is the case with the rules B-INV and B-WAIT in [MP95], rule B-SAFE is hardly ever directly applicable, because the assertions labeling the nodes are not inductive: they must be strengthened. To represent the strengthening of an automaton, we add a new label μ to the definition of a node, $\langle \mu, \nu, \delta, f \rangle$, where μ is an assertion, and we change the definition of $\theta_{\mathcal{A}}$ for a node into

$$\theta_{\mathcal{A}}(\langle \mu, \nu, \delta, f \rangle) = \mu .$$

Using these definitions, Figure 9 shows the more general rule SAFE that allows strengthening of the intermediate assertions.

For a future safety formula φ , TS $\mathcal{S} : \langle V, \Theta_{\mathcal{S}}, \mathcal{T} \rangle$, and strengthened automaton $\mathcal{A}(\varphi)$		
T0.	$\mu(n) \rightarrow \nu(n)$	for $n \in \mathcal{N}(\mathcal{A}(\varphi))$
T1.	$\Theta_{\mathcal{S}} \rightarrow \theta_{\mathcal{A}}(\mathcal{A}(\varphi))$	
T2.	$\{\mu(n)\} \mathcal{T} \{\theta_{\mathcal{A}}(\delta(n))\}$	for $n \in \mathcal{N}(\mathcal{A}(\varphi))$
$\mathcal{S} \models \varphi$		

Fig. 9. General temporal rule SAFE

Note that terminal nodes, that is, nodes with $\delta = \epsilon_{\mathcal{A}}$, never need to be strengthened. This is so, because consecution conditions from terminal nodes are all of the form $\mu(n) \wedge \rho_{\tau} \rightarrow true$, since $\theta_{\mathcal{A}}(\epsilon_{\mathcal{A}}) = true$, and thus trivially valid.

Some strengthening can be applied automatically. For automata of the form

$$\mathcal{A} : n_0 : \langle true, \mathcal{A}, f \rangle \wedge \mathcal{A}_1$$

node n_0 can be strengthened with $\theta_{\mathcal{A}}(\mathcal{A}_1)$ without changing the language of the automaton. In the following special cases we will apply this default strengthening.

Special Cases

We consider the relation between rule SAFE and the special verification rules of [MP95].

INV

For an invariance formula $\varphi : \Box p$, with p a state formula, the (simplified) automaton $\mathcal{A}(\varphi)$ consists of a single node n_0 , labeled by p and with next-state automaton n_0 . If we strengthen node n_0 with χ , rule SAFE produces the following verification conditions (after applying the default strengthening):

$$\begin{aligned} \text{T0.} \quad & \chi \rightarrow p && \text{for } n_0 \\ \text{T1.} \quad & \Theta_S \rightarrow \chi \\ \text{T2.} \quad & \{\chi\} \mathcal{T} \{\chi\} && \text{for } n_0 \end{aligned}$$

which is identical to rule INV.

WAIT

For a wait-for formula,

$$\varphi : \Box(p \rightarrow q\mathcal{W}r)$$

rule SAFE, using the automaton shown in Figure 6 with node n_3 strengthened by χ , results in the following verification conditions (after applying the default strengthening):

$$\begin{aligned} \text{T0.} \quad & \chi \rightarrow q && \text{for } n_3 \\ \text{T1.} \quad & \Theta_S \rightarrow \neg p \vee r \vee \chi \\ \text{T2.} \quad & \{\neg p \vee r \vee \chi\} \mathcal{T} \{\neg p \vee r \vee \chi\} && \text{for } n_0 \\ & \{\chi\} \mathcal{T} \{\chi \vee r\} && \text{for } n_3 \end{aligned}$$

Comparing these conditions with those of rule WAIT we notice that the state-validity $p \rightarrow \chi \vee r$ in WAIT has been replaced by the invariance conditions for the same formula, represented by T1 and the first set of conditions of T2. The other conditions are identical.

N-WAIT

Consider the nested wait-for formula, with p, q_0, \dots, q_n state formulas:

$$\Box(p \rightarrow q_n \mathcal{W}(q_{n-1} \dots \mathcal{W}(q_1 \mathcal{W}q_0) \dots))$$

The procedure described above generates the (simplified) automaton shown in Figure 10 for this formula. It is easy to see that this automaton will generate the same verification conditions as given in N-WAIT, again with the exception that the state validity in N-WAIT is replaced by the invariance conditions for the same formula.

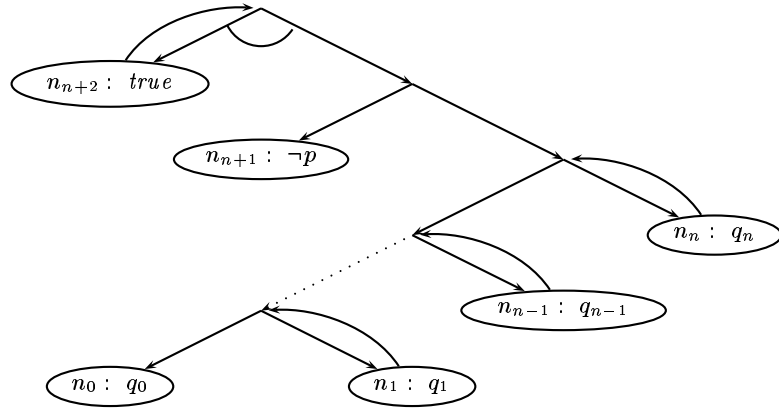


Fig. 10. Alternating automaton for $\Box(p \rightarrow q_n \mathcal{W}(q_{n-1} \dots \mathcal{W}(q_1 \mathcal{W}q_0) \dots))$

Theorem 2 (Soundness of rule B-SAFE). *For a TS \mathcal{S} and future safety formula φ , if the premises $T1$ and $T2$ of rule B-SAFE are \mathcal{S} -state valid then $\mathcal{S} \models \varphi$.*

Before we prove this theorem we present some supporting definitions and lemmas.

Definition 6 (k-Run) A finite tree T is a k -run of an infinite sequence of states $\sigma : s_0, s_1, \dots$ in an alternating automaton \mathcal{A} if one of the following holds:

$\mathcal{A} = \epsilon_{\mathcal{A}}$	and	$T = \epsilon_T$
$\mathcal{A} = n$	and	$T = \langle n, T' \rangle$ and $s_0 \models \nu(n)$ and
		(a) $k = 1$ and $T' = \epsilon_T$, or
		(b) $k > 1$ and T' is a $(k-1)$ -run of σ^1 in $\delta(n)$
$\mathcal{A} = \mathcal{A}_1 \wedge \mathcal{A}_2$	and	$T = T_1 \cdot T_2$ and T_1 is a k -run of σ in \mathcal{A}_1
		and T_2 is a k -run of σ in \mathcal{A}_2
$\mathcal{A} = \mathcal{A}_1 \vee \mathcal{A}_2$	and	T is a k -run of σ in \mathcal{A}_1 or T is a k -run of σ in \mathcal{A}_2

Definition 7 (k-Model) An infinite sequence of states σ is a k -model of an alternating automaton \mathcal{A} if there exists a k -run of σ in \mathcal{A} .

Example The sequence

$$\langle \neg p, -, - \rangle, \langle p, q, \neg r \rangle, \langle -, -, - \rangle, \dots$$

is a 2-model of the automaton shown in Figure 4. Its 2-run is shown in Figure 11. \square

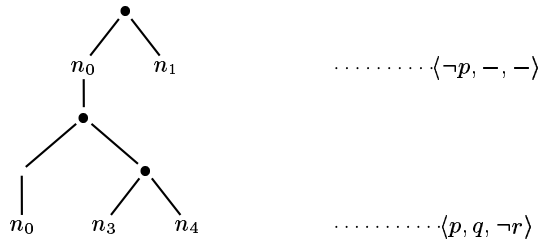


Fig. 11. 2-run of $\langle \neg p, -, - \rangle, \langle p, q, \neg r \rangle, \langle -, -, - \rangle, \dots$

Definition 8 (k-Nodes) The set of k -nodes of a tree T , written $\mathcal{N}_k(T)$ consists of the nodes present at depth k . Formally:

$$\begin{aligned} \mathcal{N}_k(\epsilon_T) &= \emptyset \\ \mathcal{N}_k(\langle n, T \rangle) &= \begin{cases} \{n\} & \text{if } k = 1 \\ \mathcal{N}_{k-1}(T) & \text{otherwise} \end{cases} \\ \mathcal{N}_k(T_1 \cdot T_2) &= \mathcal{N}_k(T_1) \cup \mathcal{N}_k(T_2) \end{aligned}$$

Example For the tree T shown in Figure 5, $\mathcal{N}_1(T) = \{n_0, n_1\}$, $\mathcal{N}_2(T) = \{n_0, n_3, n_4\}$, $\mathcal{N}_3(T) = \{n_0, n_1, n_3, n_4\}$, etc. \square

Lemma 2. For an infinite sequence of states $\sigma : s_0, s_1, \dots$, $k > 0$, and alternating automata \mathcal{A} , \mathcal{A}_1 , and \mathcal{A}_2 the following hold:

- σ is a k -model of $\epsilon_{\mathcal{A}}$.
- σ is a k -model of $\mathcal{A} = n$ iff $s_0 \models \nu(n)$ and σ^1 is a $(k-1)$ -model of $\delta(n)$.
- σ is a k -model of $\mathcal{A}_1 \wedge \mathcal{A}_2$ iff σ is a k -model of \mathcal{A}_1 and σ is a k -model of \mathcal{A}_2 .
- σ is a k -model of $\mathcal{A}_1 \vee \mathcal{A}_2$ iff σ is a k -model of \mathcal{A}_1 or σ is a k -model of \mathcal{A}_2 .

Proof These follow directly from the definition of k -run and k -model. \square

Lemma 3 (Initial Condition). *An infinite sequence of states $\sigma : s_0, \dots$ is a 1-model of an alternating automaton \mathcal{A} iff $s_0 \models \theta_{\mathcal{A}}(\mathcal{A})$.*

Proof The proof is by induction on the structure of the automaton.

- $\mathcal{A} = \epsilon_{\mathcal{A}}$. By lemma 2, the sequence σ is a 1-model of \mathcal{A} ; $\theta_{\mathcal{A}}(\epsilon_{\mathcal{A}}) = \text{true}$ and thus $s_0 \models \theta_{\mathcal{A}}(\mathcal{A})$.
- $\mathcal{A} = n$. The sequence σ is a 1-model of \mathcal{A} iff $s_0 \models \nu(n)$ iff $s_0 \models \theta_{\mathcal{A}}(\mathcal{A})$.
- $\mathcal{A} = \mathcal{A}_1 \wedge \mathcal{A}_2$. By lemma 2, the sequence σ is a 1-model of \mathcal{A} iff σ is a 1-model of \mathcal{A}_1 and σ is a 1-model of \mathcal{A}_2 , iff, by the inductive hypothesis, $s_0 \models \theta_{\mathcal{A}}(\mathcal{A}_1)$ and $s_0 \models \theta_{\mathcal{A}}(\mathcal{A}_2)$, iff $s_0 \models \theta_{\mathcal{A}}(\mathcal{A}_1) \wedge \theta_{\mathcal{A}}(\mathcal{A}_2)$ iff, by the definition of $\theta_{\mathcal{A}}$, $s_0 \models \theta_{\mathcal{A}}(\mathcal{A}_1 \wedge \mathcal{A}_2)$.
- $\mathcal{A} = \mathcal{A}_1 \vee \mathcal{A}_2$. Similar to the above case.

□

Lemma 4. *If an infinite sequence of states σ is a k -model of an alternating automaton \mathcal{A} with k -run T , and if for all nodes $n \in \mathcal{N}_k(T)$, σ^k is a 1-model of $\delta(n)$, then σ is a $(k+1)$ -model of \mathcal{A} .*

Proof Assume T is a k -run of σ in \mathcal{A} , and for all $n \in \mathcal{N}_k(T)$ σ^k is a 1-model of $\delta(n)$ with 1-run T_n . Then we can construct a tree T' by replacing each occurrence of $\langle n, \epsilon_T \rangle$ in T by $\langle n, T_n \rangle$. It is easy to see that T' is a $(k+1)$ -run of σ in \mathcal{A} , and thus σ is a $(k+1)$ -model of \mathcal{A} . □

Lemma 5. *An infinite sequence of states $\sigma : s_0, s_1 \dots$ is a k -model, with $k > 0$, of an alternating automaton \mathcal{A} , with k -run T iff for all $1 \leq j \leq k$, for all nodes $n \in \mathcal{N}_j(T)$, $s_{j-1} \models \nu(n)$.*

Proof By induction on k , and the structure of the tree T . □

Example Consider Figure 5. The state $s_2 : \langle \neg p, q, \neg r \rangle$ indeed satisfies the assertion ν of all nodes in $\mathcal{N}_3(T) = \{n_0, n_1, n_3, n_4\}$. □

Lemma 6. *Given a TS \mathcal{S} , a future temporal formula φ , and an infinite sequence of states $\sigma \in \mathcal{L}(\mathcal{S})$, if the premises T1 and T2 are \mathcal{S} -state valid, then σ is a k -model of $\mathcal{A}(\varphi)$ for all $k > 0$.*

Proof The proof is by induction on k .

- Base case. By the definition of a run of \mathcal{S} , $s_0 \models \Theta_{\mathcal{S}}$, and thus, by premise T1, $s_0 \models \theta_{\mathcal{A}}(\mathcal{A}(\varphi))$. Then, by lemma 3, σ is a 1-model of $\mathcal{A}(\varphi)$.
- Inductive step. Assume σ is a k -model of $\mathcal{A}(\varphi)$ with k -run T . Then, by lemma 5, for all $n \in \mathcal{N}_k(T)$, $s_{k-1} \models \nu(n)$. By the consecution requirement for runs, there exists some transition $\tau \in \mathcal{T}$ such that (s_{k-1}, s_k) satisfies ρ_{τ} . By premise T2, for every node $n \in \mathcal{N}(\mathcal{A}(\varphi))$, every transition starting from a state that satisfies $\nu(n)$ leads to a state that satisfies $\theta_{\mathcal{A}}(\delta(n))$, and thus, by lemma 3 and lemma 4, σ is a $(k+1)$ -model of $\mathcal{A}(\varphi)$.

□

Lemma 7. *For a safety formula φ , if a sequence of states σ is a k -model of $\mathcal{A}(\varphi)$ for any $k > 0$, then σ is a model of $\mathcal{A}(\varphi)$.*

Proof This follows directly from the definition of a safety formula: a safety property holds iff it cannot be violated in finite time. \square

Proof of Theorem 2 Consider a TS \mathcal{S} and a future safety formula φ and assume that the premises T1 and T2 are \mathcal{S} -state valid. Consider an arbitrary sequence of states $\sigma \in \mathcal{L}(\mathcal{S})$, and future safety formula φ . By lemma 6, σ is a k -model of $\mathcal{A}(\varphi)$ for all $k > 0$. Then, by lemma 7, σ is a model of $\mathcal{A}(\varphi)$. Finally, by theorem 1, $\sigma \models \varphi$. \square

6 Past formulas

In Section 4 we showed how to construct alternating automata for LTL formulas containing future operators only. Here we will extend the procedure to include past operators as well.

To define an alternating automaton for LTL formulas including past operators, we add a component g to the definition of a node, such that a node is now defined as

$$\langle \nu, \delta, f, g \rangle$$

where g indicates whether the node is past (indicated by “ \leftarrow ”) or future (indicated by “ \rightarrow ”).

To accomodate the presence of past nodes we extend the notions of a run and model of an automaton.

Definition 9 (Run) Given an infinite sequence of states $\sigma : s_0, s_1, \dots$, and a position $j \geq 0$, a tree T is called a run of σ at position j if one of the following holds:

$$\begin{array}{ll} \mathcal{A} = \epsilon_{\mathcal{A}} & \text{and} \quad T = \epsilon_T \\ \mathcal{A} = n & \text{and} \quad T = \langle n, T' \rangle \text{ and } s_0 \models \nu(n) \text{ and} \\ & \left\{ \begin{array}{l} \text{(a) } T' \text{ is a run of } \sigma \text{ in } \delta(n) \text{ at } j+1, \\ \quad \text{if } g(n) = \rightarrow, \text{ or} \\ \text{(b) } T' \text{ is a run of } \sigma \text{ in } \delta(n) \text{ at } j-1, \\ \quad \text{if } g(n) = \leftarrow \text{ and } j > 0, \text{ or} \\ \text{(c) } T' = \epsilon_T \text{ if } g(n) = \leftarrow, f(n) = +, \text{ and } j = 0. \end{array} \right. \\ \mathcal{A} = \mathcal{A}_1 \wedge \mathcal{A}_2 & \text{and} \quad T = T_1 \cdot T_2, \\ & T_1 \text{ is a run of } \mathcal{A}_1 \text{ and } T_2 \text{ is a run of } \mathcal{A}_2 \\ \mathcal{A} = \mathcal{A}_1 \vee \mathcal{A}_2 & \text{and} \quad T \text{ is a run of } \mathcal{A}_1 \text{ or } T \text{ is a run of } \mathcal{A}_2 \end{array}$$

Definition 10 (Accepting run) A run T is *accepting* if every infinite branch of T contains infinitely many accepting nodes.

Definition 11 (Model at j) An infinite sequence of states σ is a model of an alternating automaton \mathcal{A} at position j if there exists an accepting run of σ in \mathcal{A} at position j .

Definition 12 (Model) An infinite sequence of states σ is a model of an alternating automaton \mathcal{A} if it is a model at position 0.

Given an LTL formula φ , an alternating automaton $\mathcal{A}(\varphi)$ is constructed as before, where all nodes constructed before are future nodes, and with the following additions for the past operators:

$$\begin{aligned}
\mathcal{A}(\ominus \varphi) &= \langle \text{true}, \mathcal{A}(\varphi), +, \leftarrow \rangle \\
\mathcal{A}(\ominus \varphi) &= \langle \text{true}, \mathcal{A}(\varphi), -, \leftarrow \rangle \\
\mathcal{A}(\Box \varphi) &= \langle \text{true}, \mathcal{A}(\Box \varphi), +, \leftarrow \rangle \wedge \mathcal{A}(\varphi) \\
\mathcal{A}(\Diamond \varphi) &= \langle \text{true}, \mathcal{A}(\Diamond \varphi), -, \leftarrow \rangle \vee \mathcal{A}(\varphi) \\
\mathcal{A}(\varphi \mathcal{S} \psi) &= \mathcal{A}(\psi) \vee (\langle \text{true}, \mathcal{A}(\varphi \mathcal{S} \psi), -, \leftarrow \rangle \wedge \mathcal{A}(\varphi)) \\
\mathcal{A}(\varphi \mathcal{B} \psi) &= \mathcal{A}(\psi) \vee (\langle \text{true}, \mathcal{A}(\varphi \mathcal{B} \psi), +, \leftarrow \rangle \wedge \mathcal{A}(\varphi))
\end{aligned}$$

Again, these constructions closely resemble the expansion congruences for the past formulas:

$$\begin{aligned}
\Box \varphi &\approx \varphi \wedge \ominus \Box \varphi \\
\Diamond \varphi &\approx \varphi \vee \ominus \Diamond \varphi \\
\varphi \mathcal{S} \psi &\approx \psi \vee (\varphi \wedge \ominus (\varphi \mathcal{S} \psi)) \\
\varphi \mathcal{B} \psi &\approx \psi \vee (\varphi \wedge \ominus (\varphi \mathcal{B} \psi))
\end{aligned}$$

Example For a causality formula $\varphi : \Box(p \rightarrow \Diamond r)$ with p , q , and r state formulas, the automaton is shown in Figure 12. \square

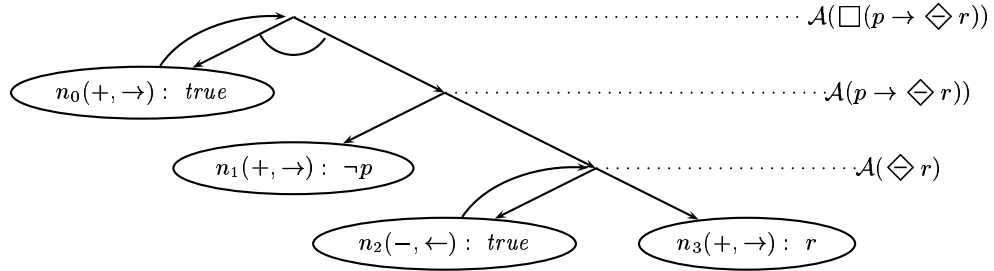


Fig. 12. Alternating automaton for $\Box(p \rightarrow \Diamond r)$

We denote with $\mathcal{PN}(\mathcal{A})$ the past nodes of \mathcal{A} and with $\mathcal{FN}(\mathcal{A})$ the future nodes of \mathcal{A} .

We can now formulate the verification rule GSAFE, shown in Figure 13, that is applicable to arbitrary general safety formulas. Again, we augment the definition of a node with a strengthening μ .

For a general safety formula φ , and ts $\mathcal{S} : \langle V, \Theta_S, \mathcal{T} \rangle$, and strengthened automaton $\mathcal{A}(\varphi)$		
T0.	$\mu(n) \rightarrow \nu(n)$	for $n \in \mathcal{N}(\mathcal{A}(\varphi))$
T1.	$\Theta_S \rightarrow \theta_{\mathcal{A}}(\mathcal{A}(\varphi))$	
T2.	$\{\mu(n)\} \mathcal{T} \{\theta_{\mathcal{A}}(\delta(n))\}$	for $n \in \mathcal{FN}(\mathcal{A}(\varphi))$
	$\{\mu(n)\} \mathcal{T}^{-1} \{\theta_{\mathcal{A}}(\delta(n))\}$	for $n \in \mathcal{PN}(\mathcal{A}(\varphi))$
T3.	$\Theta_S \rightarrow \neg\mu(n)$	for $n \in \mathcal{PN}_{rej}(\mathcal{A}(\varphi))$
$\mathcal{S} \models \varphi$		

Fig. 13. General safety rule GSAFE

Premise T0 requires that the assertions used to strengthen the node imply the original assertions. Premise T1 requires that the initial condition of the system implies the (strengthened) initial condition of the automaton for φ . Premise T2 requires the regular consecution condition for all future nodes, and the inverse consecution condition for all past nodes, where $\mathcal{T}^{-1} = \{\tau^{-1} \mid \tau \in \mathcal{T}\}$ and

$$\{p\} \tau^{-1} \{q\} = p' \wedge \rho_\tau \rightarrow q .$$

Finally, premise T3 requires that no initial state satisfies a rejecting past node. This last requirement ensures that “promises for the past” are fulfilled before the first state is reached.

Special cases

As for future formulas we compare the verification conditions produced by rule GSAFE with the premises of a special verification rule involving past operators presented in [MP95].

CAUS

For a causality formula $\varphi : \Box(p \rightarrow \Diamond r)$ with p , q , and r state formulas, rule GSAFE, based on the automaton shown in Figure 12 with node n_2 strengthened

with χ , results in the following verification conditions:

- T0. $\chi \rightarrow true$ for n_2
- T1. $\Theta_S \rightarrow \neg p \vee \chi \vee r$
- T2. $\{\neg p \vee \chi \vee r\} \mathcal{T} \{\neg p \vee \chi \vee r\}$ for n_0
 $\{\chi\} \mathcal{T}^{-1} \{\chi \vee r\}$ for n_2
- T3. $\Theta_S \rightarrow \neg \chi$

For state formulas p and r , the premises of the rule CAUS are the following:

- C1. $p \rightarrow \chi \vee r$
- C2. $\Theta_S \rightarrow \neg \chi \vee r$
- C3. $\{\chi\} \mathcal{T}^{-1} \{\chi \vee r\}$

Premise C1 corresponds to premises T1, and T2 for n_0 . Premise C2 is represented by the (stronger) premise T3, and premise C3 is identical to premise T2 for n_2 .

7 Discussion

The work presented in this paper is a first attempt to use alternating automata as a basis for the deductive verification of LTL properties. We have shown that it successfully generalizes several of the special verification rules for the corresponding classes of formulas, thus obviating the need to implement these rules separately in a verification tool. Instead the single rule GSAFE suffices. The rule SAFE has been implemented in **STeP**, the Stanford Temporal Prover, a verification tool for algorithmic and deductive verification of reactive systems [BBC⁺95, BBC⁺00].

It is straightforward to extend rule SAFE to general reactivity properties by adding a fourth premise

$$\text{T3. } \square \diamond \neg \mu(n) \quad \text{for all rejecting nodes } n \in \mathcal{N}_{rej}(\mathcal{A}(\varphi))$$

which can be reduced to first-order verification conditions by one of the rules given in [MP91]. The reason we have omitted this extension from this paper is that we found this rule not useful for many of these properties. We are currently working on a more general rule that allows the application of different proof methods on different parts of the automaton. This will be presented in a forthcoming paper.

8 Related Work

Alternating finite state automata over finite words were first introduced in [CKS81] and shown to be exponentially more succinct than nondeterministic

automata. In [MH84] it was shown that alternating finite automata on infinite words with Büchi acceptance conditions are as expressive as nondeterministic ω -automata with Büchi acceptance conditions, and in [MSS88,Var94] it was shown that for every LTL formula an alternating Büchi automaton can be constructed that is linear in the size of the formula. In [Var95,Var96,Var97] alternating automata are used as the basis for a model checking algorithm for finite-state systems and both linear and branching time temporal logics. In [Var98] a theory of two-way alternating automata on infinite trees is developed to check satisfiability of μ -calculus formulas with both forward and backward modalities. In that paper a direction is introduced in the next-state relation, redirecting a run to the parent node when encountering a past operator. The procedure presented in this paper takes the opposite approach for past operators: it reverses the direction in the sequence, instead of in the automaton.

Deductive verification methods using automata include a sound and complete proofrule based on \forall -automata [MP87], and generalized verification diagrams [BMS95,MBSU98].

Acknowledgements

We thank Anca Browne, Michael Colón, Bernd Finkbeiner, Matteo Slanina, Calogero Zarba, and Zhang Ting for their suggestions and comments on this paper.

References

- [BBC⁺95] N.S. Bjørner, A. Browne, E.S. Chang, M. Colón, A. Kapur, Z. Manna, H.B. Sipma, and T.E. Uribe. STeP: The Stanford Temporal Prover, User's Manual. Technical Report STAN-CS-TR-95-1562, Computer Science Department, Stanford University, November 1995. available from <http://www-step.stanford.edu/>.
- [BBC⁺00] N.S. Bjørner, A. Browne, M. Colón, B. Finkbeiner, Z. Manna, H.B. Sipma, and T.E. Uribe. Verifying temporal properties of reactive systems: A STeP tutorial. *Formal Methods in System Design*, 16(3):227–270, June 2000.
- [BBM97] N.S. Bjørner, A. Browne, and Z. Manna. Automatic generation of invariants and intermediate assertions. *Theoretical Computer Science*, 173(1):49–87, February 1997. Preliminary version appeared in 1st Intl. Conf. on Principles and Practice of Constraint Programming, vol. 976 of LNCS, pp. 589–623, Springer-Verlag, 1995.
- [BMS95] A. Browne, Z. Manna, and H.B. Sipma. Generalized temporal verification diagrams. In *15th Conference on the Foundations of Software Technology and Theoretical Computer Science*, vol. 1026 of *Lecture Notes in Computer Science*, pages 484–498. Springer-Verlag, 1995.
- [CKS81] A.K. Chandra, D.C. Kozen, and L.J. Stockmeyer. Alternation. *J. ACM*, 28:114–133, 1981.
- [GF96] O. Grumberg and L. Fix. Verification of temporal properties. *Logic and Computation*, 6(3):343–361, June 1996.

- [MBSU98] Z. Manna, A. Browne, H.B. Sipma, and T.E. Uribe. Visual abstractions for temporal verification. In A. Haeberer, editor, *Algebraic Methodology and Software Technology (AMAST'98)*, vol. 1548 of *Lecture Notes in Computer Science*, pages 28–41. Springer-Verlag, December 1998.
- [MH84] S. Miyano and T. Hayashi. Alternating finite automata on ω -words. *Theoretical Computer Science*, 32, 1984.
- [MP87] Z. Manna and A. Pnueli. Specification and verification of concurrent programs by \forall -automata. In B. Banieqbal, H. Barringer, and A. Pnueli, editors, *Temporal Logic in Specification*, number 398 in *Lecture Notes in Computer Science*, pages 124–164. Springer-Verlag, Berlin, 1987. Also in *Proc. 14th ACM Symp. Princ. of Prog. Lang.*, Munich, Germany, pp. 1–12, January 1987.
- [MP91] Z. Manna and A. Pnueli. Completing the temporal picture. *Theoretical Computer Science*, 83(1):97–130, 1991.
- [MP95] Z. Manna and A. Pnueli. *Temporal Verification of Reactive Systems: Safety*. Springer-Verlag, New York, 1995.
- [MSS88] D.E. Muller, A. Saoudi, and P.E. Schupp. Weak alternating automata give a simple explanation of why most temporal and dynamic logics are decidable in exponential time. In *Proc. 3rd IEEE Symp. Logic in Comp. Sci.*, pages 422–427, 1988.
- [Var94] M.Y. Vardi. Nontraditional applications of automata theory. In M. Hagiya and J.C. Mitchell, editors, *Proc. International Symposium on Theoretical Aspects of Computer Software*, vol. 789 of *Lecture Notes in Computer Science*, pages 575–597. Springer-Verlag, 1994.
- [Var95] M.Y. Vardi. Alternating automata and program verification. In J. van Leeuwen, editor, *Computer Science Today. Recent Trends and Developments*, vol. 1000 of *Lecture Notes in Computer Science*, pages 471–485. Springer-Verlag, 1995.
- [Var96] M.Y. Vardi. An automata-theoretic approach to linear temporal logic. In F. Moller and G. Birtwistle, editors, *Logics for Concurrency. Structure versus Automata*, vol. 1043 of *Lecture Notes in Computer Science*, pages 238–266. Springer-Verlag, 1996.
- [Var97] M.Y. Vardi. Alternating automata: Checking truth and validity for temporal logics. In *Proc. of the 14th Intl. Conference on Automated Deduction*, vol. 1249 of *Lecture Notes in Computer Science*. Springer-Verlag, July 1997.
- [Var98] M.Y. Vardi. Reasoning about the past with two-way automata. In K. Larsen, S. Skyum, and G. Winskel, editors, *Proc. 25th Intl. Colloq. Aut. Lang. Prog.*, *Lecture Notes in Computer Science*, pages 628–641, Aalborg, Denmark, 1998. Springer-Verlag.