

Verification Diagrams: Logic + Automata

Zohar Manna and Henny B. Sipma *

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

Abstract. We use automata on infinite words to reduce the verification of linear temporal logic (LTL) properties over infinite-state systems to the proof of first-order verification conditions and an algorithmic language inclusion check. The automaton serves as a temporal abstraction of the system, preserving a subset of both safety and liveness properties. The first-order verification conditions prove that the abstraction is conservative; the algorithmic check verifies that the abstraction satisfies the property. Automata precisely separate the combinatoric from the logic part of the proof, such that the combinatoric part can be handled completely by algorithmic methods.

1 Introduction

Verification diagrams cleanly separate combinatorics, handled by the underlying automata, from logic, represented by first-order verification conditions, in the proof that a reactive system satisfies a temporal specification. Automata are ubiquitous in program verification. However, all of their use has been in model checking [Kur94,VW86], the combinatoric part of the proof: both the system and the negation of the property are represented as a finite-state automaton and property satisfaction is checked by means of a decidable emptiness check of the product automaton. In this paper we show that automata can also successfully be used in the verification of infinite-state systems in the form of *verification diagrams* [MP94]. These diagrams are temporal abstractions of the system that preserve liveness properties: the acceptance condition of the automaton restricts the infinite behavior of the abstract system [BMS95,MBSU98].

To show that a system \mathcal{S} satisfies a temporal property φ , a verification diagram \mathcal{G} is constructed such that the language inclusion (where the language of a diagram is similar to that of the underlying automaton)

$$\mathcal{L}(\mathcal{S}) \subseteq \mathcal{L}(\mathcal{G})$$

* 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.

can be proved by first-order verification conditions, and the language inclusion

$$\mathcal{L}(\mathcal{G}) \subseteq \mathcal{L}(\varphi)$$

can be proved algorithmically, thus separating the deductive and algorithmic parts of the proof, and eliminating the need to perform any deductive temporal reasoning.

Construction of the diagram may be an iterative process, starting with the diagram based on the automaton for the property and refining this diagram until all first-order verification conditions can be proved. In this case the diagram is guaranteed to satisfy the property. The verification diagram is a true abstraction of the system in the same domain: it over approximates the set of computations of the system.

Like in model checking one can also start with a diagram based on the automaton for the negation of the property. The resulting *falsification diagram* [SUM99] over approximates the set of computations of the system that do not satisfy the property. The goal is now to refine the diagram, justified by first-order verification conditions, until it is empty, proving that no computation satisfies the negation of the property. This process is called *Deductive Model Checking*.

Both verification diagrams and falsification diagrams take as starting point a nondeterministic ω -automaton [Tho88] for the (negation of the) property. The size of the automaton is worst-case exponential in the size of the property, which is undesirable, since the number of first-order verification conditions is proportional to the size of the automaton. Recently we have investigated *alternating automata*, which are linear in the size of the property, as the basis for diagrams and verification rules [MS00].

In this paper we will give an overview of the use of diagrams in verification. The remainder of the paper is organized as follows. Section 2 provides the preliminaries: our computational model of fair transition systems, our specification language of linear temporal logic (LTL), and the basics of ω -automata. Section 3 presents verification diagrams, separated in the logic part and the combinatoric part. Sections 4, 5 and 6 introduce alternating automata, and show how they can be used to reduce the proof of an LTL property to a set of first-order verification conditions.

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} .

Transitions can be marked as *just* or *compassionate*. Just (or *weakly fair*) transitions cannot be continuously enabled without ever being taken. Compassionate (or *strongly fair*) transitions cannot be enabled infinitely often without being taken. Every compassionate transition is also just. A *computation* is a run that satisfies these fairness requirements. The set of all computations 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 simplicity of the presentation, we will omit the past temporal operators in this paper. However both verification diagrams and the alternating automata are applicable to LTL formulas that include past operators. 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 formula if it contains only state formulas, boolean connectives and future temporal operators. We say that a formula is a general safety formula if it is of the form $\Box \varphi$, 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 computations of \mathcal{S} .

2.3 Nondeterministic ω -automata

Verification diagrams are based on nondeterministic ω -automata. Automata are represented by a tuple $\mathcal{A} : \langle N, N_0, E, \nu, \mathcal{F} \rangle$, where N (N_0) are the (initial) nodes, E are the edges, ν is the node labeling, a function from the set of nodes to boolean expressions over atomic assertions, and \mathcal{F} is the acceptance condition, in our case a set of subsets of nodes, also known as a Muller acceptance condition.

An infinite sequence of nodes $\pi : n_0, n_1, \dots$ is called a path of an automaton \mathcal{A} if n_0 is an initial node, and for each $i > 0$, $\langle n_i, n_{i+1} \rangle \in E$. The set of nodes that appear infinitely often in π is called the limit set of π , written $\text{inf}(\pi)$. A path π is accepting if $\text{inf}(\pi) \in \mathcal{F}$. The set $\text{inf}(\pi)$ must necessarily form a *strongly connected subgraph* (SCS) in the automaton, that is each node in $\text{inf}(\pi)$ can be reached from every other node in $\text{inf}(\pi)$ without leaving this set.

A sequence of states s_0, s_1, \dots is a model of \mathcal{A} if there exists an accepting path n_0, n_1, \dots in \mathcal{A} such that for all $i \geq 0$ $s_i \models \nu(n_i)$. The set of models of \mathcal{A} is called the language of \mathcal{A} , written $\mathcal{L}(\mathcal{A})$.

3 Verification Diagrams

Verification diagrams are a complete proof method (relative to first-order reasoning) to prove arbitrary state quantified LTL properties over infinite-state systems.

Verification diagrams [MP94,BMS95,BMS96,MBSU98] are nondeterministic ω -automata augmented with an additional node labeling μ , a function from nodes to assertions over the program variables. A sequence of states s_0, s_1, \dots is a model of a verification diagram if there exists an accepting path $\pi : n_0, n_1, \dots$ in the diagram (that is, accepted by the underlying automaton) such that for every $i \geq 0$, $s_i \models \mu(n_i)$. The language of the diagram \mathcal{G} , written $\mathcal{L}(\mathcal{G})$, is the set of all its models. The underlying automaton of a diagram \mathcal{G} is denoted by \mathcal{G}_A .

3.1 Verification Diagrams: The Logic Part

To show that all computations of a system \mathcal{S} are included in the language of a diagram \mathcal{G} , that is, $\mathcal{L}(\mathcal{S}) \subseteq \mathcal{L}(\mathcal{G})$, we have to show

Initiation Every initial state of \mathcal{S} must be able to be mapped onto some initial state of the diagram. This holds if the following condition holds:

$$\Theta \rightarrow \mu(N_0)$$

where $\mu(S)$ with $S = \{n_1, \dots, n_k\} \subseteq N$ stands for

$$\mu(S) \stackrel{\text{def}}{=} \mu(n_1) \vee \dots \vee \mu(n_k)$$

It states that every run of \mathcal{S} can start at some initial node of \mathcal{G} .

Consecution For every node $n \in N$ and for every state $s \models \mu(n)$, every successor state of s must be able to be mapped onto a successor node of n . This holds if the following condition holds for every node $n \in N$:

$$\mu(n) \wedge \rho_\tau \rightarrow \mu'(succ(n))$$

where $succ(n)$ stands for all successor nodes of n .

Acceptance The acceptance condition of the automaton eliminates from the diagram language all sequences of states all of whose paths end up in a nonaccepting SCS. We have to show that none of these sequences correspond to a computation of the system. We say that an SCS S is *transient* if every computation of \mathcal{S} with a path ending in S has a way of leaving S . To show that every computation has at least one accepting path in the diagram it suffices to show that every nonaccepting SCS is transient[Sip99]. An SCS can be shown to be transient in one of the following three ways:

Just exit An SCS S has a just exit, if there is a just transition τ such that the following verification conditions hold for every node $m \in S$:

$$\mu(m) \rightarrow enabled(\tau)$$

and

$$\mu(m) \wedge \rho_\tau \rightarrow \mu'(succ(m) - S)$$

The first condition states that τ is enabled on every node, and the second condition ensures that the computation can leave the SCS at every node.

Compassionate exit An SCS has a compassionate exit, if there is a compassionate transition τ such that the following conditions hold for every node $m \in S$:

$$\mu(m) \rightarrow \neg enabled(\tau)$$

or

$$\mu(m) \wedge \rho_\tau \rightarrow \mu'(succ(m) - S)$$

and for some node $n \in S$, τ is enabled at n :

$$\mu(n) \rightarrow enabled(\tau)$$

This states that for every node in S either τ is disabled or τ can lead out of S , and there is at least one node n where τ can indeed leave S .

Well-founded SCS An SCS $S : \{n_1, \dots, n_k\}$ is *well-founded* if there exist ranking functions $\{\delta_1, \dots, \delta_k\}$, where each δ_i maps the system states into elements of a well-founded domain (\mathcal{D}, \succ) , such that the following verification conditions are valid: there is a *cut-set*¹ E of edges in S such that for all edges $(n_1, n_2) \in E$ and every transition τ ,

$$\mu(n_1) \wedge \rho_\tau \wedge \mu'(n_2) \rightarrow \delta_1 \succ \delta'_2 ,$$

and for all other edges $(n_1, n_2) \notin E$ in S and for all transitions τ ,

$$\mu(n_1) \wedge \rho_\tau \wedge \mu'(n_2) \rightarrow \delta_1 \succeq \delta'_2 .$$

This means that there is no computation that ends up in S : it would have to traverse at least one of the edges in E infinitely often, which contradicts the well-foundedness of the ranking functions.

In addition, we need to show that the language of the diagram is included in the language of the underlying automaton of the diagram, that is

$$\mathcal{L}(\mathcal{G}) \subseteq \mathcal{L}(\mathcal{G}_A)$$

This holds if the following first-order verification holds for every node n in \mathcal{G} :

$$\mu(n) \rightarrow \nu(n) .$$

Thus, if the above verification conditions hold it is ensured that every computation of the system is represented in the language of the underlying automaton of the diagram.

3.2 Verification Diagrams: The Automata Part

Having shown $\mathcal{L}(\mathcal{S}) \subseteq \mathcal{L}(\mathcal{G}_A)$ it remains to show that all models of the underlying automaton of the diagram satisfy the property, that is

$$\mathcal{L}(\mathcal{G}_A) \subseteq \mathcal{L}(\varphi) .$$

This check can be performed using a straightforward abstraction and standard ω -automata model checking.

Let $B = \{b_1, \dots, b_n\}$ be the set of first-order atomic formulas appearing in the property φ to be proven. Abstracting both the automaton for the negation of the property and the underlying automaton of the diagram with the abstraction function that in each node labeling replaces each atomic formula with the corresponding proposition of the boolean algebra over B , we obtain two finite-state ω -automata, \mathcal{G}_A^α and $\mathcal{A}^\alpha(\neg\varphi)$, and we can check the language inclusion $\mathcal{L}(\mathcal{G}_A^\alpha) \subseteq \mathcal{L}(\mathcal{A}^\alpha(\varphi))$ by checking $\mathcal{L}(\mathcal{G}_A^\alpha \times \mathcal{A}^\alpha(\neg\varphi))$ for emptiness.

¹ A cut-set of an SCS S is a set of edges E such that every loop in S contains some edge in E (that is, the removal of E disconnects S).

It is easy to show that the abstraction function and its corresponding concretization function (which replaces in each node labeling every proposition with the corresponding assertion) form a Galois insertion, and thus from

$$\mathcal{L}(\mathcal{G}_A^\alpha) \subseteq \mathcal{L}(\mathcal{A}^\alpha(\varphi))$$

we can conclude

$$\mathcal{L}(\mathcal{G}_A) \subseteq \mathcal{L}(\mathcal{A}(\varphi))$$

as required.

3.3 Verification Diagrams: Semi-Automatic Generation

As mentioned in the Introduction one can take the automaton for the property as a starting point for the verification diagram. The task at hand is now to refine the diagram by splitting nodes and strengthening the assertions labeling the nodes until the verification conditions associated with the diagram hold. If the diagram is refined in this manner, the combinatoric check becomes redundant, since the diagram is guaranteed to satisfy the property.

The disadvantage of this approach is that the diagram may get very large, since the size of the automaton is worst-case exponential in the size of the property. In the next section we introduce alternating automata, which are linear in the size of the property, to alleviate this problem to a certain extent.

4 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.

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

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

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. A *tree* is defined recursively as follows:

$$\begin{array}{ll}
T ::= \epsilon_T & \text{empty tree} \\
| T \cdot T & \text{composition} \\
| \langle \text{node}, T \rangle & \text{single node with child tree}
\end{array}$$

A tree may have both finite and infinite branches.

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:

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

A run T is *accepting* if every infinite branch contains infinitely many accepting nodes. 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 $\mathcal{L}(\mathcal{A})$.

5 Translating LTL formulas into Alternating Automata

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 :

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

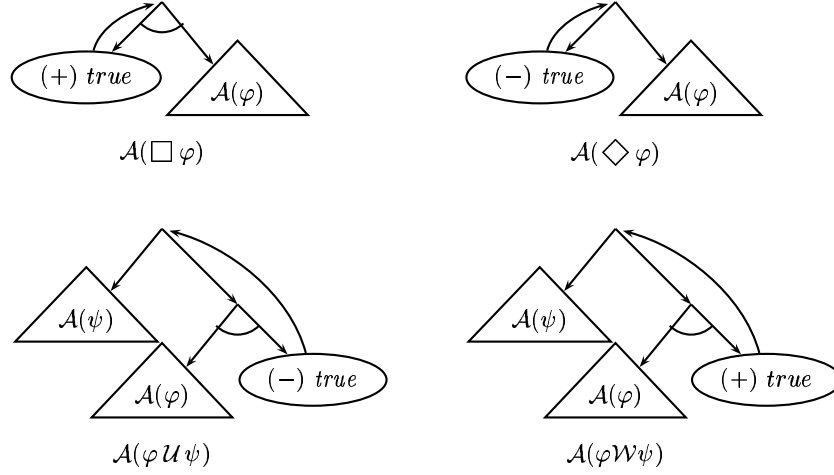


Fig. 1. Alternating automata for the temporal operators \square , \diamond , \mathcal{U} , \mathcal{W}

The constructions for the temporal formulas are illustrated in Figure 1.

In [MS00] it is shown that for a future temporal formula φ , $\mathcal{L}(\varphi) = \mathcal{L}(\mathcal{A}(\varphi))$.

6 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:

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

Intuitively, the initial condition of an automaton characterizes the set of initial states of sequences accepted by the automaton.

Basic Rule

Following the style of verification rules of [MP95] we can now present the basic temporal rule B-SAFE, shown in Figure 2. 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 $\tau \in \mathcal{T}$.

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. 2. Basic temporal rule B-SAFE

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 .

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 3 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. 3. 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.

In [MS00] we show that rule B-SAFE is sound, that is, 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 $\mathcal{S} \models \varphi$.

7 Implementation

Verification diagrams have been implemented in STeP, the Stanford Temporal Prover, a verification tool that supports algorithmic and deductive verification of reactive systems [BBC⁺95, BBC⁺00]. We are currently implementing support for interactive refinement and heuristics for automatic generation of verification diagrams.

The rule SAFE based on alternating automata has also been implemented in STeP, obviating the need for any specialized verification rules for safety properties. However, the strengthenings still have to be provided by the user.

Both verification diagrams and rule SAFE have been convenient in the proof of temporal properties, especially in proving properties of modular systems.

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.
- [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.
- [BMS96] A. Browne, Z. Manna, and H.B. Sipma. Hierarchical verification using verification diagrams. In *2nd Asian Computing Science Conf.*, vol. 1179 of *Lecture Notes in Computer Science*, pages 276–286. Springer-Verlag, December 1996.
- [Kur94] R.P. Kurshan. *Computer-Aided Verification of Coordinating Processes: The Automata-Theoretic Approach*. Princeton University Press, 1994.
- [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.

- [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.
- [MP94] Z. Manna and A. Pnueli. Temporal verification diagrams. 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 726–765. Springer-Verlag, 1994.
- [MP95] Z. Manna and A. Pnueli. *Temporal Verification of Reactive Systems: Safety*. Springer-Verlag, New York, 1995.
- [MS00] Z. Manna and H.B. Sipma. Alternating the temporal picture for safety. In U. Montanari, J.D. Rolim, and E. Welzl, editors, *Proc. 27th Intl. Colloq. Aut. Lang. Prog.*, vol. 1853, pages 429–450, Geneva, Switzerland, July 2000. Springer-Verlag.
- [Sip99] H.B. Sipma. *Diagram-based Verification of Discrete, Real-time and Hybrid Systems*. PhD thesis, Computer Science Department, Stanford University, February 1999. To appear as STAN-CS Technical Report.
- [SUM99] H.B. Sipma, T.E. Uribe, and Z. Manna. Deductive model checking. *Formal Methods in System Design*, 15(1):49–74, July 1999. Preliminary version appeared in *Proc. 8th Intl. Conference on Computer Aided Verification*, vol. 1102 of *LNCS*, Springer-Verlag, pp. 208–219, 1996.
- [Tho88] W. Thomas. Automata on infinite objects. Technical Report 88-17, RWTH Aachen, 1988. In *Handbook of Theoretical Computer Science*, North-Holland.
- [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.
- [VW86] M.Y. Vardi and P. Wolper. An automata-theoretic approach to automatic program verification. In *Proc. First IEEE Symp. Logic in Comp. Sci.*, pages 332–344, June 1986.