Alternating-time Temporal Logic*

Rajeev Alur† Thomas A. Henzinger‡ Orna Kupferman §

Abstract

Temporal logic comes in two varieties: *linear-time temporal logic* assumes implicit universal quantification over all paths that are generated by system moves; *branching-time temporal logic* allows explicit existential and universal quantification over all paths. We introduce a third, more general variety of temporal logic: *alternating-time temporal logic* offers selective quantification over those paths that are possible outcomes of games, such as the game in which the system and the environment alternate moves. While linear-time and branching-time logics are natural specification languages for closed systems, alternating-time logics are natural specification languages for open systems. For example, by preceding the temporal operator "eventually" with a selective path quantifier, we can specify that in the game between the system and the environment, the system has a strategy to reach a certain state. Also the problems of receptiveness, realizability, and controllability can be formulated as model-checking problems for alternating-time formulas.

Depending on whether we admit arbitrary nesting of selective path quantifiers and temporal operators, we obtain the two alternating-time temporal logics ATL and ATL*. We interpret the formulas of ATL and ATL* with respect to two models of composition for open systems, synchronous and asynchronous. For synchronous systems, the expressive power of ATL beyond CTL comes at no cost: the model-checking complexity of synchronous ATL is linear in the size of the system and the length of the formula. The symbolic model-checking algorithm for CTL extends with few modifications to synchronous ATL, and with some work, also to asynchronous ATL, whose model-checking complexity is quadratic. This makes ATL an obvious candidate for the automatic verification of open systems. In the case of ATL*, the model-checking problem is closely related to the synthesis problem for linear-time formulas, and requires doubly exponential time for both synchronous and asynchronous systems.

1 Introduction

In 1977, Pnueli proposed to use *linear-time temporal logic* (LTL) to specify requirements for reactive systems [Pnu77]. A formula of LTL is interpreted over a computation, which is an infinite sequence of states. A reactive system satisfies an LTL formula if all its computations do. Due to the implicit use of universal quantification over the set of computations, LTL cannot express existential, or possibility, properties. *Branching-time temporal logics*, such as CTL and CTL*, do provide explicit quantification over the set of computations [CE81, EH86]. For instance, for a state predicate φ, the CTL formula $\forall\Diamond\varphi$ requires that a state satisfying φ is visited in all computations, and the CTL formula $\exists\Diamond\varphi$ requires that there exists a computation that visits a state satisfying φ. The problem of *model checking* is to verify whether a finite-state abstraction of a reactive system satisfies a temporal-logic specification [CE81, QS81]. Efficient model checkers exist for both LTL (e.g. SPIN [Hol97]) and CTL (e.g. SMV [McM93]), and are increasingly being used as debugging aids for industrial designs.

The logics LTL and CTL have their natural interpretation over the computations of closed systems, where a *closed system* is a system whose behavior is completely determined by the state of the system. However, the compositional modeling and design of reactive systems requires each component to be viewed

*This research was supported in part by the ONR YIP award N00014-95-1-0520, by the NSF CAREER award CCR-9501708, by the NSF grant CCR-9504469, by the AFOSR contract F49620-93-1-0056, by the ARO MURI grant DAAH-04-96-1-0341, by the ARPA grant NAG2-892, and by the SRC contract 95-DC-324.036.

†Computer and Information Science Department, University of Pennsylvania, and Bell Labs, alur@cis.upenn.edu

‡Electrical Engineering and Computer Sciences Department, University of California, Berkeley, tah@eecs.berkeley.edu

§Electrical Engineering and Computer Sciences Department, University of California, Berkeley, orna@eecs.berkeley.edu

as an open system, where an *open system* is a system that interacts with its environment and whose behavior depends on the state of the system as well as the behavior of the environment. Models for open systems, such as CSP [Hoa85], I/O automata [Lyn96], and reactive modules [AH96], distinguish between *internal* nondeterminism, choices made by the system, and *external* nondeterminism, choices made by the environment. Consequently, besides universal (do all computations satisfy a property?) and existential (does some computation satisfy a property?) questions, a third question arises naturally: can the system resolve its internal choices so that the satisfaction of a property is guaranteed no matter how the environment resolves the external choices? Such an *alternating* satisfaction can be viewed as a winning condition in a two-player game between the system and the environment. Alternation is a natural generalization of existential and universal branching, and has been studied extensively in theoretical computer science [CKS81].

Different researchers have argued for game-like interpretations of LTL and CTL specifications for open systems. We list four such instances here. (1) *Receptiveness* [Dil89, AL93, GSSL94]: given a reactive system, specified by a set of *safe* computations (typically, generated by a transition relation) and a set of *live* computations (typically, expressed by an LTL formula), the receptiveness problem is to determine whether every finite safe computation can be extended to an infinite live computation irrespective of the behavior the environment. It is sensible, and necessary for compositionality, to require an affirmative answer to the receptiveness problem. (2) *Realizability (program synthesis)* [ALW89, PR89a, PR89b]: given an LTL formula ψ over sets of input and output signals, the synthesis problem requires the construction of a reactive system that assigns to every possible input sequence an output sequence so that the resulting computation satisfies ψ. (3) *Supervisory control* [RW89]: given a finite-state machine whose transitions are partitioned into controllable and uncontrollable, and a set of safe states, the control problem requires the construction of a controller that chooses the controllable transitions so that the machine always stays within the safe set (or satisfies some more general LTL formula). (4) *Module checking* [KV96]: given an open system and a CTL* formula φ, the module-checking problem is to determine if, no matter how the environment restricts the external choices, the system satisfies φ.

All the above approaches use the temporal-logic syntax that was developed for specifying closed systems, and reformulate its semantics for open systems. In this paper, we propose, instead, to enrich temporal logic so that alternating properties can be specified explicitly within the logic: we introduce *alternating-time temporal logics* for the specification and verification of open systems. Our formulation of open systems considers, instead of just a system and an environment, the more general setting of a set Σ of agents that correspond to different components of the system and the environment. For the scheduling of agents, we consider two policies. In each state of a *synchronous* system, it is known in advance which agent proceeds. In each state of an *asynchronous* system, several agents may be enabled, and an unknown scheduler determines which agent takes the next step. In the latter case, the scheduler is required to be fair to each agent; that is, in an infinite computation, an agent cannot be continuously enabled without being scheduled.

For a set $A \subseteq \Sigma$ of agents, a set Γ of computations, and a state w of the system, consider the following game between a protagonist and an adversary. The game starts at the state w. Whenever the scheduled agent is in the set A, the protagonist chooses the next state, and otherwise, the adversary chooses the next state. If the resulting infinite computation belongs to the set Γ, then the protagonist wins. If the protagonist has a winning strategy, we say that the alternating-time formula $\langle\langle A\rangle\rangle\Gamma$ is satisfied in the state w. Here, $\langle\langle A\rangle\rangle$ is a path quantifier, parameterized with the set A of agents, which ranges over all computations that the agents in A can force the game into, irrespective of how the agents in $\Sigma\setminus A$ play. Hence, the parameterized path quantifier $\langle\langle A\rangle\rangle$ is a generalization of the path quantifiers of branching-time temporal logics: the existential path quantifier $\exists$ corresponds to $\langle\langle \Sigma\rangle\rangle$, and the universal path quantifier $\forall$ corresponds to $\langle\langle \emptyset\rangle\rangle$. In particular, closed systems can be viewed as systems with a single agent *sys*. Then, the two possible parameterized path quantifiers $\langle\langle sys\rangle\rangle$ and $\langle\langle \emptyset\rangle\rangle$ match exactly the path quantifiers $\exists$ and $\forall$ required for specifying such systems. Depending on the syntax used to specify the set Γ of computations, we obtain two alternating-time temporal logics: in the logic ATL*, the set Γ is specified by a formula of LTL; in the more restricted logic ATL, the set Γ is specified by a single temporal operator applied to a state formula. Thus, ATL is the alternating generalization of CTL, and ATL* is the alternating generalization of CTL*.

Alternating-time temporal logics can conveniently express properties of open systems as illustrated by the following five examples:

1. In a multi-process distributed system, we can require any subset of processes to attain a goal, irrespective of the behavior of the remaining processes. Consider, for example, the cache-coherence protocol for Gigamax verified using SMV [McM93]. One of the desired properties is the absence of deadlocks, where a deadlocked state is one in which a processor, say a, is permanently blocked from accessing a memory cell. This requirement was specified using the CTL formula

   $$\forall\Box\,(\exists\Diamond\, \mathit{readable} \ \wedge\ \exists\Diamond\, \mathit{writable}).$$

   The ATL formula

   $$\forall\Box\,(\langle\!\langle a\rangle\!\rangle \Diamond \mathit{readable} \ \wedge\ \langle\!\langle a\rangle\!\rangle \Diamond \mathit{writable})$$

   captures the informal requirement more precisely. While the CTL formula only asserts that it is always possible for all processors to *cooperate* so that a can eventually read and write ("collaborative possibility"), the ATL formula is stronger: it guarantees a memory access for processor a, *no matter what the other processors in the system do* ("adversarial possibility").

2. While the CTL formula $\forall\Box\,\varphi$ asserts that the state predicate φ is an invariant of a system component irrespective of the behavior of all other components ("adversarial invariance"), the ATL formula $[\![a]\!]\Box\varphi$ (which stands for $\langle\!\langle \Sigma \setminus \{a\}\rangle\!\rangle\,\Box\varphi$) states the weaker requirement that φ is a *possible invariant* of the component a; that is, a cannot violate $\Box\varphi$ on its own, and therefore the other system components may cooperate to achieve $\Box\varphi$ ("collaborative invariance"). For φ to be an invariant of a complex system, it is necessary (but not sufficient) to check that every component a satisfies the ATL formula $[\![a]\!]\Box\varphi$.

3. The *receptiveness* of a system whose live computations are given by the LTL formula ψ is specified by the ATL* formula $\forall\Box\,\langle\!\langle \mathit{sys}\rangle\!\rangle\,\psi$.

4. Checking the *realizability (program synthesis)* of an LTL formula ψ corresponds to model checking of the ATL* formula $\langle\!\langle \mathit{sys}\rangle\!\rangle\psi$ in a maximal model that considers all possible inputs and outputs.

5. The *controllability* of a system whose safe states are given by the state predicate φ is specified by the ATL formula $\langle\!\langle \mathit{control}\rangle\!\rangle\,\Box\,\varphi$. Controller synthesis, then, corresponds to model checking of this formula. More generally, for an LTL formula ψ, the ATL* requirement $\langle\!\langle \mathit{control}\rangle\!\rangle\psi$ asserts that the controller has a strategy to ensure the satisfaction of ψ.

Notice that ATL is better suited for compositional reasoning than CTL. For instance, if a component a satisfies the CTL formula $\exists\Diamond\,\varphi$, we cannot conclude that the compound system $a\|b$ also satisfies $\exists\Diamond\,\varphi$. On the other hand, if a satisfies the ATL formula $\langle\!\langle a\rangle\!\rangle\Diamond\varphi$, then so does $a\|b$.

The model-checking problem for alternating-time temporal logics requires the computation of winning strategies. In the case of synchronous ATL, all games are *finite* reachability games. Consequently, the model-checking complexity is linear in the size of the system and the length of the formula, just as in the case of CTL. While checking existential reachability corresponds to iterating the existential next-time operator $\exists\bigcirc$, and checking universal reachability corresponds to iterating the universal next $\forall\bigcirc$, checking alternating reachability corresponds to iterating an appropriate mix of $\exists\bigcirc$ and $\forall\bigcirc$, as governed by a parameterized path quantifier. This suggests a simple symbolic model-checking procedure for synchronous ATL, and shows how existing symbolic model checkers for CTL can be modified to check ATL specifications, at no extra cost. In the asynchronous model, due to the presence of fairness constraints, ATL model checking requires the solution of *infinite* games, namely, generalized Büchi games [VW86]. Consequently, the model-checking complexity is quadratic in the size of the system, and the symbolic algorithm involves a nested fixed-point computation. The model-checking problem for ATL* is much harder: we show it to be complete for 2EXPTIME in both the synchronous and asynchronous cases.

2 The Alternating-time Logic ATL

2.1 Syntax

The temporal logic *ATL* (*alternating-time logic*) is defined with respect to a finite set Π of *propositions* and a finite set Σ of *agents*. An ATL formula is one of the following:

(S1) p, for propositions $p \in \Pi$.

(S2) $\neg\varphi$ or $\varphi_1 \vee \varphi_2$, where φ, φ_1, and φ_2 are ATL formulas.

(S3) $\langle\langle A\rangle\rangle \bigcirc \varphi$ or $\langle\langle A\rangle\rangle \varphi_1 \mathcal{U} \varphi_2$, where $A \subseteq \Sigma$ is a set of agents, and φ, φ_1, and φ_2 are ATL formulas.

The operator $\langle\langle\ \rangle\rangle$ is a *path quantifier*, and $\bigcirc$ ("next") and $\mathcal{U}$ ("until") are *temporal operators*. The logic ATL is similar to the branching-time logic CTL, only that path quantifiers are parameterized by sets of agents. When $A = \{a_1, \ldots, a_n\}$ is known, we write $\langle\langle a_1, \ldots, a_n\rangle\rangle$ instead of $\langle\langle \{a_1, \ldots, a_n\}\rangle\rangle$.

2.2 Synchronous-structure semantics

The formulas of ATL can be interpreted over a *synchronous structure* $S = \langle \Pi, \Sigma, W, R, \pi, \sigma\rangle$, where Π is the set of propositions, Σ is the set of agents, W is a set of states, $R \subseteq W \times W$ is a total transition relation (i.e., for every state $w \in W$ there exists a state $w' \in W$ such that $R(w, w')$), the function $\pi\colon W \to 2^\Pi$ maps each state $w \in W$ to the set $\pi(w) \subseteq \Pi$ of propositions that are true in w, and the function $\sigma\colon W \to \Sigma$ maps each state $w \in W$ to the agent $\sigma(w) \in \Sigma$ that is enabled in w. Note that precisely one agent is enabled in each state. For a set $A \subseteq \Sigma$ of agents, we denote by $W_A \subseteq W$ the set of states w for which $\sigma(w) \in A$. When $A = \{a\}$ is a singleton, we write W_a instead of $W_{\{a\}}$. For two states w and w' with $R(w, w')$, we say that w' is a *successor* of w. A *computation* of S is an infinite sequence $\gamma = w_0, w_1, w_2, \ldots$ of states such that for all $i \geq 0$, we have $R(w_i, w_{i+1})$. We refer to a computation starting at state w_0 as a w_0-*computation*. For a computation γ and an index $i \geq 0$, we use $\gamma[i]$ and $\gamma[0, i]$ to denote the i-th state in γ and the finite prefix $w_0, w_1, \ldots, w_i$ of γ, respectively.

In order to define the semantics of ATL, we first define the notion of strategies. A *strategy* for an agent $a \in \Sigma$ is a function $f_a : W^* \cdot W_a \to W$ such that if $f_a(\gamma \cdot w) = w'$, for $\gamma \in W^*$ and $w \in W_a$, then $R(w, w')$. Thus, a strategy maps a finite prefix of a computation to a possible extension of the computation. Intuitively, the strategy f_a suggests for each history γ and state w in which a is enabled, a successor $f_a(\gamma \cdot w)$ of w. Each strategy f_a induces a set of computations that agent a can enforce. Given a state w, a set $A = \{a_1, \ldots, a_n\}$ of agents, and a set $F_A = \{f_{a_1}, \ldots, f_{a_n}\}$ of strategies for the agents in A, we define the *outcomes* of F_A from w to be the set $out(w, F_A)$ of all w-computations that the agents in A can enforce when they cooperate and follow the strategies in F_A; that is, a w-computation γ is in $out(w, F_A)$ iff whenever γ visits a state whose agent is $a_k \in A$, then the computation proceeds according to the strategy f_{a_k}. Formally, $\gamma = w_0, w_1, \ldots$ is in $out(w, F_A)$ iff (1) $w_0 = w$, and (2) for all $i \geq 0$, if $\sigma(w_i) = a_k$ for some $a_k \in A$, then $w_{i+1} = f_{a_k}(\gamma[0, i])$. In particular, if $A = \Sigma$, then the outcome set $out(w, F_A)$ contains a single w-computation, and if $A = \emptyset$, then the outcome set $out(w, F_A)$ contains all w-computations.

We can now turn to a formal definition of the semantics of ATL. We write $w \models_S \varphi$ to indicate that the state w of the synchronous structure S satisfies the ATL formula φ (when the subscript S is clear from the context, we omit it). The satisfaction relation $\models$ is defined, for all states $w \in W$, inductively as follows:

- For $p \in \Pi$, we have $w \models p$ iff $p \in \pi(w)$.
- $w \models \neg\varphi$ iff not $w \models \varphi$.
- $w \models \varphi_1 \vee \varphi_2$ iff $w \models \varphi_1$ or $w \models \varphi_2$.
- $w \models \langle\langle A\rangle\rangle \bigcirc \varphi$ iff there exists a set F_A of strategies, one for each agent in A, such that for all computations $\gamma \in out(w, F_A)$, we have $\gamma[1] \models \varphi$.
- $w \models \langle\langle A\rangle\rangle \varphi_1 \mathcal{U} \varphi_2$ iff there exists a set F_A of strategies, one for each agent in A, such that for all computations $\gamma \in out(w, F_A)$ there exists an index $i \geq 0$ such that $\gamma[i] \models \varphi_2$ and for all $0 \leq j < i$, we have $\gamma[j] \models \varphi_1$.

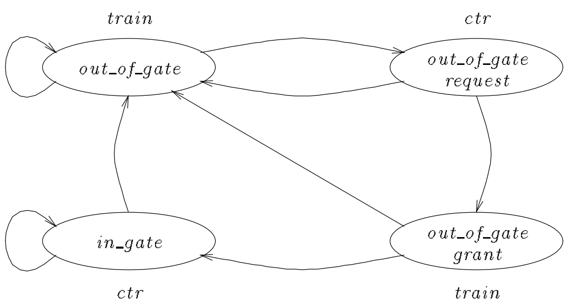

Figure 1: A synchronous train-controller system

Note that the next operator $\langle\langle A\rangle\rangle\bigcirc$ gives a local constraint: $w \models \langle\langle A\rangle\rangle\bigcirc\varphi$ iff either $w \in W_A$ and there exists a successor of w that satisfies φ, or $w \notin W_A$ and all successors of w satisfy φ. For an ATL formula φ, we denote by $[\varphi] \subseteq W$ the set of states w such that $w \models \varphi$.

Since the parameterized path quantifiers $\langle\langle\Sigma\rangle\rangle$ and $\langle\langle\emptyset\rangle\rangle$ correspond to existential and universal path quantification, respectively, we write $\exists$ for $\langle\langle\Sigma\rangle\rangle$ and $\forall$ for $\langle\langle\emptyset\rangle\rangle$. The logic CTL is the fragment of ATL interpreted over structures with a single agent: if $\Sigma = \{sys\}$, then $\langle\langle sys\rangle\rangle = \exists$ and $\langle\langle\emptyset\rangle\rangle = \forall$. As dual of $\langle\langle\ \rangle\rangle$ we use $[\![\ ]\!]$: define $[\![A]\!] = \langle\langle\Sigma \setminus A\rangle\rangle$. Since the path quantifier $\langle\langle\Sigma \setminus A\rangle\rangle$ ranges over the computations that the agents not in A have strategies to enforce, the path quantifier $[\![A]\!]$ ranges over the computations that the agents in A do not have strategies to avoid. Therefore also $\langle\langle A\rangle\rangle = [\![\Sigma \setminus A]\!]$. As in CTL, the temporal operators $\Diamond$ ("eventually") and $\Box$ ("always") are defined from the until operator: $\langle\langle A\rangle\rangle\Diamond\varphi = \langle\langle A\rangle\rangle \mathit{true}\, \mathcal{U} \varphi$ and $\langle\langle A\rangle\rangle\Box\varphi = \neg[\![A]\!]\Diamond\neg\varphi$.

Example 2.1 Consider the synchronous structure shown in Figure 1. The structure describes a protocol for a train entering a gate at a railroad crossing. At each moment, the train is either *out_of_gate* or *in_gate*. In order to enter the gate, the train issues a request, which is serviced (granted or rejected) by the controller in the next step. After a grant, the train may enter the gate or relinquish the grant. The structure has two agents: the train and the controller. Two states of the structure, labeled *ctr*, are controlled; that is, when a computation is in one of these states, the controller chooses the next state. The other two states are not controlled, and the train chooses successor states. The system satisfies the following specifications:

1. Whenever the train is outside the gate and does not have a grant to enter the gate, the controller can prevent it from entering the gate:

$$\forall\Box\,((\mathit{out\_of\_gate} \wedge \neg \mathit{grant}) \rightarrow \langle\langle \mathit{ctr}\rangle\rangle\Box \mathit{out\_of\_gate})$$

2. Whenever the train is outside the gate, the controller cannot force it to enter the gate:

$$\forall\Box\,(\mathit{out\_of\_gate} \rightarrow [\![\mathit{ctr}]\!]\Box \mathit{out\_of\_gate})$$

3. Whenever the train is outside the gate, the train and the controller can cooperate so that the train will enter the gate:

$$\forall\Box\,(\mathit{out\_of\_gate} \rightarrow \langle\langle \mathit{ctr}, \mathit{train}\rangle\rangle\Diamond \mathit{in\_gate})$$

4. Whenever the train is in the gate, the controller can force it out in a single step:

$$\forall\Box\,(\mathit{in\_gate} \rightarrow \langle\langle \mathit{ctr}\rangle\rangle\bigcirc \mathit{out\_of\_gate})$$

The first two specifications cannot be stated in CTL or CTL*. They provide more information than the CTL formula $\varphi = \forall\Box\,(\mathit{out\_of\_gate} \rightarrow \exists\Box\, \mathit{out\_of\_gate})$. While φ only requires the existence of a computation in which the train is always outside the gate, the ATL formulas guarantee that no matter how the train behaves, the controller can prevent it from entering the gate, and no matter how the controller behaves, the train can decide to stay outside the gate. Since all the states of the structure are either controller states or train states, the third specification is equivalent to the CTL formula $\forall\Box\,(\mathit{out\_of\_gate} \rightarrow \exists\Diamond\, \mathit{in\_gate})$. $\Box$

2.3 Asynchronous-structure semantics

The formulas of ATL can also be interpreted over an *asynchronous structure* $S = \langle \Pi, \Sigma, W, R, \pi, \sigma\rangle$, where Π, Σ, R, and L are as in a synchronous structure, and the function $\sigma\colon R \rightarrow \Sigma$ maps each transition $r \in R$ to the agent $\sigma(r) \in \Sigma$ that owns r. For an agent $a \in \Sigma$, we denote by $R_a \subseteq R$ the set of transitions owned by a. For two states w and w' with $R_a(w, w')$, we say that w' is an *a-successor* of w. Note that a state w

may have none or several a-successors. For a state w, we define $enabled(w) \subseteq \Sigma$ to be the set of agents a for which there exists an a-successor of w. We write W_a for the set of states w with $a \in enabled(w)$. As for synchronous structures, a *computation* of S is an infinite sequence $\gamma = w_0, w_1, w_2, \ldots$ of states such that for all $i \geq 0$, we have $R(w_i, w_{i+1})$. Given a set $A \subseteq \Sigma$ of agents, the computation γ is *A-fair* iff for each agent $a \in A$ and each index $i \geq 0$, there exists a $j \geq i$ such that either $a \notin enabled(w_j)$ or $R_a(w_j, w_{j+1})$; that is, in an A-fair computation, no agent in A can be continuously enabled without being scheduled.[1]

Similar to the synchronous case, a *strategy* for an agent $a \in \Sigma$ is a function $f_a : W^* \cdot W_a \to W$ such that if $f_a(\gamma \cdot w) = w'$, for $\gamma \in W^*$ and $w \in W_a$, then $R_a(w, w')$. Thus, the agent a applies its strategy and influences the computation whenever it is scheduled. However, unlike in synchronous structures, it is not known in advance which agent is scheduled when a particular state is encountered. Given a state w, a set $A = \{a_1, \ldots, a_n\}$ of agents, and a set $F_A = \{f_{a_1}, \ldots, f_{a_n}\}$ of strategies for the agents in A, we define the *outcomes* $out(w, F_A)$ to be the set of all w-computations that the agents in A can enforce when they follow the strategies $f_{a_1}, \ldots, f_{a_n}$ and the scheduling policy is fair with respect to A. Formally, the computation $\gamma = w_0, w_1, \ldots$ is in $out(w, F_A)$ iff (1) $w_0 = w$, (2) for all $i \geq 0$, if $\sigma(w_i, w_{i+1}) = a_k$ for some $a_k \in A$, then $w_{i+1} = f_{a_k}(\gamma[0, i])$, and (3) γ is A-fair. The definition of satisfaction of ATL formulas in an asynchronous structure is the same as in the synchronous case, with the above definition of outcomes. For example, $w \models \langle\langle A \rangle\rangle \bigcirc \varphi$ iff for every agent $a \in enabled(w)$, either $a \in A$ and there exists an a-successor w' of w such that $w' \models \varphi$, or $a \notin A$ and for all a-successors w' of w, we have $w' \models \varphi$. Thus, synchronous ATL is the fragment of asynchronous ATL interpreted over structures where for each state w, the set $enabled(w)$ of enabled agents is a singleton. If the set Σ of agents is a singleton, then the synchronous and asynchronous interpretations coincide (and are equal to CTL).

Example 2.2 Consider the asynchronous structure shown in Figure 2. The structure again describes a protocol for a train entering a gate. The protocol is similar to the one described in Example 2.1, only that requests by the train to enter the gate are serviced asynchronously, at some future step. A fair scheduling policy ensures that each request will be serviced (granted or rejected) eventually. All four specifications from Example 2.1 hold also for the asynchronous system. □

[1] Our algorithms can be easily modified to account for different, stronger types of fairness.

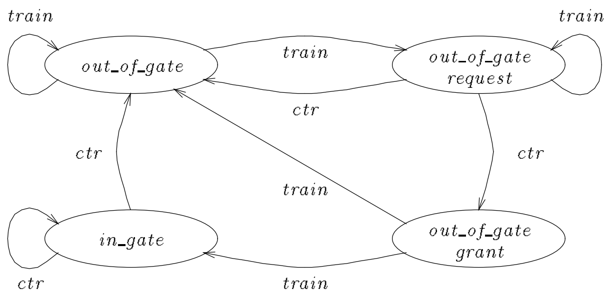

Figure 2: An asynchronous train-controller system

3 ATL Model Checking

The *synchronous* (resp. *asynchronous*) *model-checking problem* for ATL asks, given a synchronous (asynchronous) structure S and an ATL formula φ, for the set $[\varphi]$ of states of S that satisfy φ. We measure the complexity of the model-checking problem in two ways: the *joint complexity* of model checking considers the complexity in terms of both the size of the structure and the length of the formula; the *structure complexity* of model checking considers the complexity in terms of the structure only, assuming the formula is fixed. Since the structure is typically much larger than the formula, and its size is the most common computational bottle-neck, the structure-complexity measure is of particular practical interest [LP85].

3.1 The synchronous model

Model checking for synchronous ATL is very similar to CTL model checking [CE81, QS81, BCM+90]. We first present a *symbolic* algorithm, which manipulates state sets of the given synchronous structure S. The algorithm is shown in Figure 3, and uses the following primitive operations:

- The function *Sub*, when given an ATL formula φ, returns a queue $Sub(\varphi)$ of subformulas of φ such that if φ_1 is a subformula of φ and φ_2 is a subformula of φ_1, then φ_2 precedes φ_1 in the queue $Sub(\varphi)$.

- The function *PropCheck*, when given a proposition p, returns the state set $[p]$.

- The function *Pre*, when given two state sets ρ and τ, returns the set of states w such that either $w \in \rho$ and some successor of w is in τ, or $w \notin \rho$ and all successors of w are in τ.

- Union, intersection, difference, and inclusion test for state sets.

These primitives can be implemented using symbolic representations, such as binary decision diagrams, for state sets and the transition relation. If given a symbolic model checker for CTL, such as SMV [McM93], only the *Pre* operation needs to be modified for checking ATL.

for each φ' **in** $Sub(\varphi)$ **do**
  case $\varphi' = p$: $[\varphi'] := PropCheck(p)$
  case $\varphi' = \neg\theta$: $[\varphi'] := [true] \setminus [\theta]$
  case $\varphi' = \theta_1 \vee \theta_2$: $[\varphi'] := [\theta_1] \cup [\theta_2]$
  case $\varphi' = \langle\langle A\rangle\rangle \bigcirc \theta$: $[\varphi'] := Pre(W_A, [\theta])$
  case $\varphi' = \langle\langle A\rangle\rangle \theta_1 \,\mathcal{U}\, \theta_2$:
    $\rho := [false]$; $\tau := [\theta_2]$;
    while $\tau \not\subseteq \rho$ **do**
      $\rho := \rho \cup \tau$; $\tau := Pre(W_A, \rho) \cap [\theta_1]$
    od;
    $[\varphi'] := \tau$
  end case
od;
return $[\varphi]$.

Figure 3: Symbolic ATL model checking

Alternatively, the ATL model-checking algorithm can be implemented using an *enumerative* representation for the state set W of the structure S. Then, for each subformula φ' of φ, every state $w \in W$ is labeled with φ' iff $w \in [\varphi']$, and w is labeled with $\neg\varphi'$ iff $w \notin [\varphi']$. The labeling of the states with formulas proceeds in a bottom-up fashion, following the ordering $Sub(\varphi)$ from smaller to larger subformulas. If φ' is generated by the rules (S1) or (S2) or has the form $\langle\langle A\rangle\rangle \bigcirc \theta$, the labeling procedure is straightforward. For $\varphi' = \langle\langle A\rangle\rangle \theta_1 \,\mathcal{U}\, \theta_2$, the labeling procedure corresponds to solving a reachability problem for an AND-OR graph: the states in W_A are OR-nodes, the remaining states are AND-nodes, and we need to compute the set of nodes from which the OR-player can reach a state labeled by θ_2 while staying within states labeled by θ_1. Since the reachability problem for AND-OR graphs can be solved in time linear in the number of edges, it follows that the labeling procedure requires linear time for each subformula. Furthermore, since reachability for AND-OR graphs, a PTIME-hard problem [Imm81], can be reduced to model checking for synchronous ATL, we conclude the following theorem.

Theorem 3.1 *The synchronous model-checking problem for ATL is PTIME-complete, and can be solved in time $O(m\ell)$ for a structure with m transitions and an ATL formula of length ℓ. The structure complexity of the problem is also PTIME-complete.*

It is interesting to compare the model-checking complexities of synchronous ATL and CTL. While both problems can be solved in time $O(m\ell)$ [CES86], the structure complexity of CTL model checking is only NLOGSPACE-complete [BVW94]. This is because CTL model checking is related to graph reachability, as synchronous ATL model checking is related to AND-OR graph reachability.

3.2 The asynchronous model

Consider an asynchronous structure S with the state set W, and a formula φ of ATL. As in the synchronous case, for each subformula φ' of φ, we compute the set $[\varphi']$ of states that satisfy φ', starting from the innermost subformulas. For this purpose, we transform the asynchronous structure S into a synchronous structure S' as follows. The propositions of S' are the propositions of S. The agents of S' are the agents of S, plus a new agent b called *scheduler*. The states of S' are the states of S, plus for every state $w \in W$ and every agent a enabled in w, a new state $q_{w,a}$. Then, each transition of S from w to w' owned by agent a is replaced in S' by two transitions, one from w to

$q_{w,a}$ and the other from $q_{w,a}$ to w'. The propositions that are true in the states w and $q_{w,a}$ of S' are those that are true in the state w of S. The agent that is enabled in the state w of S' is b, and the agent that is enabled in the state $q_{w,a}$ of S' is a. Thus, in the synchronous structure, first the scheduler chooses one of the enabled agents, and then the chosen agent takes a step.

Consider the subformula $\langle\!\langle a \rangle\!\rangle \Diamond \theta$ of φ (more general until formulas can be handled similarly). The evaluation of this formula in a state $w_0 \in W$ corresponds to the following game between a protagonist and an adversary. The game is played on the synchronous structure S', starting from w_0. When the agent enabled in the current state is a, the protagonist chooses a successor state in S'; otherwise the adversary chooses the next state. When a state in W satisfying θ is visited, the protagonist wins. If the game continues forever, then the adversary wins iff the resulting computation is fair to the agent a; that is, it contains either infinitely many states of the form $q_{w,a}$ or infinitely many states $w \in W$ such that a is not enabled in the state w of S. Then, the state w_0 satisfies the formula $\langle\!\langle a \rangle\!\rangle \Diamond \theta$ in S iff the protagonist has a winning strategy. The winning condition of the adversary can be specified by the LTL formula $\Box(\neg\theta \wedge \Diamond(W'_a \vee \neg W_a))$, where W_a defines the states of S in which a is enabled and W'_a defines the states of S' in which a is enabled. This is a Büchi game, and the set of winning states for the adversary can be computed by a nested fixed-point computation:

$\rho := [true]$; $\tau := [\neg\theta]$;
while $\rho \not\subseteq \tau$ **do**
  $\rho := \rho \cap \tau$; $\rho' := [false]$;
  $\tau := (W'_a \cup (W \setminus W_a)) \cap \rho \cap [\neg\theta]$;
  while $\tau \not\subseteq \rho'$ **do**
    $\rho' := \rho' \cup \tau$; $\tau := Pre(S', W' \setminus W'_a, \rho') \cap [\neg\theta]$
  od
od.

In an enumerative implementation, the complexity of solving a Büchi game is quadratic in the number of transitions of a structure [VW86]. If S has m transitions, then S' has $2m$ transitions. Thus, the labeling procedure for the subformula $\langle\!\langle a \rangle\!\rangle \Diamond \theta$ requires $O(m^2)$ time.

For the subformula $\varphi' = \langle\!\langle a_1, \ldots, a_n \rangle\!\rangle \Diamond \theta$, the winning condition of the adversary is a conjunction of n Büchi conditions (to be precise, $\Box(\neg\theta \wedge \bigwedge_{1 \leq k \leq n} \Diamond(W'_{a_k} \vee \neg W_{a_k}))$. Such a game can be transformed into a game with a single Büchi condition by introducing a counter variable that ranges over the set $\{1, \ldots, n\}$ [VW86]. Hence, the labeling procedure for the subformula φ' requires $O(m^2 n^2)$ time. To determine the complexity of evaluating all subformulas of φ, we define the size of a logical connective to be 1, and the size of a temporal connective to be the number of agents in the corresponding path quantifier. Then, if $n_1, \ldots, n_k$ are the sizes of all connectives in φ, the overall time complexity is $O(m^2 n_1^2 + \cdots + m^2 n_k^2)$, or $O(m^2 \ell^2)$, because $n_1 + \cdots + n_k$ is bounded by the length ℓ of the formula φ. Finally, since the synchronous model-checking problem is a special case of the asynchronous problem, we conclude the following theorem.

Theorem 3.2 *The asynchronous model-checking problem for ATL is PTIME-complete, and can be solved in time $O(m^2\ell^2)$ for a structure with m transitions and an ATL formula of length ℓ. The structure complexity of the problem is also PTIME-complete.*

It is interesting to compare the model-checking complexities of asynchronous ATL and Fair-CTL, with generalized Büchi fairness constraints. While the latter is can be solved in $O(mk\ell)$ time [VW86], where k is the number of Büchi constraints, the best known algorithm for the former is quadratic in m. This is because Fair-CTL model checking is related to checking the emptiness of Büchi automata, and asynchronous ATL model checking is related to checking the emptiness of alternating Büchi automata.

4 The Alternating-time Logic ATL*

The logic ATL is a fragment of a more expressive logic called ATL*. There are two types of formulas in ATL*: *state formulas*, whose satisfaction is related to a specific state, and *path formulas*, whose satisfaction is related to a specific computation. Formally, a state formula is one of the following:

(S1) p, for propositions $p \in \Pi$.

(S2) $\neg\varphi$ or $\varphi_1 \vee \varphi_2$, where φ, φ_1, and φ_2 are state formulas.

(S3) $\langle\langle A\rangle\rangle\psi$, where $A \subseteq \Sigma$ is a set of agents, and ψ is a path formula.

A path formula is one of the following:

(P1) φ, for state formulas φ.

(P2) $\neg\psi$ or $\psi_1 \vee \psi_2$, where ψ, ψ_1, and ψ_2 are path formulas.

(P3) $\bigcirc\psi$ or $\psi_1 \mathcal{U} \psi_2$, where ψ, ψ_1, and ψ_2 are path formulas.

The logic ATL* consists of the set of state formulas generated by the above rules. It is similar to the branching-time temporal logic CTL*, only that path quantifiers are parameterized by sets of agents. The logic ATL is the fragment of ATL* that consists of all formulas in which every temporal operator is immediately preceded by a path quantifier.

The semantics of ATL* is defined similarly to the semantics for ATL. We write $\gamma \models_S \varphi$ to indicate that the computation γ of the structure S satisfies the path formula φ (the subscript S is usually omitted). The satisfaction relation $\models$ is defined, for all states w and computations γ, inductively as follows:

- For state formulas generated by the rules (S1) and (S2), the definition of $\models$ is the same as for ATL.
- $w \models \langle\langle A\rangle\rangle\psi$ iff there exists a set F_A of strategies, one for each agent in A, such that for all computations $\gamma \in out(w, F_A)$, we have $\gamma \models \psi$.
- $\gamma \models \varphi$ for a state formula φ iff $\gamma[0] \models \varphi$.
- $\gamma \models \neg\psi$ iff not $\gamma \models \psi$.
- $\gamma \models \psi_1 \vee \psi_2$ iff $\gamma \models \psi_1$ or $\gamma \models \psi_2$.
- $\gamma \models \bigcirc\psi$ iff $\gamma[1] \models \psi$.
- $\gamma \models \psi_1 \mathcal{U} \psi_2$ iff there exists an index $i \geq 0$ such that $\gamma[i] \models \psi_2$ and for all $0 \leq j < i$, we have $\gamma[j] \models \psi_1$.

As before, $\exists = \langle\langle\Sigma\rangle\rangle$ and $\forall = \langle\langle\emptyset\rangle\rangle$ and $[\![A]\!] = \langle\langle\Sigma \setminus A\rangle\rangle$. The temporal operators $\Diamond$ and $\Box$ are defined from $\mathcal{U}$ as usual: $\Diamond\psi = true\,\mathcal{U}\psi$, and $\Box\psi = \neg\Diamond\neg\psi$. For example, the ATL* formula $\forall\Box\,\langle\langle a\rangle\rangle((\Box req) \rightarrow (\Box\Diamond grant))$ asserts that agent a has a strategy to enforce that, whenever a request is continuously issued, infinitely many grants are given. This specification cannot be expressed in CTL* or in ATL. For single-agent structures, ATL* degenerates to CTL*.

While there is an exponential price to be paid in model-checking complexity when moving from CTL to CTL*, this price becomes even more significant when we consider the alternating-time versions of the logics.

Theorem 4.1 *In both the synchronous and asynchronous cases, the model-checking problem for ATL* is 2EXPTIME-complete. In both cases, the structure complexity of the problem is PTIME-complete.*

Proof (sketch). Given a synchronous structure S with state set W, and an ATL* formula φ, we label the states in S with state subformulas of φ, starting from the innermost state subformulas. For state subformulas of the form $\langle\langle A\rangle\rangle\psi$, we employ the algorithm for CTL* module checking from [KV96] as follows. Let ψ' the result of replacing in ψ all state subformulas, which have already been evaluated, by appropriate new propositions. For a state w and a set F_A of strategies, the set $out(w, F_A)$ of computations induces a tree, which is obtained by unwinding S from w and then pruning all subtrees whose roots are successors of states in W_A that are not chosen by the strategies in F_A. We construct a Büchi tree automaton $T_{w,A}$ that accepts all trees induced by $out(w, F_A)$ for any set F_A of strategies, and a Rabin tree automaton T_ψ that accepts all trees satisfying the CTL* formula $\forall\psi'$. By [KV96], $T_{w,A}$ has $|W|$ states. By [ES84], T_ψ has $2^{2^{O(|\psi|)}}$ states and $2^{O(|\psi|)}$ acceptance pairs. The intersection of the two automata is a Rabin tree automaton that contains the outcome trees satisfying $\forall\psi'$. Hence, by the semantics of ATL*, the state w satisfies $\langle\langle A\rangle\rangle\psi$ iff this intersection is nonempty. By [EJ88, PR89a], the nonemptiness problem for a Rabin tree automaton with n states and k pairs can be solved in time $O(kn)^{3n}$. Hence, evaluating the subformula $\langle\langle A\rangle\rangle\psi$ in a single state requires at most time $|W|^{2^{O(|\psi|)}}$. Since there are $|W|$ states and $O(|\varphi|)$ many subformulas, membership in 2EXPTIME follows.

The asynchronous case can be reduced to the synchronous case similar to the proof of Theorem 3.2. For the lower bounds, we use a reduction from the realizability problem of LTL, a 2EXPTIME-hard problem [Ros92], to model checking for synchronous ATL*. $\Box$

By contrast, CTL* model checking is only PSPACE-complete [CES86], and its structure complexity is NLOGSPACE-complete [BVW94].

5 Discussion

The verification problem for open systems, more than it corresponds to the model-checking problem for temporal logics, corresponds, in the case of linear time, to the realizability problem [ALW89, PR89a, PR89b], and in the case of branching time, to the module-checking problem [KV96]; that is, to a search for winning strategies of ω-regular games. In general, this involves a hefty computational price. The logic ATL identifies an interesting class of properties that can be checked by solving finite games, which demonstrates that there is still a great deal of reasoning about open systems that can be performed efficiently. We conclude with several remarks on variations of ATL which support our design choices.

5.1 Agents with limited memory

In the definitions of ATL and ATL*, the strategy of an agent may depend on an unbounded amount of information, namely, the full history of the game up to the current state. However, since all involved games are ω-regular, the existence of a winning strategy implies the existence of a winning *finite-state* strategy [Tho95], which depends only on a finite amount of information about the history of the game. Thus, the semantics of ATL and ATL* can be defined, equivalently, using the outcomes of finite-state strategies only. This is interesting, because a strategy can be thought of as the parallel composition of the system with a *controller*, which makes sure that the system follows the strategy. Then, for an appropriate definition of parallel composition, it is precisely the finite-state strategies that can be implemented using controllers that are synchronous structures. Indeed, for the finite reachability games and generalized Büchi games of ATL, it suffices to consider *memory-free* strategies [Tho95], which can be implemented as control maps (i.e., controllers without state). This is not the case for ATL*, whose formulas can specify the winning positions of Streett games [Tho95].

5.2 Agents with limited information

Our models assume that the agents of a structure have complete information about all propositions (which state satisfies which propositions) and about the other agents (which agent owns which transitions). Sometimes it may be more appropriate to assume that each agent $a \in \Sigma$ can observe only a subset $\Pi_a \subseteq \Pi$ of the propositions, and a strategy $f_a : \Pi_a^* \to \Pi_a$ for a must (1) depend only on the observable part of the history and (2) decide only on the observable part of the next state. From undecidability results on multi-player games with incomplete information, it follows that the model-checking problem for ATL *with incomplete information* is undecidable in both the synchronous [Yan97] and asynchronous [PR79] cases. In the special case that all path quantifiers are parameterized by single agents, and no cooperation between agents with different information is possible, decidability follows from the results on module checking with incomplete information [KV97]. In this case, the model-checking complexity for both synchronous and asynchronous ATL is EXPTIME-complete, and 2EXPTIME-complete for ATL*. The structure complexity of all four problems is EXPTIME-complete, thus rendering reasoning about agents with incomplete information infeasible even under severe restrictions.

5.3 Game logic and module checking

The parameterized path quantifier $\langle\langle A\rangle\rangle$ first stipulates the *existence* of strategies for the agents in A and then *universally* quantifies over the outcomes of the stipulated strategies. One may generalize ATL and ATL* by separating the two concerns into strategy quantifiers and path quantifiers, say, by writing $\exists A.\,\forall$ instead of $\langle\langle A\rangle\rangle$ (read $\exists A$ as "there exist strategies for the agents in A"). Then, for example, the formula $\varphi = \exists A.\,(\exists\Box\,\varphi_1 \wedge \exists\Box\,\varphi_2)$ asserts that the agents in A have strategies such that for some behavior of the remaining agents, φ_1 is always true, and for some possibly different behavior of the remaining agents, φ_2 is always true. To define the semantics of strategy quantifiers, we need to consider the tree that is induced by the outcomes of a set of strategies, and obtain three types of formulas: state formulas and path formulas as in CTL* or ATL*, and *tree formulas*, whose satisfaction is related to a specific outcome tree. For instance, while φ is a state formula, its subformula $\exists\Box\,\varphi_1 \wedge \exists\Box\,\varphi_2$ is a tree formula. We refer to the general logic with strategy quantifiers, path quantifiers, temporal operators, and boolean connectives as *game logic*. Then, ATL* is the fragment of game logic that consists of all formulas in which every strategy quantifier is immediately followed by a path quantifier (note that $\exists\!\!\exists A.\,\exists$ is equivalent to $\exists$).

Another fragment of game logic is studied in *module checking* [KV96]. There, one considers formulas of the form $\exists\!\!\exists A.\,\theta$, with a single outermost strategy quantifier followed by a CTL or CTL* formula θ. From an expressiveness viewpoint, alternating-time logics and module checking identify incomparable fragments of game logic: the formula φ from above is not equivalent to any ATL* formula, and the ATL formula $\langle\!\langle A_1\rangle\!\rangle\Diamond\langle\!\langle A_2\rangle\!\rangle\Diamond\varphi'$ is not equivalent to any formula with a single strategy quantifier. In [KV96], it is shown that the module-checking complexity is EXPTIME-complete for CTL and 2EXPTIME-complete for CTL*, and the structure complexity of both problems is PTIME-complete. Applying the method there in a bottom-up fashion can be used to solve the model-checking problem for game logic, resulting in a joint complexity of 2EXPTIME and a structure complexity of PTIME. Thus, game logic is no more expensive than ATL*. We feel, however, that unlike state and path formulas, tree formulas are not natural specifications of reactive systems.

5.4 Alternating-time fixpoint logic

Temporal properties using the until operator can be defined as fixed points of next-time properties. For closed systems, this gives the μ-calculus as a generalization of temporal logics [Koz83]. In a similar fashion, one can generalize alternating-time temporal logics to obtain an *alternating-time μ-calculus*, whose primitives are the parameterized next constructs $\langle\!\langle A\rangle\!\rangle\bigcirc$ and $[\![A]\!]\bigcirc$, least and greatest fixed-point operators, and positive boolean connectives. Then, ATL* is a proper fragment of the alternating-time μ-calculus, and every ATL formula is equivalent to a fixed-point formula without alternation of least and greatest fixed points. In practice, however, designers prefer temporal operators over fixed points [BBG$^+$94]. So, just as CTL and CTL* capture useful and friendly subsets of the μ-calculus for the specification of closed system, ATL and ATL* capture useful and friendly subsets of the alternating-time μ-calculus for the specification of open systems. It is worth noting that CTL with parameterized next constructs is not of sufficient use, because it cannot specify the unbounded alternating reachability property $\langle\!\langle A\rangle\!\rangle\Diamond\varphi$ [BVW94]. Hence it is essential that in ATL we can parameterize path quantifiers, not just next-time operators.

5.5 Alternating-time transition systems

The parameterized next construct $\langle\!\langle A\rangle\!\rangle\bigcirc$ is different from similar operators commonly written as $\exists\bigcirc_A$ and interpreted as "for some agent $a \in A$, there exists an a-successor." Rather, the ATL formula $\langle\!\langle A\rangle\!\rangle\bigcirc\varphi$ is equivalent to $\exists\bigcirc_A\varphi \wedge \forall\bigcirc_{\Sigma\setminus A}\varphi$. For the abstract specification of open systems, it is essential that the parameterized next has a game-like interpretation, not a standard modal interpretation. This is because our definitions of synchronous and asynchronous structures are only approximations of "real" concurrency models for open systems (synchronous, e.g. [AH96], or asynchronous, e.g. [Lyn96]). While in our structures, each transition corresponds to a step of a single agent, in these models, a transition may be the result of simultaneous independent decisions by several agents. The more general situation gives rise to games with complex individual moves that can be captured abstractly by *alternating transition systems* (see the full paper). The game-like interpretation of the parameterized next makes ATL robust with respect to such changes in the definition of individual moves. In particular, all of our results carry over to alternating transition systems, and therefore apply, for example, to Reactive Modules [AH96].

Acknowledgments. We thank Amir Pnueli, Moshe Vardi, and Mihalis Yannakakis for helpful discussions.

References

[AH96] R. Alur, T.A. Henzinger. Reactive modules. *Proc. IEEE LICS*, pp. 207–218, 1996.

[AL93] M. Abadi, L. Lamport. Composing specifications. *ACM Trans. Programming Languages and Systems*, 15:73–132, 1993.

[ALW89] M. Abadi, L. Lamport, P. Wolper. Realizable and unrealizable concurrent program specifications. *Proc. ICALP*, Springer LNCS 372, pp. 1–17, 1989.

[BBG+94] I. Beer, S. Ben-David, D. Geist, R. Gewirtzman, M. Yoeli. Methodology and system for practical formal verification of reactive hardware. *Proc. CAV*, Springer LNCS 818, pp. 182–193, 1994.

[BCM+90] J.R. Burch, E.M. Clarke, K.L. McMillan, D.L. Dill, L.J. Hwang. Symbolic model checking: 10^{20} states and beyond. *Proc. IEEE LICS*, pp. 428–439, 1990.

[BVW94] O. Bernholtz, M.Y. Vardi, P. Wolper. An automata-theoretic approach to branching-time model checking. *Proc. CAV*, Springer LNCS 818, pp. 142–155, 1994.

[CE81] E.M. Clarke, E.A. Emerson. Design and synthesis of synchronization skeletons using branching-time temporal logic. *Proc. Workshop on Logic of Programs*, Springer LNCS 131, pp. 52–71, 1981.

[CES86] E.M. Clarke, E.A. Emerson, A.P. Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Trans. Programming Languages and Systems*, 8:244–263, 1986.

[CKS81] A.K. Chandra, D.C. Kozen, L.J. Stockmeyer. Alternation. *J. ACM*, 28:114–133, 1981.

[Dil89] D.L. Dill. *Trace Theory for Automatic Hierarchical Verification of Speed-independent Circuits.* MIT Press, 1989.

[EH86] E.A. Emerson, J.Y. Halpern. Sometimes and not never revisited: on branching versus linear time. *J. ACM*, 33:151–178, 1986.

[EJ88] E.A. Emerson, C. Jutla. The complexity of tree automata and logics of programs. *Proc. IEEE FOCS*, pp. 368–377, 1988.

[ES84] E.A. Emerson, A.P. Sistla. Deciding branching-time logic. *Proc. ACM STOC*, pp. 14–24, 1984.

[GSSL94] R. Gawlick, R. Segala, J. Sogaard-Andersen, N.A. Lynch. Liveness in timed and untimed systems. *Proc. ICALP*, Springer LNCS 820, pp. 166–177, 1994.

[Hoa85] C.A.R. Hoare. *Communicating Sequential Processes.* Prentice-Hall, 1985.

[Hol97] G.J. Holzmann. The model checker SPIN. *IEEE Trans. Software Engineering*, 23:279–295, 1997.

[Imm81] N. Immerman. Number of quantifiers is better than number of tape cells. *J. Computer and System Sciences*, 22:384–406, 1981.

[Koz83] D.C. Kozen. Results on the propositional μ-calculus. *Theoretical Computer Science*, 27:333–354, 1983.

[KV96] O. Kupferman, M.Y. Vardi. Module checking. *Proc. CAV*, Springer LNCS 1102, pp. 75–86, 1996.

[KV97] O. Kupferman, M.Y. Vardi. Module checking revisited. *Proc. CAV*, Springer LNCS 1254, pp. 36–47, 1997.

[LP85] O. Lichtenstein and A. Pnueli. Checking that finite-state concurrent programs satisfy their linear specification. *Proc. ACM POPL*, pp. 97–107, 1985.

[Lyn96] N.A. Lynch. *Distributed Algorithms.* Morgan-Kaufmann, 1996.

[McM93] K.L. McMillan. *Symbolic Model Checking.* Kluwer, 1993.

[Pnu77] A. Pnueli. The temporal logic of programs. *Proc. IEEE FOCS*, pp. 46–57, 1977.

[PR79] G.L. Peterson, J.H. Reif. Multiple-person alternation. *Proc. IEEE FOCS*, pp. 348–363, 1979.

[PR89a] A. Pnueli, R. Rosner. On the synthesis of a reactive module. In *Proc. ACM POPL*, pp. 179–190, 1989.

[PR89b] A. Pnueli, R. Rosner. On the synthesis of an asynchronous reactive module. *Proc. ICALP*, Springer LNCS 372, pp. 652–671, 1989.

[QS81] J.P. Queille, J. Sifakis. Specification and verification of concurrent systems in Cesar. *Proc. Symp. on Programming*, Springer LNCS 137, pp. 337–351, 1981.

[RW89] P.J.G. Ramadge, W.M. Wonham. The control of discrete event systems. *IEEE Trans. Control Theory*, 77:81–98, 1989.

[Ros92] R. Rosner. *Modular Synthesis of Reactive Systems.* PhD thesis, Weizmann Institute of Science, Rehovot, Israel, 1992.

[Tho95] W. Thomas. On the synthesis of strategies in infinite games. *Proc. TACAS*, Springer LNCS 900, pages 1–13, 1995.

[VW86] M.Y. Vardi, P. Wolper. Automata-theoretic techniques for modal logics of programs. *J. Computer and System Sciences*, 32:182–221, 1986.

[Yan97] M. Yannakakis. Personal communication, 1997.