Alternating-time Temporal Logic*

Rajeev Alur[1] and Thomas A. Henzinger[2] and Orna Kupferman[2]

[1] Department of Computer and Information Science,
University of Pennsylvania, Philadelphia, PA 19104,
and Computing Science Research Center, Bell Laboratories, Murray Hill, NJ 07974.
Email: alur@cis.upenn.edu. URL: www.cis.upenn.edu/~alur.
[2] Department of Electrical Engineering and Computer Sciences,
University of California, Berkeley, CA 94720.
Email: {tah,orna}@eecs.berkeley.edu. URL: www.eecs.berkeley.edu/~{tah,orna}.

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* over *alternating transition systems*. While in ordinary transition systems, each transition corresponds to a possible step of the system, in alternating transition systems, each transition corresponds to a possible move in the game between the system and the environment. Fair alternating transition systems can capture both synchronous and asynchronous compositions of open systems. 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.

---

* An abbreviated version of this paper appeared in the *Proceedings of the 38th IEEE Symposium on Foundations of Computer Science* (FOCS '97), pp. 100-109, October 1997. This work 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.

Table of Contents

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*, on the other hand, 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 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.

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 of the environment. It is sensible, and necessary for compositionality, to require an affirmative answer to the receptiveness problem.

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.

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).

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. We model open systems by *alternating transition systems*. The transitions of an alternating transition system correspond to possible moves in a game between the agents. In each move of the game, every agent chooses a set of successor states. The game then proceeds to the (single) state in the intersection of the sets chosen by the agents. Special cases of the game are *turn-based synchronous* (in each state, only one agent restricts the set of successor states, and that agent is determined by the state), *lock-step synchronous* (the state is partitioned according to the agents, and in each step, every agent updates its component of the state), and *turn-based asynchronous* (in each state, only one agent restricts the set of successor states, and that agent is chosen by a fair scheduler). These subclasses of alternating transition systems capture various notions of synchronous and asynchronous interaction between open systems.

For a set $A \subseteq \Sigma$ of agents, a set Λ of computations, and a state q of the system, consider the following game between a protagonist and an antagonist. The game starts at the state q. At each step, to determine the next state, the protagonist chooses the choices controlled by the agents in the set A, while the antagonist chooses the remaining choices. If the resulting infinite computation belongs to the set Λ, then the protagonist wins. If the protagonist has a winning strategy, we say that the alternating-time formula $\langle\langle A \rangle\rangle \Lambda$ is satisfied in the state q. 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 $\Sigma \setminus A$ proceed. 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*, which represents the system. 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 Λ of computations, we obtain two alternating-time temporal logics: in the logic ATL*, the set Λ is specified by a formula of LTL; in the more restricted logic ATL, the set Λ 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
$$\forall\Box\,(\exists\Diamond\, read \;\wedge\; \exists\Diamond\, write).$$
The ATL formula
$$\forall\Box\,(\langle\langle a\rangle\rangle\Diamond\, read \;\wedge\; \langle\langle a\rangle\rangle\Diamond\, write)$$
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 $\forall\Box\,\langle\langle sys\rangle\rangle\,\psi$.
4. Checking the *realizability (program synthesis)* of an LTL formula ψ corresponds to model checking of the ATL* formula $\langle\langle sys\rangle\rangle\psi$ 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 $\langle\langle control \rangle\rangle \Box \varphi$. Controller synthesis, then, corresponds to model checking of this formula. More generally, for an LTL formula ψ, the ATL* requirement $\langle\langle control \rangle\rangle \psi$ 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 [VW86b]. 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.

The remaining paper is organized as follows. Section 2 defines the model of alternating transition systems, and Section 3 defines the alternating-time temporal logics ATL and ATL*. Section 4 presents symbolic model-checking procedures, and Section 5 establishes complexity bounds on model checking for alternating-time temporal logics. In Section 6, we consider more general ways of introducing game quantifiers in temporal logics. Specifically, we define an alternating-time μ-calculus and a game logic, and study their relationship to ATL and ATL*. Finally, Section 7 considers models in which agents have only partial information about (global) states. We show that for this case, of alternating transition systems with incomplete information, the model-checking problem is generally undecidable, and we describe a special case that is decidable in exponential time.

2 Alternating Transition Systems

We model open systems by alternating transition systems. While in ordinary transition systems, each transition corresponds to a possible step of the system, in alternating transition systems, each transition corresponds to a possible step in a game between the agents that constitute the system.

2.1 Definition of ATS

An *alternating transition system* (ATS, for short) is a 5-tuple $S = \langle \Pi, \Sigma, Q, \pi, \delta \rangle$ with the following components:

- Π is a set of propositions.
- Σ is a set of agents.
- Q is a set of states.
- $\pi : Q \rightarrow 2^{\Pi}$ maps each state to the set of propositions that are true in the state.
- $\delta : Q \times \Sigma \rightarrow 2^{2^Q}$ is a transition function that maps a state and an agent to a nonempty set of choices, where each choice is a set of possible next states. Whenever the system is in state q, each agent a chooses a set $Q_a \in \delta(q, a)$. In this way, an agent a ensures that the next state of the system will be in its choice Q_a. However, which state in Q_a will be next depends on the choices made by the other agents, because the successor of q must lie in the intersection $\bigcap_{a\in\Sigma} Q_a$ of the choices made by all agents. The transition function is non-blocking and the agents together choose a unique next state state. Thus, we require that this intersection always contains a unique state: assuming $\Sigma = \{a_1, \ldots, a_n\}$, then for every state $q \in Q$ and every set $Q_1, \ldots, Q_n$ of choices $Q_i \in \delta(q, a_i)$, the intersection $Q_1 \cap \ldots \cap Q_n$ is a singleton.

The number of transitions of S is defined to be $\sum_{q\in Q, a\in\Sigma} |\delta(q, a)|$. For two states q and q' and an agent a, we say that q' is an *a-successor* of q if there exists a set $Q' \in \delta(q, a)$ such that $q' \in Q'$. We denote by $succ(q, a)$ the set of a-successors of q. For two states q and q', we say that q' is a *successor* of q if for all agents $a \in \Sigma$, we have $q' \in succ(q, a)$. Thus, q' is a successor of q iff whenever the system S is in state q, the agents in Σ can cooperate so that q' will be the next state. A *computation* of S is an infinite sequence $\lambda = q_0, q_1, q_2, \ldots$ of states such that for all positions $i \geq 0$, the state q_{i+1} is a successor of the state q_i. We refer to a computation starting at state q as a *q-computation*. For a computation λ and a position $i \geq 0$, we use $\lambda[i]$, $\lambda[0, i]$, and $\lambda[i, \infty]$ to denote the i-th state in λ, the finite prefix $q_0, q_1, \ldots, q_i$ of λ, and the infinite suffix $q_i, q_{i+1}, \ldots$ of λ, respectively.

Example 1. Consider a system with two processes a and b. The process a assigns values to the boolean variable x. When $x = \mathit{false}$, then a can leave the value of x unchanged or change it to true. When $x = \mathit{true}$, then a leaves the value of x unchanged. In a similar way, the process b assigns values to the boolean variable y. When $y = \mathit{false}$, then b can leave the value of y unchanged or change it to true. When $y = \mathit{true}$, then b leaves the value of y unchanged. We model the composition of the two processes by the following ATS $S_{xy} = \langle \Pi, \Sigma, Q, \pi, \delta \rangle$:

- $\Pi = \{x, y\}$.
- $\Sigma = \{a, b\}$.
- $Q = \{q, q_y, q_x, q_{xy}\}$. The state q corresponds to $x = y = \mathit{false}$, the state q_x corresponds to $x = \mathit{true}$ and $y = \mathit{false}$, and similarly for q_y and q_{xy}.

– The labeling function $\pi : Q \rightarrow 2^{\Pi}$ is therefore as follows:
  - $\pi(q) = \emptyset$.
  - $\pi(q_x) = \{x\}$.
  - $\pi(q_y) = \{y\}$.
  - $\pi(q_{xy}) = \{x, y\}$.

– The transition function $\delta : Q \times \Sigma \rightarrow 2^{2^Q}$ is as follows:
  - $\delta(q, a) = \{\{q, q_y\}, \{q_x, q_{xy}\}\}$.
  - $\delta(q, b) = \{\{q, q_x\}, \{q_y, q_{xy}\}\}$.
  - $\delta(q_x, a) = \{\{q_x, q_{xy}\}\}$.
  - $\delta(q_x, b) = \{\{q, q_x\}, \{q_y, q_{xy}\}\}$.
  - $\delta(q_y, a) = \{\{q, q_y\}, \{q_x, q_{xy}\}\}$.
  - $\delta(q_y, b) = \{\{q_y, q_{xy}\}\}$.
  - $\delta(q_{xy}, a) = \{\{q_x, q_{xy}\}\}$.
  - $\delta(q_{xy}, b) = \{\{q_y, q_{xy}\}\}$.

  Consider, for example, the transition $\delta(q, a)$. As the process a controls only the value of x, and can change its value from *false* to *true*, the agent a can determine whether the next state of the system will be some q' with $x \in \pi(q')$ or some q' with $x \notin \pi(q')$. It cannot, however, determine the value of y. Therefore, $\delta(q, a) = \{\{q, q_y\}, \{q_x, q_{xy}\}\}$, letting a choose between $\{q, q_y\}$ and $\{q_x, q_{xy}\}$, yet leaving the choice between q and q_y, in the first case, and between q_x and q_{xy}, in the second case, to process b.

Consider the state q_x. While the state q_y is a b-successor of q_x, the state q_y is not an a-successor of q_x. Therefore, the state q_y is not a successor of q_x: when the system is in state q_x, the processes a and b cannot cooperate so that the system will move to q_y. On the other hand, the agents can cooperate so that the system will stay in state q_x or move to q_{xy}. By similar considerations, it follows that the infinite sequences $q, q, q_x, q_x, q_x, q_{xy}^{\omega}$ and $q, q_y, q_y, q_{xy}^{\omega}$ and q, q_{xy}^{ω} are three possible q-computations of the ATS S_{xy}.

Now suppose that process b can change y from *false* to *true* only when x is already *true*. The resulting ATS $S'_{xy} = \langle \Pi, \Sigma, Q, \pi, \delta' \rangle$ differs from S_{xy} only in the transition function: $\delta'(q, b) = \{\{q, q_x\}\}$, and in all other cases δ' agrees with δ. While $q, q, q_x, q_x, q_x, q_{xy}^{\omega}$ is a possible q-computation of the ATS S'_{xy}, the sequences $q, q_y, q_y, q_{xy}^{\omega}$ and q, q_{xy}^{ω} are not.

Third, suppose that process b can change y from *false* to *true* either when x is already *true*, or when simultaneously x is set to *true*. The transition function of the resulting ATS $S''_{xy} = \langle \Pi, \Sigma, Q, \pi, \delta'' \rangle$ differs from δ only in $\delta''(q, b) = \{\{q, q_x\}, \{q, q_{xy}\}\}$. In state q, if process b decides to leave y unchanged, it chooses the first option $\{q, q_x\}$. If, on the other hand, process b decides to change the value of y to *true* provided that x is simultaneously changed to *true* by process a, then b chooses the second option $\{q, q_{xy}\}\}$. Then $q, q, q_x, q_x, q_x, q_{xy}^{\omega}$ and q, q_{xy}^{ω} are possible q-computations of the ATS S''_{xy}, while $q, q_y, q_y, q_{xy}^{\omega}$ is not.

Finally, suppose we consider process b on its own. In this case, we have two agents, b and *env*, where *env* represents the environment, which may, in any state, change the value of x arbitrarily. The resulting ATS $S'''_{xy} = \langle \Pi, \Sigma''', Q, \pi, \delta''' \rangle$ has the set $\Sigma''' = \{b, env\}$ of agents and the following transition relation:

- $\delta'''(q, b) = \delta'''(q_x, b) = \{\{q, q_x\}, \{q_y, q_{xy}\}\}$.
- $\delta'''(q_y, b) = \delta'''(q_{xy}, b) = \{\{q_y, q_{xy}\}\}$.
- $\delta'''(q, \mathrm{env}) = \delta'''(q_x, \mathrm{env}) = \delta'''(q_y, \mathrm{env}) = \delta'''(q_{xy}, \mathrm{env}) = \{\{q, q_y\}, \{q, q_{xy}\}, \{q_x, q_{xy}\}, \{q_x, q_y\}\}$.

□

An ordinary *labeled transition system*, or Kripke structure, is the special case of an ATS where the set $\Sigma = \{sys\}$ of agents is a singleton set. In this special case, the sole agent sys can always determine the successor state: for all states $q \in Q$, the transition $\delta(q, sys)$ must contain a nonempty set of choices, each of which is a singleton set.

2.2 Synchronous ATS

In this section we present two special cases of alternating transition systems. Both cases correspond to a synchronous composition of agents.

Turn-based synchronous ATS In a turn-based synchronous ATS, at every state only a single agent is scheduled to proceed and that agent determines the next state. It depends on the state which agent is scheduled. Accordingly, an ATS is *turn-based synchronous* if for every state $q \in Q$, there exists an agent $a_q \in \Sigma$ such that $\delta(q, a_q)$ is a set of singleton sets and for all agents $b \in \Sigma \backslash \{a_q\}$, we have $\delta(q, b) = \{Q\}$. Thus, in every state q only the agent a_q constrains the choice of the successor state. Equivalently, a turn-based synchronous ATS can be viewed as a 6-tuple $S = \langle \Pi, \Sigma, Q, \pi, \sigma, R \rangle$, where $\sigma : Q \to \Sigma$ maps each state q to the agent a_q that is scheduled to proceed at q, and $R \subseteq Q \times Q$ is a total transition relation. Then q' is a successor of q iff $R(q, q')$.

Example 2. Consider the ATS $S_1 = \langle \Pi, \Sigma, Q, \pi, \delta \rangle$ shown in Figure 1:

- $\Pi = \{out\_of\_gate, in\_gate, request, grant\}$.
- $\Sigma = \{train, ctr\}$.
- $Q = \{q_0, q_1, q_2, q_3\}$.
-
  - $\pi(q_0) = \{out\_of\_gate\}$.
  - $\pi(q_1) = \{out\_of\_gate, request\}$.
  - $\pi(q_2) = \{out\_of\_gate, grant\}$.
  - $\pi(q_3) = \{in\_gate\}$.
-
  - $\delta(q_0, train) = \{\{q_0\}, \{q_1\}\}$.
  - $\delta(q_1, ctr) = \{\{q_0\}, \{q_1\}, \{q_2\}\}$.
  - $\delta(q_2, train) = \{\{q_0\}, \{q_3\}\}$.
  - $\delta(q_3, ctr) = \{\{q_0\}, \{q_3\}\}$.
  - $\delta(q_0, ctr) = \delta(q_1, train) = \delta(q_2, ctr) = \delta(q_3, train) = \{Q\}$.

Since S_1 is a turn-based synchronous ATS, its transition function δ induces an assignment of agents to states: $\sigma(q_0) = \sigma(q_2) = \mathit{train}$ and $\sigma(q_1) = \sigma(q_3) = \mathit{ctr}$. The ATS 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 system has two agents: the train and the controller. Two states of the system, 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. □

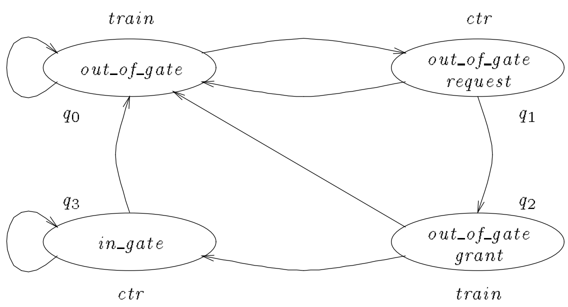

Fig. 1. A train controller as a turn-based synchronous ATS

Lock-step synchronous ATS In a lock-step synchronous ATS, the state space is the product of local state spaces, one for each agent. Then, in every state, all agents proceed simultaneously. Each agent determines its next local state, possibly dependent on the current local states of the other agents but independent of the choices taken by the other agents. Accordingly, an ATS is *lock-step synchronous* if the following two conditions are satisfied:

1. The state space has the form $Q = \Pi_{a \in \Sigma} Q_a$. Given a (global) state $q \in Q$ and an agent $a \in \Sigma$, we write $q[a]$ for the component of q local to a. Then, assuming $\Sigma = \{a_1, \ldots, a_n\}$, every state has the form $q = \langle q[a_1], \ldots, q[a_n] \rangle$.
2. For every state $q \in Q$ and every agent $a \in \sigma$, there exists a set $\{q_1, \ldots, q_k\} \subseteq Q_a$ of states local to a such that $\delta(q, a) = \{Q_1, \ldots, Q_k\}$ for $Q_i = \{q \in Q \mid q[a] = q_i\}$. Thus, while the agent a can determine its next local state, it cannot determine the next local states of the other agents.

Equivalently, the transition function δ can be replaced by a set of local transition functions $\delta_a : Q \rightarrow 2^{Q_a}$, one for each agent $a \in \Sigma$ and all of them total. Then q' is a successor of q iff for all agents $a \in \Sigma$, we have $q'[a] \in \delta_a(q)$.

Example 3. The ATS S_{xy} from Example 1 is lock-step synchronous. To see this, note that its state space $Q = \{q, q_x, q_y, q_{xy}\}$ can be viewed as the product of $Q_a = \{u, u_x\}$ and $Q_b = \{v, v_y\}$ with $q = \langle u, v\rangle$, $q_x = \langle u_x, v\rangle$, $q_y = \langle u, v_y\rangle$, and $q_{xy} = \langle u_x, v_y\rangle$. The local transition functions are as follows:

- $\delta_a(q) = \delta_a(q_y) = \{u, u_x\}$.
- $\delta_a(q_x) = \delta_a(q_{xy}) = \{u_x\}$.
- $\delta_b(q) = \delta_b(q_x) = \{v, v_y\}$.
- $\delta_b(q_y) = \delta_b(q_{xy}) = \{v_y\}$.

Also the ATS S'_{xy} from Example 1 is lock-step synchronous, but the ATS S''_{xy} and S'''_{xy} are not. For S''_{xy}, this is because the ability of process b to change the value of y depends on what process a does at the same step. □

2.3 Fair ATS

When systems are modeled as ordinary transition systems, to establish liveness properties, it is often necessary to rule out certain infinite computations that ignore enabled choices forever. For instance, in an asynchronous system consisting of many processes, we may like to restrict attention to the computations in which all the processes take infinitely many steps. Such assumptions can be incorporated in the model by adding fairness conditions. Motivated by similar concerns, we define fairness conditions for ATS.

A *fairness condition* for the ATS $S = \langle \Pi, \Sigma, Q, \pi, \delta\rangle$ is a set of fairness constraints for S, each defining a subset of the transition function. More precisely, a *fairness constraint* for S is a function $\gamma : Q \times \Sigma \rightarrow 2^{2^Q}$ such that $\gamma(q, a) \subseteq \delta(q, a)$ for all states $q \in Q$ and all agents $a \in \Sigma$. As with ordinary transition systems, a fairness condition partitions the computations of an ATS into computations that are fair and computations that are not fair. We elaborate on two interpretations for fairness constraints. Consider a computation $\lambda = q_0, q_1, q_2, \ldots$ of the ATS S, a fairness constraint $\gamma : Q \times \Sigma \rightarrow 2^{2^Q}$ for S, and an agent $a \in \Sigma$. We say that γ is *a-enabled* at position $i \geq 0$ of λ if $\gamma(q_i, a) \neq \emptyset$. We say that γ is *a-taken* at position i of λ if there exists a set $Q' \in \gamma(q_i, a)$ such that $q_{i+1} \in Q'$. The two interpretations for fairness constraints are defined with respect to a set $A \subseteq \Sigma$ of agents as follows:

- The computation λ is *weakly* $\langle\gamma, A\rangle$*-fair* if for each agent $a \in A$, either there are infinitely many positions of λ at which γ is not a-enabled, or there are infinitely many positions of λ at which γ is a-taken.
- The computation λ is *strongly* $\langle\gamma, A\rangle$*-fair* if for each agent $a \in A$, either there are only finitely many positions of λ at which γ is a-enabled, or there are infinitely many positions of λ at which γ is a-taken. With these standard definitions, strong fairness implies weak fairness.

Now, given a fairness condition Γ for the ATS S and a set $A \subseteq \Sigma$ of agents, the computation λ is *weakly/strongly* $\langle\Gamma, A\rangle$*-fair* if λ is weakly/strongly $\langle\gamma, A\rangle$-fair for all fairness constraints $\gamma \in \Gamma$. Note that for every fairness condition Γ and every set A of agents, each prefix of a computation of S can be extended to a computation that is strongly $\langle \Gamma, A \rangle$-fair. Note also that a computation λ is weakly/strongly $\langle \Gamma, A_1 \cup A_2 \rangle$-fair, for $A_1, A_2 \subseteq \Sigma$, iff λ is weakly/strongly both $\langle \Gamma, A_1 \rangle$-fair and $\langle \Gamma, A_2 \rangle$-fair,

Example 4. Consider the ATS S_{xy} from Example 1 and the fairness condition $\Gamma_y = \{\gamma\}$ with the fairness constraint $\gamma(q, b) = \gamma(q_x, b) = \{\{q_y, q_{xy}\}\}$ (we specify only the nonempty values of a fairness constraint). All computations of the ATS S_{xy} are strongly $\langle \Gamma_y, \{a\} \rangle$-fair. However, only computations in which the value of the variable y is eventually *true* are weakly or strongly $\langle \Gamma_y, \{b\} \rangle$-fair or, for that matter, $\langle \Gamma_y, \{a, b\} \rangle$-fair. This is because, as long as the value of y is *false*, the ATS S_{xy} is either in state q or in state q_x. Therefore, as long as the value of y is *false*, the fairness constraint γ is b-enabled. Thus, in a fair computation, γ will eventually be b-taken, changing the value of y to *true*. □

As with ordinary transition systems, fairness enables us to exclude some computations of an ATS. In particular, fairness enables us to model asynchronous systems.

Turn-based asynchronous ATS In a turn-based asynchronous ATS, at every state only a single agent determines the next state. However, unlike in a turn-based synchronous ATS, the state does not determine which agent is scheduled to proceed. Rather, a turn-based asynchronous ATS has a designated agent *sch*, which represents a *scheduler*. The scheduler *sch* proceeds at all states and determines one other agent to proceed with it. That other agent determines the next state. Fairness constraints are used to guarantee that the scheduling policy is fair. Accordingly, an ATS is *turn-based asynchronous* if there exists an agent $sch \in \Sigma$ and for every state $q \in Q$ and every agent $a \in \Sigma \backslash \{sch\}$, there exists a local transition function $\delta_a : Q \rightarrow 2^Q$ such that the following four conditions are satisfied:

1. For all states $q \in Q$ and all agents $a, b \in \Sigma \backslash \{sch\}$, if $a \neq b$ then $\delta_a(q) \cap \delta_b(q) = \emptyset$. We say that agent a is *enabled* in state q if $\delta_a(q) \neq \emptyset$.
2. For all states $q \in Q$, we have $\delta(q, sch) = \{\delta_a(q) \mid \text{the agent } a \in \Sigma \setminus \{sch\}$ is enabled in $q\}$. That is, if the scheduler *sch* chooses the option $\delta_a(q)$, the agent a is scheduled to proceed in state q.
3. For all states $q \in Q$ and all agents $a \in \Sigma \setminus \{sch\}$ that are not enabled in q, we have $\delta(q, a) = \{Q\}$. That is, if the agent a is not enabled, it does not influence the successor state.
4. For all states $q \in Q$ and all agents $a \in \Sigma \setminus \{sch\}$ that are enabled in q, assuming $\delta_a(q) = \{q_1, \ldots, q_k\}$, we have $\delta(q, a) = \{(Q \setminus \delta_a(q)) \cup \{q_1\}, \ldots, (Q \setminus \delta_a(q)) \cup \{q_k\}\}$. That, if the agent a is enabled in state q, it chooses a successor state in $\delta_a(q)$ provided it is scheduled to proceed. If, however, a is not scheduled to proceed in q, then it does not influence the successor state, which must lie in $Q \setminus \delta_a(q)$ because of the first condition.

Equivalently, a turn-based asynchronous ATS can be viewed as a 6-tuple $S = \langle \Pi, \Sigma \setminus \{sch\}, Q, \pi, R, \sigma \rangle$, where $R \subseteq Q \times Q$ is a total transition relation, and $\sigma : R \rightarrow \Sigma \setminus \{s\}$ maps each transition to an agent. Then $\delta_a(q) = \{q' \in Q \mid R(q, q')$ and $\sigma(q, q') = a\}$. Note that, while in a turn-based synchronous ATS we label states with agents, in a turn-based asynchronous ATS we label transitions with agents.

In order to ensure fairness of the scheduler, we impose a fairness condition $\Gamma = \{\gamma_a \mid a \in \Sigma \setminus \{sch\}\}$ with a turn-based asynchronous ATS. The fairness condition Γ contains a fairness constraint γ_a for each agent a different from sch, which ensures that the scheduler does not neglect a forever. For all states $q \in Q$, we have $\gamma_a(q, sch) = \{\delta_a(q)\}$ and for all agents $b \in \Sigma \setminus \{sch\}$ (possibly $b = a$), we have $\gamma_a(q, b) = \emptyset$. Then, a computation λ is weakly $\langle \gamma_a, \{sch\} \rangle$-fair iff either there are infinitely many positions of λ at which the agent a is not enabled, or there are infinitely many positions of λ at which a is scheduled to proceed. Similarly, λ is strongly $\langle \gamma_a, \{sch\} \rangle$-fair iff either there are only finitely many positions of λ at which the agent a is enabled, or there are infinitely many positions of λ at which a is scheduled to proceed.

Example 5. As an example of a turn-based asynchronous ATS consider the modeling of the sender process of the alternating-bit protocol shown in Figure 2. There are two agents, the sender and the environment. In the initial state q_0, only the sender is enabled, and it chooses either to stay in q_0 or to move to state q_1. The transition from q_0 to q_1 corresponds to sending a message tagged with the bit 0. In state q_1 the sender is waiting to receive an acknowledgment. Both agents are enabled, and the scheduler chooses one of them. The sender, if scheduled to proceed in state q_1, continues to wait. Each environment transitions correspond to the reception of an acknowledgment by the sender. If the acknowledgment bit is 0, the sender proceeds to toggle its bit by moving to state q_2, and if the acknowledgment bit is 1, the sender attempts to resend the message by moving back to state q_0. This phenomenon is modeled by letting the environment, when scheduled in state q_1, choose between q_0 and q_2. State q_1 is similar to state q_0, and q_3 is similar to q_1.

Formally, $Q = \{q_0, q_1, q_2, q_3\}$ and $\Sigma = \{sender, env, sch\}$. The set Π contains four propositions: *send0* is true in state q_0, *wait0* is true in state q_1, *send1* is true in state q_2, and *wait1* is true in state q_3. The local transition functions are as follows:

- $\delta_{sender}(q_0) = \{q_0, q_1\}$.
- $\delta_{env}(q_0) = \emptyset$.
- $\delta_{sender}(q_1) = \{q_1\}$.
- $\delta_{env}(q_1) = \{q_0, q_2\}$.
- $\delta_{sender}(q_2) = \{q_2, q_3\}$.
- $\delta_{env}(q_2) = \emptyset$.
- $\delta_{sender}(q_3) = \{q_3\}$.
- $\delta_{env}(q_3) = \{q_0, q_2\}$.

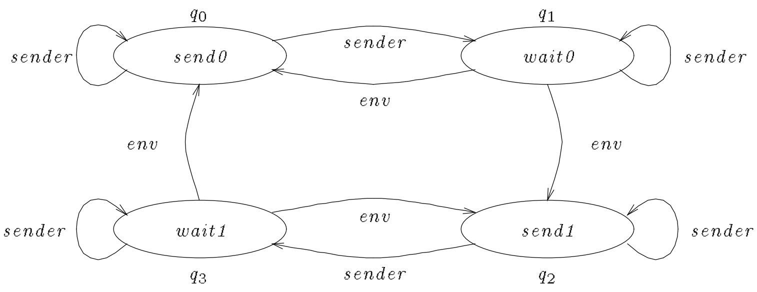

Fig. 2. A send protocol as a turn-based asynchronous ATS

These local transition functions induce the following transition function:

- $\delta(q_0, sch) = \{\{q_0, q_1\}\}$.
- $\delta(q_0, sender) = \{\{q_0, q_2, q_3\}, \{q_1, q_2, q_3\}\}$.
- $\delta(q_0, env) = \{\{q_0, q_1, q_2, q_3\}\}$.
- $\delta(q_1, sch) = \{\{q_1\}, \{q_0, q_2\}\}$.
- $\delta(q_1, sender) = \{\{q_0, q_2, q_3, q_1\}\}$.
- $\delta(q_1, env) = \{\{q_1, q_3, q_0\}, \{q_1, q_3, q_2\}\}$.
- $\delta(q_2, sch) = \{\{q_2, q_3\}\}$.
- $\delta(q_2, sender) = \{\{q_0, q_1, q_2\}, \{q_0, q_1, q_3\}\}$.
- $\delta(q_2, env) = \{\{q_0, q_1, q_2, q_3\}\}$.
- $\delta(q_3, sch) = \{\{q_3\}, \{q_0, q_2\}\}$.
- $\delta(q_3, sender) = \{\{q_0, q_1, q_2, q_3\}\}$.
- $\delta(q_3, env) = \{\{q_1, q_3, q_0\}, \{q_1, q_3, q_2\}\}$.

The weak-fairness constraint γ_{env} ensures that if the sender is waiting in state q_1 or q_2, it will eventually receive an acknowledgment:

- $\gamma_{env}(q_1, sch) = \gamma_{env}(q_3, sch) = \{\{q_0, q_2\}\}$

(we specify only the nonempty values of a fairness constraint). The assumption that the environment does not keep sending incorrect acknowledgments forever, which ensures progress of the protocol, can be modeled by a strong-fairness constraint γ':

- $\gamma'(q_1, env) = \{\{q_1, q_3, q_2\}\}$.
- $\gamma'(q_3, env) = \{\{q_1, q_3, q_0\}\}$.

$\square$

3 Alternating-time Temporal Logic

3.1 ATL Syntax

The temporal logic ATL (*Alternating-time Temporal 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) $\langle\!\langle A \rangle\!\rangle \bigcirc \varphi$, $\langle\!\langle A \rangle\!\rangle \Box \varphi$, 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"), $\Box$ ("always"), and $\mathcal{U}$ ("until") are *temporal operators*. The logic ATL is similar to the branching-time temporal logic CTL, only that path quantifiers are parameterized by sets of agents. Sometimes we write $\langle\!\langle a_1, \ldots, a_n \rangle\!\rangle$ instead of $\langle\!\langle \{a_1, \ldots, a_n\} \rangle\!\rangle$. Additional boolean connectives are defined from $\neg$ and $\vee$ in the usual manner. As in CTL, we write $\langle\!\langle A \rangle\!\rangle \Diamond \varphi$ for $\langle\!\langle A \rangle\!\rangle \mathit{true}\, \mathcal{U} \varphi$.

3.2 ATL Semantics

We interpret ATL formulas over the states of a given ATS S that has the same propositions and agents. The labeling of the states of S with propositions is used to evaluate the atomic formulas of ATL. The logical connectives $\neg$ and $\vee$ have the standard interpretation. To evaluate a formula of the form $\langle\!\langle A \rangle\!\rangle \psi$ at a state q of S, consider the following two-player game. The game proceeds in an infinite sequence of rounds, and after each round, the position of the game is a state of S. The initial position is q. Now consider the game in some position u. To update the position, first the protagonist chooses for every agent $a \in A$, a set $Q_a \in \delta(u, a)$. Then, the antagonist chooses a successor v of u such that $v \in Q_a$ for all $a \in A$, and the position of the game is updated to v. In this way, the game continues forever and produces a computation. The protagonist wins the game if the resulting computation satisfies the subformula ψ, read as a linear temporal formula whose outermost operator is $\bigcirc$, $\Box$, or $\mathcal{U}$. The ATL formula $\langle\!\langle A \rangle\!\rangle \psi$ holds at the state q if the protagonist has a winning strategy in this game.

In order to define the semantics of ATL formally, we first define the notion of strategies. Consider an ATS $S = \langle \Pi, \Sigma, Q, \pi, \delta \rangle$. A *strategy* for an agent $a \in \Sigma$ is a mapping $f_a : Q^+ \to 2^Q$ such that for all $\lambda \in Q^*$ and all $q \in Q$, we have $f_a(\lambda \cdot q) \in \delta(q, a)$. Thus, the strategy f_a maps each finite prefix $\lambda \cdot q$ of a computation to a set in $\delta(q, a)$. This set contains possible extensions of the computation as suggested to agent a by the strategy. Each strategy f_a induces a set of computations that agent a can enforce. Given a state q, a set A of agents, and a set $F_A = \{f_a \mid a \in A\}$ of strategies, one for each agent in A, we define the *outcomes* of F_A from q to be the set $\mathit{out}(q, F_A)$ of all q-computations that the agents in A can enforce when they cooperate and follow the strategies in F_A; that is, a computation $\lambda = q_0, q_1, q_2, \ldots$ is in $\mathit{out}(q, F_A)$ if $q_0 = q$ and for all positions $i \geq 0$, the state q_{i+1} is a successor of q_i satisfying $q_{i+1} \in \bigcap_{a \in A} f_a(\lambda[0, i])$.

We can now turn to a formal definition of the semantics of ATL. We write $S, q \models \varphi$ ("state q satisfies formula φ in the structure S") to indicate that the formula φ holds at state q of S. When S is clear from the context we omit it and write $q \models \varphi$. The relation $\models$ is defined, for all states q of S, inductively as follows:

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

Note that the next operator $\bigcirc$ is local: $q \models \langle\langle A \rangle\rangle \bigcirc \varphi$ iff for each agent $a \in A$ there exists a set $Q_a \in \delta(q, a)$ such that for each state $q' \in \bigcap_{a \in A} Q_a$, if q' is a successor of q, then $q' \models \varphi$.

It is often useful to express an ATL formula in a dual form. For that, we use the path quantifier $[\![A]\!]$, for a set A of agents. While the ATL formula $\langle\langle A \rangle\rangle \psi$ intuitively means that the agents in A can cooperate to make ψ true (they can "enforce" ψ), the dual formula $[\![A]\!]\psi$ means that the agents in A cannot cooperate to make ψ false (they cannot "avoid" ψ). Using the path quantifier $[\![\ ]\!]$, we can write, for a set A of agents and an ATL formula φ, the ATL formula $[\![A]\!] \bigcirc \varphi$ for the ATL formula $\neg\langle\langle A \rangle\rangle \bigcirc \neg\varphi$, $[\![A]\!] \Box \varphi$ for $\neg\langle\langle A \rangle\rangle \Diamond \neg\varphi$, and $[\![A]\!] \Diamond \varphi$ for $\neg\langle\langle A \rangle\rangle \Box \neg\varphi$ (similar abbreviations can be defined for the dual of the $\mathcal{U}$ operator). Let us make this more precise. For a state q and a set Λ of q-computations, we say that the agents in A can *enforce* the set Λ of computations if $out(q, F_A) \subseteq \Lambda$ for some set F_A of strategies for the agents in A. Dually, we say that the agents in A can *avoid* the set Λ of computations if $\Lambda \cap out(q, F_A) = \emptyset$ for some set F_A of strategies for the agents in A. If the agents in A can enforce a set Λ of computations, then the agents in $\Sigma \setminus A$ cannot avoid Λ. Therefore, $q \models \langle\langle A \rangle\rangle \psi$ implies $q \models [\![\Sigma \setminus A]\!]\psi$. The converse of this statement is not necessarily true. To see this, consider $\Sigma = \{a, b\}$, $\delta(q, a) = \{\{q_1, q_2\}, \{q_3, q_4\}\}$ and $\delta(q, b) = \{\{q_1, q_3\}, \{q_2, q_4\}\}$, assuming each state q_i satisfies the proposition p_i and no other propositions. Then $q \not\models \langle\langle a \rangle\rangle \bigcirc (p_1 \vee p_4)$ and $q \not\models [\![b]\!] \bigcirc (p_1 \vee p_4)$; that is, neither does a have a strategy to enforce $\bigcirc(p_1 \vee p_4)$ nor does b have a strategy to avoid $\bigcirc(p_1 \vee p_4)$.

Example 6. Recall the turn-based synchronous ATS S_1 from Example 2. Recall that in a turn-based synchronous ATS, every state is labeled with an agent that determines the successor state. In this simplified setting, to determine the truth of a formula with path quantifier $\langle\langle A \rangle\rangle$, we can consider the following simpler version of the ATL game. In every state u, if the agent scheduled to proceed in u belongs to A, then the protagonist updates the position to some successor of u, and otherwise, the antagonist updates the position to some successor of u. Therefore, every state of S_1 satisfies the following ATL formulas:

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

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

2. Whenever the train is out of the gate, the controller cannot force it to enter the gate.

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

3. Whenever the train is out of the gate, the train and the controller can cooperate so the train will enter the gate.

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

4. Whenever the train is out of the gate, it can eventually request a grant for entering the gate, in which case the controller decides whether the grant is given or not.

$$\langle\langle\ \rangle\rangle\Box(\mathit{out\_of\_gate} \rightarrow \langle\langle \mathit{train}\rangle\rangle\Diamond(\mathit{request} \wedge (\langle\langle \mathit{ctr}\rangle\rangle\Diamond \mathit{grant}) \wedge (\langle\langle \mathit{ctr}\rangle\rangle\Box\neg \mathit{grant}))$$

5. Whenever the train is in the gate, the controller can force it out in the next step.

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

These natural requirements cannot be stated in CTL or CTL*. Consider the first two ATL formulas. They provide more information than the CTL formula

$$\forall\Box(\mathit{out\_of\_gate} \rightarrow \exists\Box \mathit{out\_of\_gate}).$$

While the CTL formula only requires the existence of a computation in which the train is always out of the gate, the two 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 out of the gate. By contrast, since the train and the controller are the only agents in this example, the third ATL formula is equivalent to the CTL formula

$$\forall\Box\,(\mathit{out\_of\_gate} \rightarrow \exists\Diamond\, \mathit{in\_gate}).$$

$\Box$

Turn-based synchronous ATS It is worth noting that in the special case of a turn-based synchronous ATS, the agents in A can enforce a set Λ of computations iff the agents in $\Sigma \setminus A$ cannot avoid Λ. Therefore, for all states q of a turn-based synchronous ATS, $q \models \langle\langle A\rangle\rangle\psi$ iff $q \models [\![\Sigma \setminus A]\!]\psi$, or equivalently, $[\![A]\!] = \langle\langle \Sigma \setminus A\rangle\rangle$. Due to this strong duality, over turn-based synchronous ATS, we can define the temporal operator $\Box$ from $\Diamond$: $\langle\langle A\rangle\rangle\Box\varphi = [\![\Sigma \setminus A]\!]\Box\varphi = \neg[\![A]\!]\Diamond\neg\varphi = \neg\langle\langle \Sigma \setminus A\rangle\rangle\Diamond\neg\varphi$.

Single-agent ATS Recall that a labeled transition system is an ATS with the single agent *sys*. In this case, which is a special case of turn-based synchronous, there are only two path quantifiers: $\langle\langle sys\rangle\rangle = [\![\ ]\!]$ and $\langle\langle\ \rangle\rangle = [\![sys]\!]$. Then each set $out(q, \{f_{sys}\})$ of outcomes contains a single q-computation, and each set $out(q, \emptyset)$ of outcomes contains all q-computations. Accordingly, the path quantifiers $\langle\langle sys\rangle\rangle$ and $\langle\langle\ \rangle\rangle$ are equal, respectively, to the existential and universal path quantifiers $\exists$ and $\forall$ of the logic CTL. In other words, over labeled transition systems, ATL is identical to CTL. We write, over arbitrary ATS, $\exists$ for the path quantifier $\langle\langle\Sigma\rangle\rangle$, and $\forall$ for the path quantifier $[\![\Sigma]\!]$. This is because, regarding $\exists\psi$, all agents can cooperate to enforce a condition ψ iff there exists a computation that fulfills ψ, and regarding $\forall\psi$, all agents cannot cooperate to avoid ψ iff all computations fulfill ψ.

3.3 Fair-ATL

Since fairness constraints rule out certain computations, in their presence we need to refine the interpretation of formulas of the form $\langle\langle A\rangle\rangle\psi$. In particular, in the Fair-ATL game we require the antagonist to satisfy all fairness constraints. This leads us to the following definition. The logic Fair-ATL has the same syntax as ATL. The formulas of Fair-ATL are interpreted over an ATS S, a fairness condition Γ for S, and a state q of S. The satisfaction relation $S, \Gamma, q \models_F \varphi$ ("state q *fairly* satisfies formula φ in the structure S with respect to fairness condition Γ") for propositions and boolean connectives is defined as in the case of ATL. Moreover:

- $q \models_F \langle\langle A\rangle\rangle\bigcirc\varphi$ iff there exists a set F_A of strategies, one for each agent in A, such that for all $\langle\Gamma, \Sigma \setminus A\rangle$-fair computations $\lambda \in out(q, F_A)$, we have $\lambda[1] \models_F \varphi$.
- $q \models \langle\langle A\rangle\rangle\Box\varphi$ iff there exists a set F_A of strategies, one for each agent in A, such that for all $\langle\Gamma, \Sigma\setminus A\rangle$-fair computations $\lambda \in out(q, F_A)$ and all positions $i \geq 0$, we have $\lambda[1] \models \varphi$.
- $q \models_F \langle\langle A\rangle\rangle\varphi_1 \,\mathcal{U} \varphi_2$ iff there exists a set F_A of strategies, one for each agent in A, such that for all $\langle\Gamma, \Sigma \setminus A\rangle$-fair computations $\lambda \in out(q, F_A)$ there exists a position $i \geq 0$ such that $\lambda[i] \models_F \varphi_2$ and for all positions $0 \leq j < i$, we have $\lambda[j] \models_F \varphi_1$.

Note that the path quantifier $\langle\langle A\rangle\rangle$ ranges over the computations that are fair only with respect to the agents in $\Sigma\setminus A$. To see why, observe that once Γ contains a fairness constraint γ for which there exists an agent $a \in A$ such that $\gamma(q, a)$ is nontrivial for some state q (that is, $\emptyset \subset \gamma(q, a) \subset \delta(q, a)$), the agents in A can enforce computations that are not $\langle\Gamma, \Sigma\rangle$-fair. The above definition assures that the agents in A do not accomplish their tasks in such a vacuous way, by violating fairness.

Example 7. Consider the ATS S_1 from Example 2. Unless the controller cooperates with the train, there is no guarantee that the train eventually enters the gate:

$$q_0 \not\models \langle\langle train \rangle\rangle \Diamond in\_gate$$

So suppose we add a fairness condition $\Gamma_1 = \{\gamma_{ctr}\}$ for S_1, which imposes fairness on the control decisions in state q_1, namely, $\gamma_{ctr}(q_1, ctr) = \{\{q_2\}\}$ (all other values of γ_{ctr} are empty). If we interpret γ_{ctr} as a strong fairness constraint, then the train has a strategy to eventually enter the gate:

$$q_0 \models_F \langle\langle train \rangle\rangle \Diamond in\_gate$$

To see this, whenever the train is in q_0, let it move to q_1. Eventually, due to the strong fairness constraint, the controller will move to q_2. Then the train can move to q_3. On the other hand, if we interpret γ_{ctr} as a weak fairness constraint, cooperation between the train and the controller is still required to enter the gate, and the Fair-ATL formula is not satisfied in q_0. To see this, note that the train cannot avoid the weakly $\langle \gamma_{ctr}, \{train, ctr\}\rangle$-fair computation $q_0, q_1, q_0, q_1, \ldots$ □

3.4 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, an ATL* 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 ATL* state formulas.
(S3) $\langle\langle A \rangle\rangle \psi$, where $A \subseteq \Sigma$ is a set of agents and ψ is an ATL* path formula.

An ATL* path formula is one of the following:

(P1) An ATL* state formula.
(P2) $\neg\psi$ or $\psi_1 \vee \psi_2$, where ψ, ψ_1, and ψ_2 are ATL* path formulas.
(P3) $\bigcirc\psi$ or $\psi_1 \mathcal{U} \psi_2$, where ψ, ψ_1, and ψ_2 are ATL* path formulas.

The logic ATL* consists of the set of state formulas generated by the rules (S1–3). The logic ATL* is similar to the branching-time temporal logic CTL*, only that path quantification is parameterized by agents. Additional boolean connectives and temporal operators are defined from $\neg$, $\vee$, $\bigcirc$, and $\mathcal{U}$ in the usual manner; in particular, $\Diamond\psi = true\,\mathcal{U}\psi$ and $\Box\psi = \neg\Diamond\neg\psi$. As with ATL, we use the dual path quantifier $[\![A]\!]\psi = \neg\langle\langle A \rangle\rangle\neg\psi$, and the abbreviations $\exists = \langle\langle \Sigma \rangle\rangle$ and $\forall = [\![\Sigma]\!]$. The logic ATL can be viewed as 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* formulas is defined with respect to an ATS S. We write $S, \lambda \models \psi$ to indicate that the path formula ψ holds at computation λ of the structure S. The satisfaction relation $\models$ is defined, for all states q and computations λ of S, inductively as follows:

- For state formulas generated by the rules (S1–2), the definition is the same as for ATL.
- $q \models \langle\langle A\rangle\rangle\psi$ iff there exists a set F_A of strategies, one for each agent in A, such that for all computations $\lambda \in out(q, F_A)$, we have $\lambda \models \psi$.
- $\lambda \models \varphi$ for a state formula φ iff $\lambda[0] \models \varphi$.
- $\lambda \models \neg\psi$ iff $\lambda \not\models \psi$.
- $\lambda \models \psi_1 \vee \psi_2$ iff $\lambda \models \psi_1$ or $\lambda \models \psi_2$.
- $\lambda \models \bigcirc\psi$ iff $\lambda[1, \infty] \models \psi$.
- $\lambda \models \psi_1 \mathcal{U} \psi_2$ iff there exists a position $i \geq 0$ such that $\lambda[i, \infty] \models \psi_2$ and for all positions $0 \leq j < i$, we have $\lambda[j, \infty] \models \psi_1$.

For example, the ATL* formula

$$\langle\langle a\rangle\rangle((\Diamond\Box req) \vee (\Box\Diamond grant))$$

asserts that agent a has a strategy to enforce computations in which only finitely many requests are sent or infinitely many grants are given. Such a requirement cannot be expressed in CTL* or in ATL. Since weak and strong fairness constraints can be expressed within ATL* (provided appropriate propositions are available), there is no need for Fair-ATL*,

Remark. 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. When we consider finite ATS, all involved games are ω-regular. Then, the existence of a winning strategy implies the existence of a winning *finite-state* strategy [Rab70], which depends only on a finite amount of information about the history of the game. Thus, the semantics of ATL and ATL* with respect to finite ATS 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, finite-state strategies can be implemented using finite ATS. Indeed, for the finite reachability games and generalized Büchi games of ATL, it suffices to consider *memory-free* strategies [EJ88], 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].

4 Symbolic Model Checking

4.1 ATL Symbolic Model Checking

The *model-checking problem* for ATL asks, given an ATS $S = \langle \Pi, \Sigma, Q, \pi, \delta\rangle$ and an ATL formula φ, for the set $[\varphi] \subseteq Q$ of states of S that satisfy φ. ATL Model checking is similar to CTL model checking [CE81, QS81, BCM$^+$90]. We present a *symbolic* algorithm, which manipulates state sets of 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 Reg, when given a proposition $p \in \Pi$, returns the state set $[p]$.
- The function Pre, when given a set $A \subseteq \Sigma$ of agents and a set $\tau \subseteq Q$ of states, returns the set containing all states q such that whenever S is in state q, the agents in A can cooperate and force the next state to lie in τ. Formally, $Pre(A, \tau)$ contains state $q \in Q$ iff for each agent $a \in A$ there exists a set $Q_a \in \delta(q, a)$ such that for each state $q' \in \bigcap_{a \in A} Q_a$, if q' is a successor of q, then $q' \in \tau$.
- 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. In the special case that the ATS S is turn-based synchronous, the computation of the function Pre used in the symbolic model checking is particularly simple. Recall that in this case, $\sigma(q)$ denotes the agent that is scheduled to proceed in state q. Then, when given a set A of agents and a set τ of states, $Pre(A, \tau)$ returns the set containing all states q such that either $\sigma(q) \in A$ and some successor of q is in τ, or $\sigma(q) \notin A$ and all successors of q are in τ.

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

Fig. 3. ATL symbolic model checking

4.2 Fair-ATL Symbolic Model Checking

We turn our attention to the model-checking problem for Fair-ATL: given an ATS $S = \langle \Pi, \Sigma, Q, \pi, \delta \rangle$, a fairness condition Γ for S, and a Fair-ATL formula φ, compute the set $[\varphi]_F$ of states of S that fairly satisfy φ with respect to Γ. We use the weak interpretation for Γ; the case of strong fairness constraints can be handled similarly. Recall that to evaluate a formula of the form $\langle\!\langle A \rangle\!\rangle \psi$, we need to restrict attention to computations that satisfy all fairness constraints for agents not in A. To determine which fairness constraints are satisfied by a computation, we augment the state space by adding new propositions that indicate for each agent $a \in \Sigma$ and each fairness constraint γ, whether or not γ is a-enabled, and whether or not γ is a-taken. For this purpose, we define the ATS $S_F = \langle \Pi_F, \Sigma, Q_F, \pi_F, \delta_F \rangle$:

- For every agent $a \in \Sigma$ and every fairness constraint $\gamma \in \Gamma$, there is a new proposition $\langle \gamma, a, \mathit{enabled} \rangle$ and a new proposition $\langle \gamma, a, \mathit{taken} \rangle$: $\Pi_F = \Pi \cup (\Gamma \times \Sigma \times \{\mathit{enabled}, \mathit{taken}\})$.
- The states of S_F correspond to the transitions of S: $Q_F = \{\langle q, q' \rangle \mid q'$ is a successor of q in $S\}$.
- For every state $\langle q, q' \rangle \in Q_F$ and every agent $a \in \Sigma$, the transition $\delta_F(\langle q, q' \rangle, a)$ is obtained from $\delta(q', a)$ by replacing each state q'' appearing in $\delta(q', a)$ by the state $\langle q', q'' \rangle$. For example, if $\delta(q_0, a) = \{\{q_1, q_2\}, \{q_3\}\}$, then

$$\delta_F(\langle q, q_0 \rangle, a) = \{\{\langle q_0, q_1 \rangle, \langle q_0, q_2 \rangle\}, \{\langle q_0, q_3 \rangle\}\}.$$

- For every state $\langle q, q' \rangle \in Q_F$, we have

$$\begin{aligned} \pi_F(\langle q, q' \rangle) \;=\; & \pi(q) \cup \{\langle \gamma, a, \mathit{enabled} \rangle \mid \gamma(q, a) \neq \emptyset\} \cup \\ & \{\langle \gamma, a, \mathit{taken} \rangle \mid \text{ there exists } Q' \in \gamma(q, a) \text{ such that } q' \in Q'\}. \end{aligned}$$

Intuitively, a state of the form $\langle q, q' \rangle$ in S_F corresponds to the ATS S being in state q with the agents deciding that the successor of q will be q'. There is a one-to-one correspondence between computations of S and S_F, and between strategies in S and S_F. The new propositions in $\Gamma \times \Sigma \times \{\mathit{enabled}, \mathit{taken}\}$ allow us to identify the fair computations. Consequently, evaluating formulas of Fair-ATL over states of S can be reduced to evaluating, over states of S_F, ATL* formulas that encode the fairness constraints in Γ.

Proposition 1. *A state q of the ATS S fairly satisfies the Fair-ATL formula $\langle\!\langle A \rangle\!\rangle \psi$ with respect to the fairness condition Γ iff for each agent $a \in A$, there exists a set $Q_a \in \delta(q, a)$ such that for every successor q' of q with $q' \in \bigcap_{a \in A} Q_a$, the state $\langle q, q' \rangle$ of the ATS S_F satisfies the ATL* formula*

$$\langle\!\langle A \rangle\!\rangle (\psi \;\vee \bigvee_{\gamma \in \Gamma, a \in \Sigma \setminus A} \Diamond \Box (\langle \gamma, a, \mathit{enabled} \rangle \wedge \neg \langle \gamma, a, \mathit{taken} \rangle)).$$

This proposition reduces Fair-ATL model checking to a special case of ATL* model checking. Rather than presenting the model-checking algorithm for full Fair-ATL, we consider the sample formula $\langle\!\langle A \rangle\!\rangle \Diamond p$, for a proposition p. Consider the following game on the structure S_F. When a state labeled by p is visited, the protagonist wins. If the game continues forever, then the protagonist wins iff the resulting computation is not weakly $\langle \Gamma, \Sigma \setminus A \rangle$-fair. The winning condition for the antagonist can therefore be specified by the LTL formula

$$\Box(\neg p \;\wedge \bigwedge_{\gamma \in \Gamma, a \in \Sigma \setminus A} \Diamond(\neg \langle \gamma, a, enabled \rangle \vee \langle \gamma, a, taken \rangle)).$$

This is a generalized Büchi condition. The set of winning states in such a game can be computed using nested fixed points. To obtain an algorithm for this example, we note that the CTL* formula $\exists \Box (p \wedge \bigwedge_{1 \le i \le k} \Diamond p_i)$ can be computed symbolically as the greatest fixpoint

$$\nu X.(p \wedge \exists \bigcirc (p \, \mathcal{U} (p_1 \wedge p \, \mathcal{U} (p_2 \wedge \ldots \wedge p \, \mathcal{U} (p_k \wedge X))))).$$

Consequently, the algorithm of Figure 4 computes the set $\hat{\rho} \subseteq Q_F$ of winning states for the protagonist. The function Pre_F is like Pre, but operates on the structure S_F. By Proposition 1, the first projection of $\hat{\rho}$ gives the desired set $[\langle\!\langle A \rangle\!\rangle \Diamond p]_F \subseteq Q$ of states in the original structure S.

$\rho := [true]$; $\tau := [\neg p]$;
while $\rho \not\subseteq \tau$ **do**
  $\rho := \rho \cap \tau$;
  foreach $\gamma \in \Gamma$ **do**
    foreach $a \in \Sigma \setminus A$ **do**
      $\rho'' := [\rho] \cap (([true] \setminus Reg(\langle \gamma, a, enabled \rangle)) \cup Reg(\langle \gamma, a, taken \rangle))$;
      $\rho' := [false]$; $\tau := [\rho] \cap \rho''$;
      while $\tau' \not\subseteq \rho'$ **do** $\rho' := \rho' \cup \tau'$; $\tau' := Pre_F(\Sigma \setminus A, \rho') \cap [\neg p])$ **od**;
      $\rho := \tau'$;
    od
  od;
  $\tau := Pre_F(\Sigma \setminus A, \rho) \cap [\neg p]$
od;
return $\hat{\rho} := [true] \setminus \tau$

Fig. 4. Nested fixed-point computation for Fair-ATL symbolic model checking

5 Model-checking Complexity

We measure the complexity of the model-checking problem in two different ways: the *joint complexity* of model checking considers the complexity in terms of both the structure and the formula; the *structure complexity* of model checking (called "program complexity" in [VW86a]) considers the complexity in terms of the structure, 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 [LP85], the structure-complexity measure is of particular practical interest.

5.1 ATL Model-checking Complexity

Theorem 2. *The model-checking problem for ATL is PTIME-complete, and can be solved in time $O(m\ell)$ for an ATS with m transitions and an ATL formula of length ℓ. The structure complexity of the problem is also PTIME-complete, even in the special case of turn-based synchronous ATS.*

Proof: Consider an ATS S with m transitions and an ATL formula φ of length ℓ. We claim that the algorithm presented in Figure 3 can be implemented in time $O(m\ell)$. To see this, observe that the size of $Sub(\varphi)$ is bounded by ℓ, and that executing each of the case statements in the algorithm involves, at most, a calculation of a single fixed point, which can be done in time linear in m (see [Cle93]). Since reachability in AND-OR graphs is known to be PTIME-hard [Imm81], and can be specified using the fixed ATL formula $\langle\langle a\rangle\rangle\Diamond p$ interpreted over a turn-based synchronous ATS, hardness in PTIME, for both the joint and the structure complexity, is immediate. $\square$

It is interesting to compare the model-checking complexities of turn-based synchronous ATL and CTL. While the two 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, whereas turn-based synchronous ATL model checking is related to AND-OR graph reachability.

5.2 Fair-ATL Model-checking Complexity

As in Section 4.2, we consider the case of fairness constraints.

Theorem 3. *The model-checking problem for Fair-ATL is PTIME-complete, and can be solved in time $O(m^2n^2c^2\ell)$ for a fair ATS with m transitions and n agents, c weak fairness constraints, and an ATL formula of size ℓ. The structure complexity of the problem is also PTIME-complete.*

Proof: Consider an ATS S with m transitions, n agents, and c weak fairness constraints. Let φ be a Fair-ATL formula. Each state of S is labeled with each subformula of φ, starting with the innermost subformulas. Let us consider the case corresponding to a subformula of the form $\langle\langle A\rangle\rangle\Diamond\theta$ (the cases corresponding to $\Box$ and $\mathcal{U}$ are similar). As described in Section 4.2, we first construct the ATS S', and the truth of $\langle\langle A\rangle\rangle\Diamond\theta$ can be evaluated by solving a generalized Büchi game over the structure S'. The number of transitions in S' equals m. Note that the winning condition for the antagonist corresponds to visiting, for each fairness constraint γ and each agent $a \notin A$, infinitely often a state satisfying $\langle \gamma, a, \mathit{taken} \rangle \vee \neg \langle \gamma, a, \mathit{enabled} \rangle$. Thus, there are cn Büchi constraints. Since the complexity of solving Büchi games is quadratic (use the nested fixed-point computation of Figure 4), the cost of processing a temporal connective is $O(m^2 n^2 c^2)$. This concludes the upper bound. Since the model-checking problem for ATL is a special case of the model-checking problem for Fair-ATL (with $\Gamma = \emptyset$), hardness in PTIME follows from Theorem 2. $\square$

5.3 ATL* Model-checking Complexity

We have seen that the transition from CTL to ATL does not involve a substantial computational price. In this section we consider the model-checking complexity of ATL*. While there is an exponential price to pay in model-checking complexity when moving from CTL to CTL*, this price becomes even more significant (namely, doubly exponential) when we consider the alternating-time versions of both logics.

Before we discuss ATL* model checking, let us briefly recall CTL* model checking [EL85]. The computationally difficult case corresponds to evaluating a state formula of the form $\exists \psi$, for an LTL formula ψ. The solution is to construct a Büchi automaton $\mathcal{A}$ that accepts all computations that satisfy ψ. To determine whether a state q satisfies the formula $\exists \psi$, we need to check if some q-computation is accepted by the automaton $\mathcal{A}$, and this can be done by analyzing the product of $\mathcal{A}$ with the structure. The complexity of CTL* model checking reflects the cost of translating LTL formulas to ω-automata. In case of an ATL* state formula $\langle\!\langle A \rangle\!\rangle \psi$, the solution is similar, but requires the use of tree automata, because satisfaction corresponds to the existence of winning strategies. Therefore, model checking requires checking the nonemptiness of the intersection of two tree automata: one accepting trees in which all paths satisfy ψ, and the other accepting trees that correspond to possible strategies of the protagonist.

In order to solve the model-checking problem for ATL*, we first define the notion of execution trees. Consider an ATS S, a set A of agents, and a set $F_A = \{f_a \mid a \in A\}$ of strategies for the agents in A. For a state q of S, the set $\mathit{out}(q, F_A)$ of q-computations is fusion-closed, and therefore induces a tree $\mathit{exec}(q, F_A)$. Intuitively, the tree $\mathit{exec}(q, F_A)$ is obtained by unwinding S starting from q according to the successor relation, while pruning subtrees whose roots are not chosen by the strategies in F_A. Formally, the tree $\mathit{exec}(q, F_A)$ has as nodes the following elements of Q^*:

- q is a node (the root).
- For a node $\lambda \cdot q' \in Q^*$, the successor nodes (children) of $\lambda \cdot q'$ are all strings of the form $\lambda \cdot q' \cdot q''$, where q'' is a successor of q' and $q'' \in \bigcap_{a \in A} f_a(\lambda \cdot q')$.

A tree t is a $\langle q, A\rangle$-*execution tree* if there exists a set F_A of strategies, one for each agent in A, such that $t = exec(q, F_A)$.

Theorem 4. *The model-checking problem for ATL* is 2EXPTIME-complete, even in the special case of turn-based synchronous ATS. The structure complexity of the problem is PTIME-complete.*

Proof: Consider an ATS S and an ATL* formula φ. As in the algorithm for CTL* model checking, we label each state q of S by all state subformulas of φ that are satisfied in q. We do this in a bottom-up fashion, starting from the innermost state subformulas of φ. For subformulas generated by the rules (S1–2), the labeling procedure is straightforward. For subformulas φ' generated by (S3), we employ the algorithm for CTL* module checking [KV96] as follows. Let $\varphi' = \langle\langle A\rangle\rangle\psi$; since the satisfaction of all state subformulas of ψ has already been determined, we can assume that ψ is an LTL formula. We construct a Rabin tree automaton $\mathcal{A}_\psi$ that accepts precisely the trees satisfying the CTL* formula $\forall\psi$, and for each state q of S, we construct a Büchi tree automaton $\mathcal{A}_{S,q,A}$ that accepts precisely the $\langle q, A\rangle$-execution trees. The automaton $\mathcal{A}_\psi$ has $2^{2^{O(|\psi|)}}$ states and $2^{O(|\psi|)}$ Rabin pairs [ES84]. The automaton $\mathcal{A}_{S,q,A}$ has $|Q|$ states. The product of the two automata $\mathcal{A}_\psi$ and $\mathcal{A}_{S,q,A}$ is a Rabin tree automaton that accepts precisely the $\langle q, A\rangle$-execution trees satisfying $\forall\psi$. Hence, $q \models \langle\langle A\rangle\rangle\psi$ iff the product automaton is nonempty. The nonemptiness problem for a Rabin tree automaton with n states and r pairs can be solved in time $O(nr)^{3r}$ [EJ88, PR89a]. Hence, labeling a single state with φ' requires at most time $(|Q| \cdot 2^{2^{|\psi|}})^{2^{O(|\psi|)}} = |Q|^{2^{O(|\psi|)}}$. Since there are $|Q|$ states and at most $|\varphi|$ subformulas, membership in 2EXPTIME follows.

For the lower bound, we use a reduction from the realizability problem for LTL [PR89a], which is shown to be 2EXPTIME-hard in [Ros92]. In this problem, we are given an LTL formula ψ over a set Π of propositions and we determine whether there exists a turn-based synchronous ATS S with two agents, *sys* and *env*, such that

1. the transitions in S alternate between *sys* states and *env* states,
2. every *env* state has 2^Π successors, each labeled by a different subset of 2^Π, and
3. some state of S satisfies $\langle\langle sys\rangle\rangle\psi$.

Intuitively, a state of S that satisfies $\langle\langle sys\rangle\rangle\psi$ witnesses a strategy of the system to satisfy ψ irrespective of what the environment does. Let S_Π be the maximal two-agent turn-based synchronous ATS over Π that alternates between *sys* and *env* states:

$$S_\Pi = \langle \Pi, \{sys, env\}, 2^\Pi \times \{s, e\}, \pi, \sigma, (2^\Pi \times \{s\}) \times (2^\Pi \times \{e\}) \cup (2^\Pi \times \{e\}) \times (2^\Pi \times \{s\})\rangle,$$

where for every $w \subseteq \Pi$, we have $\pi(\langle w, s\rangle) = \pi(\langle w, e\rangle) = w$, $\sigma(\langle w, s\rangle) = \{sys\}$, and $\sigma(\langle w, e\rangle) = \{env\}$. It is easy to see that ψ is realizable iff there exists some state in S_Π that satisfies $\langle\langle sys \rangle\rangle\psi$. Since the 2EXPTIME lower bound holds already for LTL formulas with a fixed number of propositions, the size of S_Π is fixed, and we are done.

The lower bound for the structure complexity of the problem follows from Theorem 2, and the upper bound follows from fixing $|\psi|$ in the complexity analysis of the joint complexity above. $\square$

6 Beyond ATL*

In this section we suggest two more formalisms for the specification of open systems. We compare the two formalisms with ATL and ATL* and consider their expressiveness and their model-checking complexity. Given two logics L_1 and L_2, we say that the logic L_1 is *as expressive* as the logic L_2 if for every formula φ_2 of L_2, there exists a formula φ_1 of L_1 such that φ_1 and φ_2 are equivalent (i.e., they are true in the same states of each ATS). The logic L_1 is *more expressive* than L_2 if L_1 is as expressive as L_2 and L_2 is not as expressive as L_1.

6.1 The Alternating-time μ-Calculus

The formulas of the logic AMC (*Alternating-time μ-Calculus*) are constructed from propositions, boolean connectives, the next operator $\bigcirc$, each occurrence parameterized by a set of agents, as well as the least fixed-point operator μ. Formally, given a set Π of propositions, a set V of propositional variables, and a set Σ of agents, an AMC formula is one of the following:

- p, for propositions $p \in \Pi$.
- X, for a propositional variables $X \in V$.
- $\neg\varphi$ or $\varphi_1 \vee \varphi_2$, where φ, φ_1, and φ_2 are AMC formulas.
- $\langle\langle A \rangle\rangle \bigcirc \varphi$, where $A \subseteq \Sigma$ is a set of agents and φ is an AMC formula.
- $\mu X.\varphi$, where φ is an AMC formula in which all free occurrences of X (i.e., those that do not occur in a subformula of φ starting with μX) fall under an even number of negations.

The logic AMC is similar to the μ-calculus of [Koz83], only that the next operator $\bigcirc$ is parameterized by sets of agents rather than by a universal or an existential path quantifier. Additional boolean connectives are defined from $\neg$ and $\vee$ in the usual manner. As with ATL, we use the dual $[\![A]\!]\bigcirc\varphi = \neg\langle\langle A \rangle\rangle\bigcirc\neg\varphi$, and the abbreviations $\exists = \langle\langle \Sigma \rangle\rangle$ and $\forall = [\![\Sigma]\!]$. As with the μ-calculus, we write $\nu X.\varphi$ to abbreviate $\neg\mu X.\neg\varphi$. Using both the greatest fixed-point operator ν, the dual next operator $[\![A]\!]\bigcirc$, and the connective $\wedge$, we can write every AMC formula in *positive normal form*, where all occurrences of $\neg$ are in front of propositions. An AMC formula φ is *alternation free* if when φ is written in positive normal form, there are no occurrences of ν (resp. μ) on any syntactic path from an occurrence of μX (resp. νX) to an occurrence of X. For example, the formula $\mu X.(p \vee$

$\mu Y.(X \vee \langle\langle a\rangle\rangle \bigcirc Y))$ is alternation free; the formula $\nu X.\mu Y.((p \wedge X) \vee \langle\langle a\rangle\rangle \bigcirc Y)$ is not. The *alternation-free fragment of AMC* contains only alternation-free formulas.

We now turn to the semantics of AMC. We first need some definitions and notations. Given an ATS $S = \langle \Pi, \Sigma, Q, \pi, \delta\rangle$, a *valuation* $\mathcal{V}$ is a function from the propositional variables V to subsets of Q. For a valuation $\mathcal{V}$, a propositional variable X, and a set $Q' \subseteq Q$ of states, we denote by $\mathcal{V}[X := Q']$ the valuation that maps X to Q' and agrees with $\mathcal{V}$ on all other variables. An AMC formula φ is interpreted as a mapping φ^S from valuations to state sets. Then, $\varphi^S(\mathcal{V})$ denotes the set of states that satisfy the AMC formula φ under the valuation $\mathcal{V}$. The mapping φ^S is defined inductively as follows:

- For a proposition $p \in \Pi$, we have $p^S(\mathcal{V}) = \{q \in Q \mid p \in \pi(q)\}$.
- For a propositional variable $X \in V$, we have $X^S(\mathcal{V}) = \mathcal{V}(X)$.
- $(\neg\varphi)^S(\mathcal{V}) = Q \setminus \varphi^S(\mathcal{V})$.
- $(\varphi_1 \vee \varphi_2)^S(\mathcal{V}) = \varphi_1^S(\mathcal{V}) \cup \varphi_2^S(\mathcal{V})$.
- $(\langle\langle A\rangle\rangle \bigcirc \varphi)^S(\mathcal{V}) = \{q \in Q \mid$ for each agent $a \in A$, there exists a set $Q_a \in \delta(q, a)$ such that for each state $q' \in \bigcap_{a\in A} Q_a$, if q' is a successor of q, then $q' \in \varphi^S(\mathcal{V})\}$.
- $(\mu X.\varphi)^S(\mathcal{V}) = \bigcap\{Q' \subseteq Q \mid \varphi^S(\mathcal{V}[X := Q']) \subseteq Q'\}$.

Consider an AMC formula of the form $\mu X.\varphi$. Then, given a valuation $\mathcal{V}$, the subformula φ can be viewed as a function $h^S_{\varphi,\mathcal{V}}$ that maps each state set $Q' \subseteq Q$ to the state set $\varphi^S(\mathcal{V}[X := Q'])$. Since all free occurrences of X fall under an even number of negations, the function $h^S_{\varphi,\mathcal{V}}$ is monotonic; that is, if $Q' \subseteq Q''$, then $h^S_{\varphi,\mathcal{V}}(Q) \subseteq h^S_{\varphi,\mathcal{V}}(Q'')$. Consequently, by standard fixed-point theory, the function $h^S_{\varphi,\mathcal{S}}$ has a least fixed-point, namely, $\bigcap\{Q' \subseteq Q \mid \varphi^S(\mathcal{V}[X := Q']) \subseteq Q'\}$. Furthermore, if each state has only finitely many successor states, the function $h^S_{\varphi,\mathcal{V}}$ is continuous, and the least fixed-point can be computed by iterative approximation starting from $X = [false]$:

$$(\mu X.\varphi)^S(\mathcal{V}) \;=\; \bigcap_{i\geq 0} (h^S_{\varphi,\mathcal{V}})^i([false]).$$

If the ATS S has only finitely many states, the intersection is finite, and the iterative approximation converges in a finite number of steps.

A *sentence* of AMC is a formula that contains no free occurrences of propositional variables. Sentences φ define the same mapping φ^S for any and all valuations. Therefore, for a state q of S and a sentence φ, we write $S, q \models \varphi$ ("state q satisfies formula φ in structure S") iff $q \in \varphi^S$. For example, the AMC formula $\mu X.(q \vee (p \wedge \langle\langle A\rangle\rangle \bigcirc X))$ is equivalent to the ATL formula $\langle\langle A\rangle\rangle p\,\mathcal{U} q$.

AMC expressiveness All temporal properties using the always and until operators can be defined as fixed points of next-time properties. For closed systems, this gives the μ-calculus as a generalization of temporal logics. It is known that the μ-calculus is more expressive than CTL*, and the alternation-free μ-calculus is more expressive than CTL. Similarly, and for the same reasons, AMC is more expressive than ATL*, and its alternation-free fragment is more expressive than ATL.

Theorem 5. *AMC is more expressive than ATL*. The alternation-free fragment of AMC is more expressive than ATL.*

Proof: The translation from alternating-time temporal logics to AMC is very similar to the translation from branching-time temporal logics to μ-calculus [EL86], with $\langle\langle A\rangle\rangle\bigcirc$ replacing $\exists\bigcirc$. We describe here the translation of ATL formulas to the alternation-free fragment of AMC. For this, we present a function

$$g : \text{ ATL formulas } \rightarrow \text{ alternation-free AMC formulas}$$

such that for every ATL formula φ, the formulas φ and $g(\varphi)$ are equivalent. The function g is defined inductively as follows:

- For $p \in \Pi$, we have $g(p) = p$.
- $g(\neg\varphi) = \neg g(\varphi)$.
- $g(\varphi_1 \vee \varphi_2) = g(\varphi_1) \vee g(\varphi_2)$.
- $g(\langle\langle A\rangle\rangle\bigcirc\varphi) = \langle\langle A\rangle\rangle\bigcirc g(\varphi)$.
- $g(\langle\langle A\rangle\rangle\Box\varphi) = \nu X.(g(\varphi) \wedge \langle\langle A\rangle\rangle\bigcirc X)$.
- $g(\langle\langle A\rangle\rangle\varphi_1 \mathcal{U} \varphi_2) = \mu X.(f(\varphi_2) \vee (g(\varphi_1) \wedge \langle\langle A\rangle\rangle\bigcirc X))$.

To establish that AMC is more expressive than ATL*, and its alternation-free fragment is more expressive than ATL, note that for a single-agent ATS, (alternation-free) AMC is the same as the (alternation-free) μ-calculus, CTL* is the same as ATL*, and CTL is the same as ATL. $\square$

The alternating-time μ-calculus, however, is not a natural and convenient specification language for reasoning about open systems. Writing and understanding formulas in the μ-calculus is hard already in the context of closed systems, and in practice, designers avoid the nonintuitive use of fixed points and prefer simple temporal operators (see [BBG$^+$94]). Using AMC as a specification language for open systems would require even more complicated formulas, with extra nesting of fixed points, making the μ-calculus even less appealing. 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 AMC for the specification of open systems. This is because ATL and ATL* have as primitives parameterized path quantifiers, not just parameterized next-time operators.

AMC model checking Algorithms and tools for μ-calculus model checking can be easily modified to handle AMC. Indeed, the only difference between the μ-calculus and AMC is the definition of the next operator, which has a game-like interpretation in AMC. Hence, as in Section 4.1, the modification involves only the *Pre* function. Therefore, the complexity of the model-checking problem for the μ-calculus [EL86] implies the following.

Theorem 6. *The model-checking problem for the alternation-free fragment of AMC can be solved in time $O(m\ell)$ for an ATS with m transitions and a formula of size ℓ. The model-checking problem for AMC can be solved in time $O(m^{d+1})$ for an ATS with m transitions and an formula of alternation depth $d \geq 1$.*

AMC and propositional logic of games In [Par83], Parikh defines a propositional logic of games. Parikh's logic extends dynamic logics (e.g., PDL [FL79]) in a way similar to the way in which AMC extends the μ-calculus. The formulas in Parikh's logic are built with respect to a set of atomic games, which correspond to the choices of agents in an ATS. Cooperation between agents and fixed-point expressions are specified in Parikh's logic by the usual PDL operations, such as disjunction and iteration, on games. The alternation-free fragment of AMC can be embedded into Parikh's logic. For example, the AMC formula $\mu X.p \vee \langle\langle a,b \rangle\rangle \bigcirc X$ corresponds to the formula $\langle (a \vee b)* \rangle p$ in Parikh's logic. In [Par83], Parikh's logic is shown to be decidable and a complete set of axioms is given; the model-checking problem is not studied.

6.2 Game Logic

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 $\exists\!\!\exists A.\forall$ instead of $\langle\langle A \rangle\rangle$ (read $\exists\!\!\exists A$ as "there exist strategies for the agents in A"). Then, for example, the formula $\hat{\varphi} = \exists\!\!\exists A.(\exists \Box \varphi_1 \wedge \exists \Box \varphi_2)$ 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.

We refer to the general logic with strategy quantifiers, path quantifiers, temporal operators, and boolean connectives as *game logic* (GL, for short). There are three types of formulas in GL: *state formulas*, whose satisfaction is related to a specific state of the given ATS S, *tree formulas*, whose satisfaction is related to a specific execution tree of S (for the definition of execution trees, recall Section 5.3), and *path formulas*, whose satisfaction is related to a specific computation of S. Formally, a GL 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 GL state formulas.
(S3) $\exists\!\!\exists A.\theta$, where $A \subseteq \Sigma$ is a set of agents and θ is a GL tree formula.

A GL tree formula is one of the following:

(T1) φ, for a GL state formula φ.
(T2) $\neg\theta$ or $\theta_1 \vee \theta_2$, where θ, θ_1 and θ_2 are GL tree formulas.
(T3) $\exists\psi$, where ψ is a GL path formula.

A GL path formula is one of the following:

(P1) θ, for a GL tree formula θ.
(P2) $\neg\psi$ or $\psi_1 \vee \psi_2$, where ψ, ψ_1, and ψ_2 are GL path formulas.
(P3) $\bigcirc\psi$ or $\psi_1 \mathcal{U} \psi_2$, where ψ, ψ_1, and ψ_2 are GL path formulas.

The logic GL consists of the set of state formulas generated by the rules (S1–3). For instance, while the formula $\hat{\varphi}$ from above is a GL (state) formula, its subformula $\exists\Box\, \varphi_1 \wedge \exists\Box\, \varphi_2$ is a tree formula.

We now define the semantics of GL. We write $S, q \models \varphi$ to indicate that the state formula φ holds at state q of the structure S. We write $S, t \models \theta$ to indicate that the tree formula θ holds at execution tree t of the structure S. We write $S, t, \lambda \models \psi$ to indicate that the path formula ψ holds at infinite path λ of the execution tree t of the structure S (note that in this case, λ is a computation of S). If t is an execution tree of S, and λ is a node of t, we write $t(\lambda)$ for the subtree of t with root λ. The satisfaction relation $\models$ is defined inductively as follows:

- For formulas generated by the rules (S1–2), the definition is the same as for ATL. For formulas generated by the rules (T2) and (P2), the definition is obvious.
- $q \models \exists A.\theta$ iff there exists a set F_A of strategies, one for each agent in A, so that $exec(q, F_A) \models \theta$.
- $t \models \varphi$ for a state formula φ iff $q \models \varphi$, where q is the root of the execution tree t.
- $t \models \exists\psi$ for a path formula ψ iff there exists a rooted infinite path λ in t such that $t, \lambda \models \psi$.
- $t, \lambda \models \theta$ for a tree formula θ iff $t \models \theta$.
- $t, \lambda \models \bigcirc\psi$ iff $t(\lambda[0,1]), \lambda[1,\infty] \models \psi$,
- $t, \lambda \models \psi_1 \mathcal{U} \psi_2$ iff there exists a position $i \geq 0$ such that $t(\lambda[0,i]), \lambda[i,\infty] \models \psi_2$ and for all positions $0 \leq j < i$, we have $t(\lambda[0,j]), \lambda[j,\infty] \models \psi_1$.

GL expressiveness The logic ATL* is the syntactic fragment of GL that consists of all formulas in which every strategy quantifier is immediately followed by a path quantifier (note that $\exists A.\exists$ is equivalent to $\exists$). Since the formula $\exists A.(\exists\Box\, p \wedge \exists\Box\, q)$ is not equivalent to any ATL* formula, GL is more expressive than ATL*.

Another syntactic fragment of GL is studied in *module checking* [KV96]. There, one considers formulas of the form $\exists A.\theta$, with a single outermost strategy quantifier followed by a CTL or CTL* formula θ. Since the GL formula $\langle\langle A_1\rangle\rangle\Diamond\langle\langle A_2\rangle\rangle\Diamond p$ is not equivalent to any formula with a single outermost strategy quantifier, GL is more expressive than module checking. Furthermore, from an expressiveness viewpoint, alternating-time logics and module checking identify incomparable fragments of game logic. 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. Hence, from a computational viewpoint, ATL is advantageous.

GL model checking The model-checking problem for CTL* can be solved by repeatedly applying, in a bottom-up fashion, an LTL model-checking procedure on subformulas [EL85]. The same technique can be used in order to solve the model-checking problem for GL by repeatedly applying the CTL* module-checking algorithm from [KV96]. The complexity of CTL* module checking then implies the following.

Theorem 7. *The model-checking problem for GL is 2EXPTIME-complete. The structure complexity of the problem is PTIME-complete.*

Thus, game logic is not more expensive than ATL*. We feel, however, that unlike state and path formulas, tree formulas are not natural specifications of reactive systems.

7 Incomplete Information

According to our definition of ATL, every agent has complete information about the state of an ATS. In certain modeling situations it may appropriate, however, to assume that an agent can observe only a subset of the propositions. Then, the strategy of the agent can depend only on the observable part of the history. In this section we study such agents with incomplete information. Using known results on multi-player games with incomplete information, we show that this setting is much more complex than the setting with complete information. Our main result is negative: we show that the ATL model-checking problem is undecidable for cooperating agents with incomplete information. We state this result for our weakest version of ATS, namely, turn-based synchronous ATS.

7.1 ATS with Incomplete Information

A *turn-based synchronous ATS with incomplete information* is a pair $\langle S, P\rangle$ consisting of a turn-based synchronous ATS $S = \langle \Pi, \Sigma, Q, \pi, \sigma, R\rangle$ and a vector $P = \{\Pi_a \mid a \in \Sigma\}$ that contains sets $\Pi_a \subseteq \Pi$ of propositions, one for each agent in Σ. The *observability vector* P defines for each agent a the set Π_a of propositions observable by a. Consider an agent $a \in \Sigma$. For a state $q \in Q$, we term $\pi(q) \cap \Pi_a$ the *a-view* of q. We write $Q_a = 2^{\Pi_a}$ for the set of possible a-views, and $\pi_a : Q \to Q_a$ for the function that maps each state to its a-view. The function π_a is extended to computations of S in the natural way: if $\lambda = q_0, q_1, q_2, \ldots$, then $\pi_a(\lambda) = \pi_a(q_0), \pi_a(q_1), \pi_a(q_2), \ldots$ Two states q and q' are *a-stable* if $\pi(q) \setminus \pi_a(q) = \pi(q') \setminus \pi_a(q')$; that is, q and q' agree on all propositions that a cannot observe. We require that the transition function of a can influence only propositions that a can observe and is independent of propositions that a cannot observe. Formally, we require that the following two conditions hold for all agents $a \in \Sigma$ and all states $q_1, q_1', q_2 \in Q$:

1. If $\sigma(q_1) = a$ and $R(q_1, q_1')$, then q_1 and q_1' are a-stable.
2. If $\sigma(q_1) = \sigma(q_2) = a$ and $\pi_a(q_1) = \pi_a(q_2)$ and $R(q_1, q_1')$, then for every state q_2' such that $\pi_a(q_1') = \pi_a(q_2')$ and q_2 and q_2' are a-stable, we have $R(q_2, q_2')$.

In other words, the transition function of agent a maps each a-view of a state in which a is scheduled into a set of a-views of possible successor states. Accordingly, we define the relation $R_a \subseteq Q_a \times Q_a$ such that $R_a(v, v')$ iff for any and all a-stable states q and q' with $\sigma(q) = a$ and $\pi_a(q) = v$ and $\pi_a(q') = v'$, we have $R(q_1, q_2)$.

7.2 ATL with Incomplete Information

When we specify properties of an ATS with incomplete information using ATL formulas, we restrict ourselves to a syntactic fragment of ATL. To see why, consider the ATL formula $\langle\langle a\rangle\rangle\Diamond p$ for $p \notin \Pi_a$. The formula requires agent a to have a strategy to eventually reach a state in which the proposition p, which a cannot observe, is true. Such a requirement does not make sense. Consequently, whenever a set of agents is supposed to attain a certain task, we require that each agent in the set can observe the propositions that are involved in the task (this includes all propositions that appear in the task as well as all propositions that are observable by agents appearing in the task). Formally, given an observability vector P, we define for each ATL formula φ the set $inv_P(\varphi) \subseteq \Pi$ of *involved propositions*. The definition proceeds by induction on the structure of the formula:

- For $p \in \Pi$, we have $inv_P(p) = \{p\}$.
- $inv_P(\neg\varphi) = arg(\varphi)$.
- $inv_P(\varphi_1 \vee \varphi_2) = inv_P(\varphi_1) \cup inv_P(\varphi_2)$.
- $inv_P(\langle\langle A\rangle\rangle\bigcirc\varphi) = inv_P(\varphi) \cup \bigcup_{a\in A} \Pi_a$.
- $inv_P(\langle\langle A\rangle\rangle\Box\varphi) = inv_P(\varphi) \cup \bigcup_{a\in A} \Pi_a$.
- $inv_P(\langle\langle A\rangle\rangle\varphi_1 \,\mathcal{U} \varphi_2) = inv_P(\varphi_2) \cup inv_P(\varphi_2) \cup \bigcup_{a\in A} \Pi_a$.

The ATL formula φ is *well-formed* with respect to the observability vector P if the following two conditions hold:

1. For every subformula of φ of the form $\langle\langle A\rangle\rangle\bigcirc\theta$ or $\langle\langle A\rangle\rangle\Box\theta$ and for every agent $a \in A$, we have $inv_P(\theta) \subseteq \Pi_a$.
2. For every subformula of φ of the form $\langle\langle A\rangle\rangle\theta_1 \,\mathcal{U} \theta_2$ and for every agent $a \in A$, we have $inv(\theta_1) \cup inv(\theta_2) \subseteq \Pi_a$.

Note that if the formula $\langle\langle A\rangle\rangle\psi$ is well-formed, then each agent in A can observe all propositions that are observable by agents appearing in ψ, but it may not be able to observe some propositions that are observable by other agents in A.

When we interpret an ATL formula φ over a turn-based synchronous ATS $\langle S, P\rangle$ with incomplete information, we require φ to be well-formed with respect to P. The definition of the satisfaction relation is as in the case of complete information (see Section 3.2), except for the following definitions of strategies and outcomes. Now, a *strategy* for an agent $a \in \Sigma$ is a mapping $f_a : Q_a^+ \to Q_a$ such that for all $\chi \in (2^{\Pi_a})^*$ and $v, v' \in \Pi_a$, we have $R_a(v, v')$. Thus, the strategy f_a maps each a-view of a finite computation prefix to the a-view of a possible successor state. Given a state $q \in Q$, a set $A \subseteq \Sigma$ of agents, and a set $F_A = \{f_a \mid a \in A\}$ of strategies, one for each agent in A, a computation $\lambda = q_0, q_1, q_2, \ldots$ is in an *outcome* in $out(q, F_A)$ if $q_0 = q$ and for all positions $i \geq 0$, if $\sigma(q_i) \in A$, then $\pi_a(q_{i+1}) = f_a(\pi_a(\lambda[0, j]))$ for $a = \sigma(q_i)$. Thus, for example, $q \models \langle\langle A\rangle\rangle \bigcirc \varphi$ iff either $\sigma(q) \in A$ and there exists a $\sigma(q)$-view $v \subseteq \Pi_{\sigma(q)}$ such that for all states q' with $R(q, q')$ and $\pi_{\sigma(q)}(q') = v$, we have $q' \models \varphi$, or $\sigma(q) \notin A$ and for all states q' with $R(q, q')$, we have $q' \models \varphi$.

Theorem 8. *The model-checking problem for ATL with incomplete information is undecidable, even in the special case of turn-based synchronous ATS.*

Proof: The outcome problem for multi-player games with incomplete information has been proved undecidable by [Yan97]. This problem is identical to the model-checking problem for the ATL formula $\langle\langle A\rangle\rangle \Diamond p$ on a turn-based synchronous ATS with incomplete information. $\square$

We note that for Fair-ATL, proving undecidability is easier, and follows from undecidability results on asynchronous multi-player games with incomplete information [PR79, PR90].

7.3 Single-agent ATL with Incomplete Information

Single-agent ATL is the fragment of ATL in which every path quantifier is parameterized by a singleton set of agents. In this case, where agents cannot cooperate, the model-checking problem is decidable also for incomplete information. There is an exponential price to be paid, however, over the setting with complete information.

Theorem 9. *The model-checking problem for single-agent ATL with incomplete information is EXPTIME-complete. The structure complexity of the problem is also EXPTIME-complete, even in the special case of turn-based synchronous ATS.*

Proof: We start with the upper bound. Given a turn-based synchronous ATS $\langle S, P\rangle$ and an ATL formula φ, well formed with respect to P, we label the states of S by subformulas of φ, starting as usual from the innermost subformulas. Since φ is well-formed with respect to P, for each subformula of the form $\langle\langle a\rangle\rangle\psi$, the agent a can observe all labels that correspond to subformulas of $\langle\langle a\rangle\rangle\psi$, and we refer to these labels as observable propositions. For subformulas generated by the rules (S1–2), the labeling procedure is straightforward. For subformulas generated by (S3), we proceed as follows. Given a state q of S, and a well-formed ATL formula φ' of the form $\langle\langle a\rangle\rangle \bigcirc p$, $\langle\langle a\rangle\rangle \Box p$, or $\langle\langle a\rangle\rangle p_1 \mathcal{U} p_2$, for an agent a and observable propositions p, p_1, p_2, we define a turn-based synchronous ATS S'

(with complete information) and a state q' of S' such that $\langle S, P\rangle, q \models \varphi'$ iff $S', q' \models \varphi'$. Let $S = \langle \Pi, \Sigma, Q, \pi, \sigma, R\rangle$, and let $\Pi_{\varphi'}$ be the set of observable propositions in φ' (then $\Pi_{\varphi'} \subseteq \Pi_a$). In order to define S', we need the following notations. First, we add to Π_a a special proposition p_a that indicates if agent a is scheduled to proceed; that is, for all states $q \in Q$, we have $q \models p_a$ iff $\sigma(q) = a$. Let $\Pi'_a = \Pi_a \cup \{p_a\}$. For a set $Q_1 \subseteq Q$, an agent $a \in \Sigma$, and an extended a-view $\upsilon \subseteq \Pi'_a$, we define the *$\upsilon$-successor* of Q_1 as the set

$$Q_2 = \{q_2 \in Q \mid \pi_a(q_2) = \upsilon \text{ and there exists a state } q_1 \in Q \text{ such that } R(q_1, q_2)\};$$

that is, Q_2 is the set of all states with extended a-view υ that are successors of some state in Q_1. Now, $S' = \langle \Pi_{\varphi'}, \{a, b\}, Q', \pi', \sigma', R'\rangle$ is defined as follows:

- $Q' \subseteq 2^Q$ is the smallest set satisfying (1) $\{q\} \in Q'$ and (2) for all sets $Q_1 \in Q'$ and all a-views $\upsilon \subseteq \Pi'_a$, the υ-successor of Q_1 is in Q'. Note that for all sets $Q_1 \in Q'$, if $q_1, q_2 \in Q_2$, then q_1 and q_2 have the same a-view, and either agent a is scheduled to proceed in both q_1 and q_2, or a is not scheduled to proceed in both q_1 and q_2. Hence, each state in Q' corresponds to a set of states in Q that are indistinguishable by the agent a.
- For all sets $Q_1 \in Q'$, if $\pi_a(q) = \upsilon$ for any and all $q \in Q_1$, then $\pi'(Q_1) = \upsilon$.
- For all sets $Q_1 \in Q'$, if $\sigma(q) = a$ for any and all $q \in Q_1$, then $\sigma'(Q_1) = a$; otherwise, $\sigma'(Q_1) = b$.
- For all sets $Q_1, Q_2 \in Q'$, we have $R'(Q_1, Q_2)$ iff $R_a(\pi'(Q_1), \pi'(Q_2))$.

It is easy to prove that for each of the three types of φ' we have $\langle S, P\rangle, q \models \varphi'$ iff $S', \{q\} \models \varphi'$. Since the size of S' is exponential in the size of S, membership in EXPTIME follows from Theorem 2.

For the lower bound, we observe that the model-checking problem for the ATL formula $\langle\langle a\rangle\rangle \Diamond p$ on a turn-based synchronous ATS with the two agents a and b and incomplete information is identical to the outcome problem for two-player games with incomplete information. The latter problem is known to be EXPTIME-hard [Rei84]. □

8 Conclusions

Methods for reasoning about closed systems are, in general, not applicable for reasoning about open systems. 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. Indeed, existing methods for the verification of open systems could not circumvent the computational price caused by solving infinite games. The logic ATL introduced here identifies a class of verification problems for open systems for which it suffices to solve iterated finite games. The ensuing linear model-checking complexity for ATL shows that despite the pessimistic results achieved in this area area so far, there is still a great deal of interesting reasoning about open systems that can be performed naturally and efficiently.

While closed systems are naturally modeled as labeled transition systems (Kripke structures), we model open systems as alternating transition systems. In the case of closed systems, ATL degenerates to CTL, Fair-ATL to Fair-CTL [CES86], and ATL* to CTL*. Our model-checking complexity results are summarized in Table 1. All complexities in the table denote tight bounds, where m is the size of the system and ℓ is the length of the formula.

	Closed System	Open System
ATL joint complexity	PTIME [CES86]	PTIME $O(m\ell)$
ATL structure complexity	NLOGSPACE [BVW94]	PTIME
Fair-ATL joint complexity	PTIME [CES86]	PTIME $O(m^2\ell^2)$
Fair-ATL structure complexity	NLOGSPACE [KV95]	PTIME
ATL* joint complexity	PSPACE [CES86]	2EXPTIME $m^{2^{O(\ell)}}$
ATL* sstructure complexity	NLOGSPACE [BVW94]	PTIME

Table 1. Model-checking complexity results

Acknowledgments. We thank Amir Pnueli, Moshe Vardi, and Mihalis Yannakakis for helpful discussions. We also thank Luca de Alfaro and Freddy Mang for their comments on a draft of this manuscript.

References

[AH96] R. Alur and T.A. Henzinger. Reactive modules. In *Proc. 11th IEEE Symposium on Logic in Computer Science*, pages 207–218, 1996.

[AL93] M. Abadi and L. Lamport. Composing specifications. *ACM Transactions on Programming Languages and Systems*, 15(1):73–132, 1993.

[ALW89] M. Abadi, L. Lamport, and P. Wolper. Realizable and unrealizable concurrent program specifications. In *Proc. 16th Int. Colloquium on Automata, Languages and Programming*, volume 372 of *Lecture Notes in Computer Science*, pages 1–17, 1989.

[BBG+94] I. Beer, S. Ben-David, D. Geist, R. Gewirtzman, and M. Yoeli. Methodology and system for practical formal verification of reactive hardware. In *Proc. 6th Conference on Computer Aided Verification*, volume 818 of *Lecture Notes in Computer Science*, pages 182–193, 1994.

[BCM+90] J.R. Burch, E.M. Clarke, K.L. McMillan, D.L. Dill, and L.J. Hwang. Symbolic model checking: 10^{20} states and beyond. In *Proc. 5th Symposium on Logic in Computer Science*, pages 428–439, 1990.

[BVW94] O. Bernholtz, M.Y. Vardi, and P. Wolper. An automata-theoretic approach to branching-time model checking. In *Proc. 6th Conference on Computer Aided Verification*, volume 818 of *Lecture Notes in Computer Science*, pages 142–155, 1994.

[CE81] E.M. Clarke and E.A. Emerson. Design and synthesis of synchronization skeletons using branching time temporal logic. In *Proc. Workshop on Logic of Programs*, volume 131 of *Lecture Notes in Computer Science*, pages 52–71, 1981.

[CES86] E.M. Clarke, E.A. Emerson, and A.P. Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Transactions on Programming Languages and Systems*, 8(2):244–263, 1986.

[CKS81] A.K. Chandra, D.C. Kozen, and L.J. Stockmeyer. Alternation. *Journal of the Association for Computing Machinery*, 28(1):114–133, 1981.

[Cle93] R. Cleaveland. A linear-time model-checking algorithm for the alternation-free modal μ-calculus. *Formal Methods in System Design*, 2:121–147, 1993.

[Dil89] D.L. Dill. *Trace theory for automatic hierarchical verification of speed independent circuits*. MIT Press, 1989.

[EH86] E.A. Emerson and J.Y. Halpern. Sometimes and not never revisited: On branching versus linear time. *Journal of the ACM*, 33(1):151–178, 1986.

[EJ88] E.A. Emerson and C. Jutla. The complexity of tree automata and logics of programs. In *Proc. 29th IEEE Symposium on Foundations of Computer Science*, pages 368–377, 1988.

[EL85] E.A. Emerson and C.-L. Lei. Modalities for model checking: Branching time logic strikes back. In *Proc. 20th ACM Symposium on Principles of Programming Languages*, pages 84–96, 1985.

[EL86] E.A. Emerson and C.-L. Lei. Efficient model checking in fragments of the proposoitional Mu-calculus. In *Proc. 1st Symposium on Logic in Computer Science*, pages 267–278, 1986.

[ES84] E.A. Emerson and A. P. Sistla. Deciding branching time logic. In *Proc. 16th ACM Symposium on Theory of Computing*, 1984.

[FL79] M.J. Fischer and R.E. Ladner. Propositional dynamic logic of regular programs. *Journal of Computer and Systems Sciences*, 18:194–211, 1979.

[GSSL94] R. Gawlick, R. Segala, J. Sogaard-Andersen, and N. Lynch. Liveness in timed and untimed systems. In *Proc. 23rd Int. Colloquium on Automata, Languages and Programming*, volume 820 of *Lecture Notes in Computer Science*, pages 166–177, 1994.

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

[Hol97] G.J. Holzmann. The model checker SPIN. *IEEE Trans. on Software Engineering*, 23(5):279–295, May 1997. Formal Methods in Software Practice.

[Imm81] N. Immerman. Number of quantifiers is better than number of tape cells. *Journal of Computer and System Sciences*, 22(3):384–406, 1981.

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

[KV95] O. Kupferman and M.Y. Vardi. On the complexity of branching modular model checking. In *Proc. 6th Conferance on Concurrency Theory*, volume 962 of *Lecture Notes in Computer Science*, pages 408–422, 1995.

[KV96] O. Kupferman and M.Y. Vardi. Module checking. In *Proc. 8th Conference on Computer Aided Verification*, volume 1102 of *Lecture Notes in Computer Science*, pages 75–86, 1996.

[LP85] O. Lichtenstein and A. Pnueli. Checking that finite state concurrent programs satisfy their linear specification. In *Proc. 12th ACM Symposium on Principles of Programming Languages*, pages 97–107, 1985.

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

[McM93] K.L. McMillan. *Symbolic model checking.* Kluwer Academic Publishers, 1993.

[Par83] R. Parikh. Propositional game logic. In *Proc. 24th IEEE Symposium on Foundation of Computer Science*, pages 195–200, 1983.

[Pnu77] A. Pnueli. The temporal logic of programs. In *Proc. 18th IEEE Symposium on Foundation of Computer Science*, pages 46–57, 1977.

[PR79] G.L. Peterson and J.H. Reif. Multiple-person alternation. In *Proc. 20st IEEE Symposium on Foundation of Computer Science*, pages 348–363, 1979.

[PR89a] A. Pnueli and R. Rosner. On the synthesis of a reactive module. In *Proc. 16th ACM Symposium on Principles of Programming Languages*, 1989.

[PR89b] A. Pnueli and R. Rosner. On the synthesis of an asynchronous reactive module. In *Proc. 16th Int. Colloquium on Automata, Languages and Programming*, volume 372 of *Lecture Notes in Computer Science*, pages 652–671, 1989.

[PR90] A. Pnueli and R. Rosner. Distributed reactive systems are hard to synthesize. In *Proc. 31st IEEE Symposium on Foundation of Computer Science*, pages 746–757, 1990.

[QS81] J.P. Queille and J. Sifakis. Specification and verification of concurrent systems in Cesar. In *Proc. 5th International Symp. on Programming*, volume 137 of *Lecture Notes in Computer Science*, pages 337–351, 1981.

[Rab70] M.O. Rabin. Weakly definable relations and special automata. In *Proc. Symp. Math. Logic and Foundations of Set Theory*, pages 1–23. North Holland, 1970.

[Rei84] J.H. Reif. The complexity of two-player games of incomplete information. *Journal on Computer and System Sciences*, 29:274–301, 1984.

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

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

[Tho95] W. Thomas. On the synthesis of strategies in infinite games. In E.W. Mayr and C. Puech, editors, *Proc. 12th Symp. on Theoretical Aspects of Computer Science*, volume 900 of *Lecture Notes in Computer Science*, pages 1–13, 1995.

[VW86a] M.Y. Vardi and P. Wolper. An automata-theoretic approach to automatic program verification. In *Proc. 1st Symposium on Logic in Computer Science*, pages 322–331, 1986.

[VW86b] M.Y. Vardi and P. Wolper. Automata-theoretic techniques for modal logics of programs. *Journal of Computer and System Science*, 32(2):182–221, 1986.

[Yan97] M. Yannakakis. Synchronous multi-player games with incomplete information are undecidable. Personal communication, 1997.