Alternating-time Temporal Logic*†

Rajeev Alur‡ Thomas A. Henzinger§ Orna Kupferman¶

Abstract

Temporal logic comes in two varieties: linear-time temporal logic assumes implicit universal quantification over all paths that are generated by the execution of a system; 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. The problems of receptiveness, realizability, and controllability can be formulated as model-checking problems for alternating-time formulas. Depending on whether or not we admit arbitrary nesting of selective path quantifiers and temporal operators, we obtain the two alternating-time temporal logics ATL and ATL*.

ATL and ATL* are interpreted over *concurrent game structures*. Every state transition of a concurrent game structure results from a choice of moves, one for each player. The players represent individual components and the environment of an open system. Concurrent game structures can capture various forms of synchronous composition for open systems, and if augmented with fairness constraints, also asynchronous composition. Over structures without fairness constraints, the model-checking complexity of ATL is linear in the size of the game structure and length of the formula, and the symbolic model-checking algorithm for CTL extends with few modifications to ATL. Over structures with weak-fairness constraints, ATL model checking requires the solution of 1-pair Rabin games, and can be done in polynomial time. Over structures with strong-fairness constraints, ATL model checking

*A preliminary version of this paper appeared in the *Proceedings of the 38th Annual Symposium on Foundations of Computer Science* (FOCS), IEEE Computer Society Press, 1997, pp. 100–109.

†This work was supported in part by the ONR YIP award N00014-95-1-0520, by the NSF CAREER awards CCR-9501708 and CCR-9970925, by the NSF grants CCR-9970925 and CCR-9988172 by the DARPA grants NAG2-892 and NAG2-1214, by the SRC contracts 97-DC-324 and 99-TJ-688, and by a Sloan Faculty Fellowship.

‡Department of Computer and Information Science, University of Pennsylvania, Philadelphia, PA 19104. Email: alur@cis.upenn.edu. URL: www.cis.upenn.edu/~alur.

§Department of Electrical Engineering and Computer Sciences, University of California, Berkeley, CA 94720-1770. Email: tah@eecs.berkeley.edu. URL: www.eecs.berkeley.edu/~tah.

¶School of Computer Science and Engineering, Hebrew University, Jerusalem 91904, Israel. Email: orna@cs.huji.ac.il. URL: www.cs.huji.ac.il/~orna

requires the solution of games with Boolean combinations of Büchi conditions, and can be done in PSPACE. 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.

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 are interpreted over Kripke structures. A Kripke structure offers a natural model for the computations of a *closed 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. Modeling languages 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, GSSL94, AL95]: 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 necessary for executability and compositionality to obtain 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 asks for 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 [KVW01]: 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.

These four problems use standard temporal-logic syntax, which was developed for specifying closed systems, and formulate new semantical conditions for open systems. In this paper, we propose, instead, to enrich temporal logic so that alternating properties can be specified explicitly within the logic. For this purpose, we introduce *alternating-time temporal logics*, which are interpreted over game structures. In order to capture compositions of open systems, we consider, instead of two-player games between system and environment, the more general setting of multi-player games, with a set Σ of players that represent different components of the system and the environment [Sha53, HF89].

The Kripke structure is a natural "common-denominator" model for closed systems, independent of whether the high-level description of a system is given, say, as a product of state machines or as a set of guarded commands on variables. In analogy, the natural "common-denominator" model for compositions of open systems is the *concurrent game structure*. While modeling languages for open systems use a variety of different communication mechanisms (variables vs. events, synchronous vs. asynchronous interaction, etc.), they can be given a common semantics in terms of concurrent game structures, which, unlike Kripke semantics, maintains the differentiation of a design into system components and environment. A concurrent game is played on a state space. In each step of the game, every player chooses a move, and the combination of choices determines a transition from the current state to a successor state. Special cases of a concurrent game are *turn-based synchronous* (in each step, only one player has a choice of moves, and that player is determined by the current state), *Moore synchronous* (the state is partitioned according to the players, and in each step, every player updates its own component of the state independently of the other players), and *turn-based asynchronous* (in each step, only one player has a choice of moves, and that player is chosen by a fair scheduler). These subclasses of concurrent games capture various notions of synchronous and asynchronous interaction between open systems.

For a set $A \subseteq \Sigma$ of players, 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 resolves the choices controlled by the players in the set A, while the antagonist resolves the remaining choices. If the resulting infinite computation belongs to the set Λ, then the protagonist wins; otherwise the antagonist 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, $\langle\langle A \rangle\rangle$ can be viewed as a *path quantifier*, parameterized with the set A of players, which ranges over all computations that the players in A can force the game into, irrespective of how the players 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, Kripke structures can be viewed as game structures with a single player *sys*, which represents the system. Then, the two possible parameterized path quantifiers $\langle\langle \{sys\} \rangle\rangle$ (also denoted $\langle\langle sys \rangle\rangle$) and $\langle\langle \emptyset \rangle\rangle$ (also denoted $\langle\langle\ \rangle\rangle$) match exactly the path quantifiers $\exists$ and $\forall$ required for specifying closed 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 predicate. By allowing nesting of alternating properties, we obtain ATL as the alternating-time generalization of CTL, and ATL* as the alternating-time generalization of CTL*. Finally, by considering game structures with fairness constraints (for modeling asynchronous composition), we obtain Fair ATL as the alternating-time generalization of Fair CTL [Eme90].

Alternating-time temporal logics can naturally 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 deadlock state is one in which a processor, say a, is permanently blocked from accessing a memory cell. This requirement was specified using the CTL formula

   $$\forall\Box\ (\exists\Diamond\ \mathit{read}\ \wedge\ \exists\Diamond\ \mathit{write}).$$

   The ATL formula

   $$\forall\Box\ (\langle\langle a \rangle\rangle \Diamond\, \mathit{read}\ \wedge\ \langle\langle a \rangle\rangle \Diamond\, \mathit{write})$$

   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"). A necessary (but not sufficient) condition for φ to be an invariant of a composite system, is that every component a of the system 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 the ATL* formula $\langle\langle sys\rangle\rangle\psi$ in a maximal model that admits all possible inputs and outputs. (We formalize this intuition in Theorem 5.6.)

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 a composite system that contains a as a component, 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 the composite system.

The model-checking problem for alternating-time temporal logics requires the computation of winning strategies. In models without fairness constraints, all games that arise in ATL model checking are *finite* reachability games. Over Kripke structures, existential reachability ($\exists\Diamond$) can be checked by iterating the existential next-time operator $\exists\bigcirc$; universal reachability ($\forall\Diamond$), by iterating the universal next $\forall\bigcirc$. Similarly, over turn-based synchronous game structures, alternating reachability ($\langle\langle A\rangle\rangle\Diamond$) can be checked by iterating an appropriate mix of $\exists\bigcirc$ (whenever a player in A determines the successor state) and $\forall\bigcirc$ (whenever a player in $\Sigma \setminus A$ determines the successor state). Over general concurrent game structures, the next-time operators $\exists\bigcirc$ and $\forall\bigcirc$ need to be generalized to a game-based next, $\langle\langle A\rangle\rangle\bigcirc\varphi$, which characterizes the states from which the players in A can cooperate to ensure that the immediate successor state satisfies φ. The operator $\langle\langle A\rangle\rangle\bigcirc$ can be computed in time linear in the size of the game structure[1], and

[1]Two remarks about complexity are in order, the first well-known, the second particular to concurrent games. First, the number of states, in Kripke as well as game structures, is typically exponential in the high-level system description, for example, if the system description involves Boolean variables. Second, while for system descriptions with Boolean variables, the computation of $\exists\bigcirc$ is in NP (Boolean satisfiability), and $\forall\bigcirc$ is in co-NP (Boolean validity), the computation of $\langle\langle A\rangle\rangle\bigcirc$ may require PSPACE (quantified Boolean formulas) or even NEXPTIME (Henkin-quantified Boolean formulas), already in the restricted case of two players [dAHM00, dAHM01b].

iterated a linear number of times to compute $\langle\langle A\rangle\rangle\Diamond$. This gives a linear-time model-checking procedure for ATL, and indicates how symbolic model checkers for CTL can be modified to check ATL specifications.

In models with weak-fairness constraints, ATL model checking requires the solution of *infinite* games, namely, games whose winning condition is a single Rabin pair. Consequently, the model-checking complexity for ATL with weak-fairness constraints is polynomial in the size of the game structure, namely, $O(m^2 \cdot w^3 \cdot \ell)$ for a game structure with m transitions, w weak-fairness constraints, and a formula of length ℓ. In the special case of *turn-based asynchronous* game structures, the only fairness constraints are on the scheduler, to ensure the fair selection of players. In this case, the winning condition simplifies to a Büchi condition, and the ATL model-checking problem can be solved in time $O(n \cdot m \cdot k^2 \cdot \ell)$ for a turn-based asynchronous game structure with n states, m transitions, and k players, and a formula of length ℓ. In models with strong-fairness constraints, ATL model checking requires solving games where the winning condition is a Boolean combination of Büchi conditions, and can be done in PSPACE, or alternatively, in time $m^{O(w)} \cdot \ell$, for a game structure of size m with w fairness constraints, and a formula of length ℓ. The model-checking problem for ATL* is closely related to the realizability problem for LTL, and therefore much harder, namely, complete for 2EXPTIME.

The paper is organized as follows. Section 2 defines concurrent game structures and considers several special cases. Section 3 defines the alternating-time temporal logics ATL, Fair ATL, and ATL*. Section 4 presents symbolic model-checking procedures, and Section 5 establishes complexity bounds on model checking for ATL, Fair ATL, and ATL*. In Section 6, we discuss more general ways of introducing game quantifiers in temporal logics. Specifically, we define an alternating-time μ-calculus and a temporal game logic, and study their relationship to ATL and ATL*. Finally, Section 7 considers models in which individual players have only partial information about the global state of the system. We show that in this case the model-checking problem for ATL is generally undecidable, and we identify a special case that is decidable in exponential time.

2 Concurrent Game Structures

We model compositions of open systems as concurrent game structures. While in Kripke structures, a state transition represents a step of a closed system, in a concurrent game structure, a state transition results from choices made by the system components and the environment, and represents simultaneous steps by the components and the environment.

2.1 Definition

A (*concurrent*) *game structure* is a tuple $S = \langle k, Q, \Pi, \pi, d, \delta\rangle$ with the following components:

- A natural number $k \geq 1$ of *players*. We identify the players with the numbers $1, \ldots, k$.

- A finite set Q of *states*.
- A finite set Π of *propositions* (also called *observables*).
- For each state $q \in Q$, a set $\pi(q) \subseteq \Pi$ of propositions true at q. The function π is called *labeling* (or *observation*) *function*.
- For each player $a \in \{1, \ldots, k\}$ and each state $q \in Q$, a natural number $d_a(q) \geq 1$ of moves available at state q to player a. We identify the moves of player a at state q with the numbers $1, \ldots, d_a(q)$. For each state $q \in Q$, a *move vector* at q is a tuple $\langle j_1, \ldots, j_k \rangle$ such that $1 \leq j_a \leq d_a(q)$ for each player a. Given a state $q \in Q$, we write $D(q)$ for the set $\{1, \ldots, d_1(q)\} \times \cdots \times \{1, \ldots, d_k(q)\}$ of move vectors. The function D is called *move function*.
- For each state $q \in Q$ and each move vector $\langle j_1, \ldots, j_k \rangle \in D(q)$, a state $\delta(q, j_1, \ldots, j_k) \in Q$ that results from state q if every player $a \in \{1, \ldots, k\}$ chooses move j_a. The function δ is called *transition function*.

The number of states of the structure S is $n = |Q|$. The *number of transitions* of S is $m = \sum_{q \in Q} d_1(q) \times \cdots \times d_k(q)$, that is, the total number of elements in the move function D. Note that unlike in Kripke structures, the number of transitions is not bounded by n^2. For a fixed alphabet Π of propositions, the size of S is $O(m)$.

For two states q and q', we say that q' is a *successor* of q if there is a move vector $\langle j_1, \ldots, j_k \rangle \in D(q)$ such that $q' = \delta(q, j_1, \ldots, j_k)$. Thus, q' is a successor of q iff whenever the game is in state q, the players can choose moves so that q' is 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 of 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 2.1 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 synchronous composition of the two processes by the following concurrent game structure $S_{xy} = \langle k, Q, \Pi, \pi, d, \delta \rangle$:

- $k = 2$. Player 1 represents process a, and player 2 represents process b.
- $Q = \{q, q_x, q_y, q_{xy}\}$. The state q corresponds to $x = y = \mathit{false}$, the state q_x corresponds to $x = \mathit{true}$ and $y = \mathit{false}$, and q_y and q_{xy} have similar interpretations.
- $\Pi = \{x, y\}$. The values of both variables are observable.

- $\pi(q) = \emptyset$, $\pi(q_x) = \{x\}$, $\pi(q_y) = \{y\}$, and $\pi(q_{xy}) = \{x, y\}$.

- – $d_1(q) = d_1(q_y) = 2$ and $d_1(q_x) = d_1(q_{xy}) = 1$. In states q and q_y, move 1 of player 1 leaves the value of x unchanged, and move 2 changes the value of x. In states q_x and q_{xy}, player 1 has only one move, namely, to leave the value of x unchanged.
  – $d_2(q) = d_2(q_x) = 2$ and $d_2(q_y) = d_2(q_{xy}) = 1$. In states q and q_x, move 1 of player 2 leaves the value of y unchanged, and move 2 changes the value of y. In states q_y and q_{xy}, player 2 has only one move, which leaves the value of y unchanged.

- – State q has four successors: $\delta(q, 1, 1) = q$, $\delta(q, 1, 2) = q_y$, $\delta(q, 2, 1) = q_x$, and $\delta(q, 2, 2) = q_{xy}$.
  – State q_x has two successors: $\delta(q_x, 1, 1) = q_x$ and $\delta(q_x, 1, 2) = q_{xy}$.
  – State q_y has two successors: $\delta(q_y, 1, 1) = q_y$ and $\delta(q_y, 2, 1) = q_{xy}$.
  – State q_{xy} has one successor: $\delta(q_{xy}, 1, 1) = q_{xy}$.

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 (out of infinitely many) q-computations of the game structure S_{xy}.

Now suppose that process b can change y from *false* to *true* only when x is already *true*. The resulting game structure S'_{xy} differs from S_{xy} only in the move function, namely, $d'_2(q) = 1$. While $q, q, q_x, q_x, q_x, q_{xy}^\omega$ is a q-computation of 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 resulting game structure S''_{xy} differs from S_{xy} only in the transition function, namely, $\delta''(q, 1, 2) = q$. In state q, the first move of player 2 means "leave y unchanged" as before, but the second move of player 2 now means "change y if player 1 simultaneously changes x, otherwise leave y unchanged." Note that this corresponds to a Mealy-type synchronous composition of the processes a and b, where b reacts to the choice of a within a single state transition of the system. The sequences $q, q, q_x, q_x, q_x, q_{xy}^\omega$ and q, q_{xy}^ω are q-computations of S''_{xy}, but $q, q_y, q_y, q_{xy}^\omega$ is not.

Fourth, suppose we consider process a on its own, as an open system with local variable x and external variable y. In this case, we have again two players, player 1 representing process a as before, but player 2 now representing the environment. Assume that the environment may, in every state, change the value of y arbitrarily but independently of how process a updates x. The resulting game structure S^+_{xy} differs from S_{xy} in the move and transition functions:

- $d_2^+(q) = d_2^+(q_x) = d_2^+(q_y) = d_2^+(q_{xy}) = 2$. In every state, the first move of player 2 sets y to *false*, and the second move sets y to *true*.

- – State q has four successors: $\delta^+(q, 1, 1) = q$, $\delta^+(q, 1, 2) = q_y$, $\delta^+(q, 2, 1) = q_x$, and $\delta^+(q, 2, 2) = q_{xy}$.
  – State q_x has two successors: $\delta^+(q_x, 1, 1) = q_x$ and $\delta^+(q_x, 1, 2) = q_{xy}$.

– State q_y has four successors: $\delta^+(q_y, 1, 1) = q$, $\delta^+(q_y, 1, 2) = q_y$, $\delta^+(q_y, 2, 1) = q_x$, and $\delta^+(q_y, 2, 2) = q_{xy}$.

– State q_{xy} has two successors: $\delta^+(q_{xy}, 1, 1) = q_x$ and $\delta^+(q_{xy}, 1, 2) = q_{xy}$.

Note that S^+_{xy} has strictly more q-computations than S_{xy}.

Finally, consider again process a on its own, but this time with an environment that, in every state, can change the value of y arbitrarily, even dependent of how process a updates x. This is a more powerful environment than in the previous case. The resulting game structure S^*_{xy} has the following move and transition functions:

- $d^*_2(q) = d^*_2(q_x) = d^*_2(q_y) = d^*_2(q_{xy}) = 4$. In every state, the four moves of player 2 correspond to the four Boolean functions that choose the next value of y dependent on the next value of x: move 1 sets y to *false*, move 2 sets y to *true*, move 3 sets y to the next value of x, and move 4 sets y to the complement of the next value of x.

- – $\delta^*(q, 1, 1) = \delta^*(q, 1, 3) = q$, $\delta^*(q, 1, 2) = \delta^*(q, 1, 4) = q_y$, $\delta^*(q, 2, 1) = \delta^*(q, 2, 4) = q_x$, and $\delta^*(q, 2, 2) = \delta^*(q, 2, 3) = q_{xy}$.
  – $\delta^*(q_x, 1, 1) = \delta^*(q_x, 1, 4) = q_x$ and $\delta^*(q_x, 1, 2) = \delta^*(q_x, 1, 3) = q_{xy}$.
  – $\delta^*(q_y, 1, 1) = \delta^*(q_y, 1, 3) = q$, $\delta^*(q_y, 1, 2) = \delta^*(q_y, 1, 4) = q_y$, $\delta^*(q_y, 2, 1) = \delta^*(q_y, 2, 4) = q_x$, and $\delta^*(q_y, 2, 2) = \delta^*(q_y, 2, 3) = q_{xy}$.
  – $\delta^*(q_{xy}, 1, 1) = \delta^*(q_{xy}, 1, 4) = q_x$ and $\delta^*(q_{xy}, 1, 2) = \delta^*(q_{xy}, 1, 3) = q_{xy}$.

While S^*_{xy} has the same q-computations as S^+_{xy}, they describe very different games. For example, in state q of the game structure S^*_{xy}, the environment (player 2) can choose a move (namely, move 3) that ensures that in the next state, both variables x and y have the same value. In state q of the game structure S^+_{xy}, the environment has no such choice. Consequently, only in S^*_{xy} has the environment a strategy to keep x and y equal at all times.
□

A *Kripke structure* (or *labeled transition system*) is the special case of a game structure with a single player, that is, $k = 1$. In this special case, the sole player 1, which represents a closed system, can always choose the successor state on its own. If the number $d_1(q)$ of moves of player 1 at a state q is greater than 1, then the choice of successor state at q is nondeterministic.

We now define two special cases of game structures which commonly arise in the synchronous composition of open systems.

Turn-based synchronous game structures

In a turn-based synchronous game structure, at every state, only a single player has a choice of moves. Formally, a game structure $S = \langle k, Q, \Pi, \pi, d, \delta \rangle$ is *turn-based synchronous* if for every state $q \in Q$, there exists a player $a_q \in \{1, \ldots, k\}$ such that $d_b(q) = 1$ for all players

$b \in \{1, \ldots, k\} \setminus \{a_q\}$. We say that at state q, it is the *turn* of player a_q. Equivalently, a turn-based synchronous game structure can be viewed as a tuple $S = \langle k, Q, \Pi, \pi, \sigma, R \rangle$, where σ: $Q \to \{1, \ldots, k\}$ is a function that maps each state q to the player a_q, and $R \subseteq Q \times Q$ is a total transition relation. Then q' is a successor of q iff $R(q, q')$. Note that in the turn-based synchronous case, the number of transitions is $m = |R| = O(n^2)$, where n is the number of states. Also note that every 1-player structure (Kripke structure) is turn-based synchronous.

Example 2.2 Consider the turn-based synchronous game structure $S_{train} = \langle k, Q, \Pi, \pi, d, \delta \rangle$ shown in Figure 1, which describes a protocol for a train entering a railroad crossing:

- $k = 2$. Player 1 represents the train, and player 2 the gate controller.
- $Q = \{q_0, q_1, q_2, q_3\}$.
- $\Pi = \{out\_of\_gate, in\_gate, request, grant\}$.
- – $\pi(q_0) = \{out\_of\_gate\}$. The train is outside the gate.
  – $\pi(q_1) = \{out\_of\_gate, request\}$. The train is still outside the gate, but has requested to enter.
  – $\pi(q_2) = \{out\_of\_gate, grant\}$. The controller has given the train permission to enter the gate.
  – $\pi(q_3) = \{in\_gate\}$. The train is in the gate.
- – $d_1(q_0) = 2$ and $d_2(q_0) = 1$. At q_0, it is the train's turn. The train can choose to either (move 1) stay outside the gate, in q_0, or (move 2) request to enter the gate and proceed to q_1.
  – $d_1(q_1) = 1$ and $d_2(q_1) = 3$. At q_1, it is the controller's turn. The controller can choose to either (move 1) grant the train permission to enter the gate, or (move 2) deny the train's request, or (move 3) delay the handling of the request.
  – $d_1(q_2) = 2$ and $d_2(q_2) = 1$. At q_2, it is the train's turn. The train can choose to either (move 1) enter the gate or (move 2) relinquish its permission to enter the gate.
  – $d_1(q_3) = 1$ and $d_2(q_3) = 2$. At q_3, it is the controller's turn. The controller can choose to either (move 1) keep the gate closed or (move 2) reopen the gate to new requests.
- – $\delta(q_0, 1, 1) = q_0$ and $\delta(q_0, 2, 1) = q_1$.
  – $\delta(q_1, 1, 1) = q_2$ and $\delta(q_1, 1, 2) = q_0$ and $\delta(q_1, 1, 3) = q_1$.
  – $\delta(q_2, 1, 1) = q_3$ and $\delta(q_2, 2, 1) = q_0$.
  – $\delta(q_3, 1, 1) = q_3$ and $\delta(q_3, 1, 2) = q_0$.

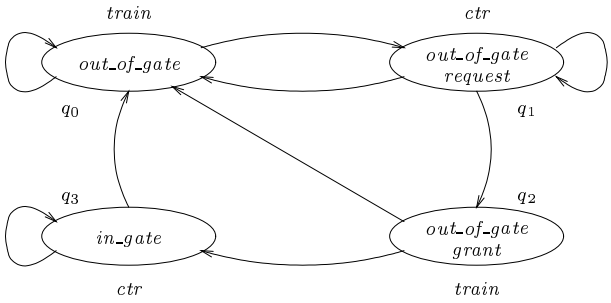

Figure 1: A turn-based synchronous game structure modeling a train controller.

Two states of the system, q_1 and q_3, are controlled; that is, when a computation is in one of these states, the controller chooses the next state. The other two states are uncontrolled, and the train chooses successor states. This gives the following mapping of states to players: $\sigma(q_0) = \sigma(q_2) = 1$ and $\sigma(q_1) = \sigma(q_3) = 2$.
□

Moore synchronous game structures

In a Moore synchronous game structure, the state space is the product of local state spaces, one for each player. In every state, all players proceed simultaneously. Each player chooses its next local state, possibly dependent on the current local states of the other players but independent of the moves chosen by the other players. Formally, a game structure $S = \langle k, Q, \Pi, \pi, d, \delta \rangle$ is *Moore synchronous* if the following two conditions are satisfied:

1. The state space has the form $Q = Q_1 \times \cdots \times Q_k$.
2. For each player $a \in \{1, \ldots, k\}$, each state $q \in Q$, and each move $j \in \{1, \ldots, d_a(q)\}$, there exists a state component $\delta_a(q, j)$ such that $\delta(q, j_1, \ldots, j_k) = \langle \delta_1(q, j_1), \ldots, \delta_k(q, j_k) \rangle$ for all states $q \in Q$ and move vectors $\langle j_1, \ldots, j_k \rangle \in D(q)$.

Thus, every global state $q \in Q$ is a k-tuple $q = \langle q_1, \ldots, q_k \rangle$ of state components q_a, each representing the local state of player a. Each δ_a, for player a, can be viewed as a local transition function that determines the next local state of player a.

Example 2.3 The concurrent game structure S_{xy} from Example 2.1 is Moore synchronous. To see this, note that its state space $Q = \{q, q_x, q_y, q_{xy}\}$ can be viewed as the product of $Q_1 = \{u, u_x\}$ and $Q_2 = \{v, v_y\}$ 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:

- $\delta_1(q, 1) = \delta_1(q_y, 1) = u$ and $\delta_1(q, 2) = \delta_1(q_y, 2) = \delta_1(q_x, 1) = \delta_1(q_{xy}, 1) = u_x$.

- $\delta_2(q,1) = \delta_2(q_x,1) = v$ and $\delta_2(q,2) = \delta_2(q_x,2) = \delta_2(q_y,1) = \delta_2(q_{xy},1) = v_y$.

Also the game structures S'_{xy} and S^+_{xy} from Example 2.1 are Moore synchronous, but the game structures 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 in the same step to x. In S^*_{xy} the environment has this Mealy-type power, of looking at the next value of x before deciding on the next value of y.

□

Moore synchronous game structures arise if a system is described as a synchronous composition of Moore machines. More general concurrent game structures can capture the synchronous composition of Mealy machines, the composition of Reactive Modules [AH96], and generalizations thereof [dAHM00, dAHM01b].

2.2 Fairness Constraints

When closed systems are modeled as Kripke structures, to establish liveness properties, it is often necessary to rule out certain (infinite) computations that ignore enabled moves 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 constraints to Kripke structures. Motivated by similar concerns, we define fairness constraints for game structures.

Consider a concurrent game structure $S = \langle k, Q, \Pi, \pi, d, \delta \rangle$. A *fairness constraint* $\langle a, \gamma \rangle$ consists of a player $a \in \{1, \ldots, k\}$ and a function γ that maps every state $q \in Q$ to a (possibly empty) subset of the moves available at state q to player a, that is, $\gamma(q) \subseteq \{1, \ldots, d_a(q)\}$. A fairness constraint partitions the computations of S into computations that are fair and computations that are not fair. Consider a computation $\lambda = q_0, q_1, q_2, \ldots$ of the game structure S and a fairness constraint $\langle a, \gamma \rangle$. We say that $\langle a, \gamma \rangle$ is *enabled* at position $i \geq 0$ of λ if $\gamma(q_i) \neq \emptyset$. We say that $\langle a, \gamma \rangle$ is *taken* at position i of λ if there is a move vector $\langle j_1, \ldots, j_k \rangle \in D(q_i)$ such that (1) $j_a \in \gamma(q_i)$ and (2) $\delta(q_i, j_1, \ldots, j_k) = q_{i+1}$. We elaborate on two interpretations for fairness constraints:

- The computation λ is *weakly* $\langle a, \gamma \rangle$*-fair* if either there are infinitely many positions of λ at which $\langle a, \gamma \rangle$ is not enabled, or there are infinitely many positions of λ at which $\langle a, \gamma \rangle$ is taken.

- The computation λ is *strongly* $\langle a, \gamma \rangle$*-fair* if either there are only finitely many positions of λ at which $\langle a, \gamma \rangle$ is enabled, or there are infinitely many positions of λ at which $\langle a, \gamma \rangle$ is taken.

With these standard definitions, strong fairness implies weak fairness.

A *weak-fairness condition* Γ_w for the game structure S is a set of fairness constraints that are interpreted in the weak manner: a computation λ of S is Γ_w*-fair* if λ is weakly $\langle a, \gamma \rangle$-fair for all fairness constraints $\langle a, \gamma \rangle \in \Gamma_w$. Similarly, a *strong-fairness condition* Γ_s for the game structure S is a set of fairness constraints that are interpreted in the strong manner: a computation λ of S is Γ_s*-fair* if λ is strongly $\langle a, \gamma \rangle$-fair for all fairness constraints $\langle a, \gamma \rangle \in \Gamma_s$. Note that for every fairness condition Γ (weak or strong), every finite prefix of a computation of S can be extended to a computation that is Γ-fair.

Example 2.4 Consider the concurrent game structure S_{xy} from Example 2.1 and the fairness constraint $\langle 2, \gamma \rangle$ with $\gamma(q) = \gamma(q_x) = \{2\}$ and $\gamma(q_y) = \gamma(q_{xy}) = \emptyset$. Only the computations of S_{xy} in which the value of the variable y is eventually *true* are (weakly or strongly) $\langle 2, \gamma \rangle$-fair. This is because, as long as the value of y is *false*, the game is either in state q or in state q_x. Therefore, as long as the value of y is *false*, the fairness constraint $\langle 2, \gamma \rangle$ is enabled. Hence, in a $\langle 2, \gamma \rangle$-fair computation, $\langle 2, \gamma \rangle$ will eventually be taken; that is, player 2 will eventually choose move 2, thus changing the value of y to *true*.
□

As fairness enables us to exclude some undesirable computations of a game structure, it can be used to model the asynchronous (or interleaving) composition of open systems.

Turn-based asynchronous game structures

In a turn-based asynchronous game structure, one player is designated to represent a *scheduler*. If the set of players is $\{1, \ldots, k\}$, we assume (without loss of generality) that the scheduler is always player k. In every state, the scheduler selects one of the other $k-1$ players, which represent —as usual— the components of the system and the environment. The selected player then determines the next state. Formally, a game structure $S = \langle k, Q, \Pi, \pi, d, \delta \rangle$ is *turn-based asynchronous* if $k \geq 2$ and for every state $q \in Q$, the following two conditions are satisfied:

1. $d_k(q) = k - 1$.
2. For all move vectors $\langle j_1, \ldots, j_k \rangle, \langle j'_1, \ldots, j'_k \rangle \in D(q)$, if $j_k = j'_k$ and $j_a = j'_a$ for $a = j_k$, then $\delta(q, j_1, \ldots, j_k) = \delta(q, j'_1, \ldots, j'_k)$.

We say that player $a \in \{1, \ldots, k-1\}$ is *scheduled* whenever player k (the scheduler) chooses move a. The move chosen by the scheduled player completely determines the next state. Equivalently, a turn-based asynchronous game structure can be viewed as a tuple $S = \langle Q, \Pi, \pi, R_1, \ldots, R_{k-1} \rangle$, where each $R_a \subseteq Q \times Q$ is a total transition relation. Whenever player a is scheduled in a state q, then the next state is chosen so that $R_a(q, q')$. We call each R_a a *component transition relation*. Note that in the turn-based asynchronous case, the number of transitions is $m = \sum_{1 \leq a < k} |R_a| = O(k \cdot n^2)$, where n is the number of states.

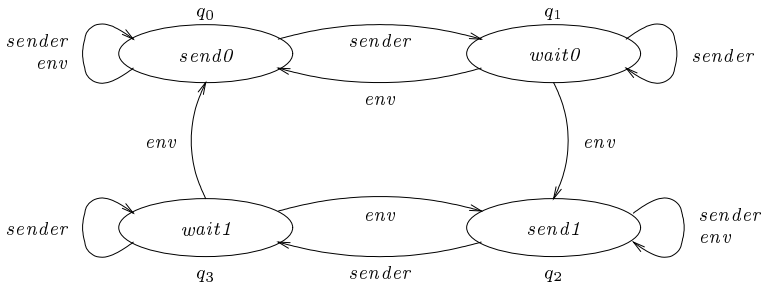

Figure 2: A turn-based asynchronous game structure modeling a message transmission protocol.

We can use fairness constraints to ensure that the scheduler is fair. Specifically, there are $k-1$ fairness constraints of the form $\langle k, \gamma_a \rangle$, one for each player $a \in \{1, \ldots, k-1\}$. The function γ_a, for $a \in \{1, \ldots, k-1\}$, is defined such that $\gamma_a(q) = \{a\}$ for every state $q \in Q$. If interpreted in the weak manner, then the fairness constraint $\langle k, \gamma_a \rangle$ ensures that the scheduler does not neglect player a forever. Note that the choice of selecting a player a is always available to the scheduler, and thus, the strong interpretation of the fairness constraints $\langle k, \gamma_a \rangle$ coincides with its weak interpretation. We therefore associate with every turn-based asynchronous game structure S the weak-fairness condition $\Gamma_w = \{\langle k, \gamma_1 \rangle, \ldots, \langle k, \gamma_{k-1} \rangle\}$.

Example 2.5 As an example of a turn-based asynchronous game structure consider the modeling of the sender process of the alternating-bit protocol shown in Figure 2. The sender is player 1, the environment is player 2, and the scheduler is player 3. In the initial state q_0, if the scheduler selects the sender, it chooses either (move 1) to stay in q_0 or (move 2) to proceed to q_1. The transition from q_0 to q_1 corresponds to sending a message tagged with the bit 0. In q_0, if the scheduler selects the environment, it has no choice of moves, and the game stays in q_0. In state q_1, the sender waits to receive an acknowledgment. If the sender is scheduled, it continues to wait in q_1. If the environment is scheduled, the transition represents 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 is modeled by letting the environment, when scheduled in state q_1, choose between q_2 (move 1) and q_0 (move 2). State q_2 is similar to state q_0, and q_3 is similar to q_1.

Formally, $k = 3$ and $Q = \{q_0, q_1, q_2, q_3\}$. The set Π contains four propositions: *send0* is true in state q_0, *wait0* is true in q_1, *send1* is true in q_2, and *wait1* is true in q_3. The move and transition functions are defined as follows:

- – $d_1(q_0) = d_1(q_2) = 2$ and $d_1(q_1) = d_1(q_3) = 1$.
  – $d_2(q_0) = d_2(q_2) = 1$ and $d_2(q_1) = d_2(q_3) = 2$.
  – $d_3(q_0) = d_3(q_1) = d_3(q_2) = d_3(q_3) = 2$.

- – $\delta(q_0, 1, 1, 1) = \delta(q_0, 1, 1, 2) = \delta(q_0, 2, 1, 2) = q_0$ and $\delta(q_0, 2, 1, 1) = q_1$.
  – $\delta(q_1, 1, 1, 2) = q_2$ and $\delta(q_1, 1, 2, 2) = q_0$ and $\delta(q_1, 1, 1, 1) = \delta(q_1, 1, 2, 1) = q_1$.
  – $\delta(q_2, 1, 1, 1) = \delta(q_2, 1, 1, 2) = \delta(q_2, 2, 1, 2) = q_2$ and $\delta(q_2, 2, 1, 1) = q_3$.
  – $\delta(q_3, 1, 1, 2) = q_2$ and $\delta(q_3, 1, 2, 2) = q_0$ and $\delta(q_3, 1, 1, 1) = \delta(q_3, 1, 2, 1) = q_3$.

This gives rise to the following component transition relations:

- $R_1 = \{(q_0, q_0), (q_0, q_1), (q_1, q_1), (q_2, q_2), (q_2, q_3), (q_3, q_3)\}$.
- $R_2 = \{(q_0, q_0), (q_1, q_2), (q_1, q_0), (q_2, q_2), (q_3, q_0), (q_3, q_2)\}$.

There are two weak-fairness constraints, in order to ensure that the scheduler (player 3) cannot neglect the sender nor the environment forever. The weak-fairness constraint $\langle 3, \gamma_1 \rangle$ guarantees that if the sender is ready to send, in state q_0 or q_2, it will eventually send; the weak-fairness constraint $\langle 3, \gamma_2 \rangle$ guarantees that if the sender waits for an acknowledgment, in state q_1 or q_3, it will eventually receive one:

- $\gamma_1(q_0) = \gamma_1(q_1) = \gamma_1(q_2) = \gamma_1(q_3) = \{1\}$.
- $\gamma_2(q_0) = \gamma_2(q_1) = \gamma_2(q_2) = \gamma_2(q_3) = \{2\}$.

In this example, additional fairness constraints are desirable to ensure progress of the protocol. The assumption that the environment (player 2) cannot keep sending incorrect acknowledgments forever, can be modeled by the strong-fairness constraints $\langle 2, \gamma_e \rangle$ and $\langle 2, \gamma'_e \rangle$:

- $\gamma_e(q_1) = \{1\}$ and $\gamma_e(q_0) = \gamma_e(q_2) = \gamma_e(q_3) = \emptyset$.
- $\gamma'_e(q_3) = \{2\}$ and $\gamma'_e(q_0) = \gamma'_e(q_1) = \gamma'_e(q_2) = \emptyset$.

□

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 $\Sigma = \{1, \ldots, k\}$ of *players*. 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 players, 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 players. Sometimes we write $\langle\langle a_1, \ldots, a_l\rangle\rangle$ instead of $\langle\langle\{a_1, \ldots, a_l\}\rangle\rangle$, and $\langle\langle\ \rangle\rangle$ instead of $\langle\langle\emptyset\rangle\rangle$. Additional Boolean connectives are defined from $\neg$ and $\vee$ in the usual manner. Similar to 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 concurrent game structure S that has the same propositions and players. 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 game between a protagonist and an antagonist. 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 player $a \in A$ a move $j_a \in \{1, \ldots, d_a(u)\}$. Then, the antagonist chooses for every player $b \in \Sigma \setminus A$ a move $j_b \in \{1, \ldots, d_b(u)\}$, and the position of the game is updated to $\delta(u, j_1, \ldots, j_k)$. 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 $\mathcal{U}$; otherwise the antagonist wins. The ATL formula $\langle\langle A\rangle\rangle \psi$ is satisfied at the state q iff 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 a game structure $S = \langle k, Q, \Pi, \pi, d, \delta\rangle$. As before, we write $\Sigma = \{1, \ldots, k\}$ for the set of players. A *strategy* for player $a \in \Sigma$ is a function f_a that maps every nonempty finite state sequence $\lambda \in Q^+$ to a natural number such that if the last state of λ is q, then $f_a(\lambda) \leq d_a(q)$. Thus, the strategy f_a determines for every finite prefix λ of a computation a move $f_a(\lambda)$ for player a. Each strategy f_a for player a induces a set of computations that player a can enforce. Given a state $q \in Q$, a set $A \subseteq \{1, \ldots, k\}$ of players, and a set $F_A = \{f_a \mid a \in A\}$ of strategies, one for each player in A, we define the *outcomes* of F_A from q to be the set $\mathit{out}(q, F_A)$ of q-computations that the players in A enforce when they 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 $i \geq 0$, there is a move vector $\langle j_1, \ldots, j_k\rangle \in D(q_i)$ such that (1) $j_a = f_a(\lambda[0, i])$ for all players $a \in A$, and (2) $\delta(q_i, j_1, \ldots, j_k) = q_{i+1}$.

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

- $q \models p$, for propositions $p \in \Pi$, 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 player 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 player in A, such that for all computations $\lambda \in out(q, F_A)$ and all positions $i \geq 0$, we have $\lambda[i] \models \varphi$.
- $q \models \langle\langle A\rangle\rangle \varphi_1 \,\mathcal{U} \varphi_2$ iff there exists a set F_A of strategies, one for each player 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-time operator $\bigcirc$ is local: $q \models \langle\langle A\rangle\rangle \bigcirc \varphi$ iff for every player $a \in A$, there exists a move $j_a \in \{1, \ldots, d_a(q)\}$ such that for all players $b \in \Sigma \setminus A$ and moves $j_b \in \{1, \ldots, d_b(q)\}$, we have $\delta(q, j_1, \ldots, j_k) \models \varphi$.

Example 3.1 Consider the state q of the concurrent game structure S_{xy} from Example 2.1. We have $S_{xy}, q \models \langle\langle 2\rangle\rangle \bigcirc y$, because player 2 can choose move 2 to set y to *true*, but $S'_{xy}, q \not\models \langle\langle 2\rangle\rangle \bigcirc y$, because in game structure S'_{xy} no such move is available to player 2. We have $S_{xy}, q \not\models \langle\langle 2\rangle\rangle \bigcirc (x = y)$, because if player 2 chooses move 1, then player 1 can choose move 2 to achieve $x \neq y$, and if player 2 chooses move 2, then player 1 can choose move 1 with the same result. However, $S''_{xy}, q \models \langle\langle 2\rangle\rangle \bigcirc (x = y)$, because in game structure S''_{xy} player 2 can choose move 2 to ensure that x and y will have equal values in the next state, which may be q or q_{xy}, depending on whether player 1 chooses move 1 or move 2. Similarly, $S^+_{xy}, q \not\models \langle\langle 2\rangle\rangle \bigcirc (x = y)$ but $S^*_{xy}, q \models \langle\langle 2\rangle\rangle \bigcirc (x = y)$, which confirms that the environment is more powerful in model S^*_{xy} than in model S^+_{xy}. □

It is often useful to express an ATL formula in a dual form. For this purpose, we use the path quantifier $[\![A]\!]$, for a set A of players. While the ATL formula $\langle\langle A\rangle\rangle \psi$ intuitively means that the players in A can cooperate to make ψ true (they can "enforce" ψ), the dual formula $[\![A]\!]\psi$ means that the players in A cannot cooperate to make ψ false (they cannot "avoid" ψ). Using the path quantifier $[\![\ ]\!]$, we can write, for a set A of players and an ATL formula φ, the formula $[\![A]\!] \bigcirc \varphi$ for $\neg\langle\langle A\rangle\rangle \bigcirc \neg\varphi$, the formula $[\![A]\!] \Box \varphi$ 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 $q \in Q$ and a set Λ of q-computations, we say that the players in A can *enforce* the set Λ of computations if there exists a set F_A of strategies, one for each player in A, such that $out(q, F_A) \subseteq \Lambda$. Dually, we say that the players in A can *avoid* the set Λ of computations if there exists a set F_A of strategies, one for each player in A, such that $\Lambda \cap out(q, F_A) = \emptyset$. If the players in A can enforce a set Λ of computations, then the players 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 the concurrent game structure with $k = 2$ and $Q = \{q, q_1, q_2, q_3, q_4\}$. Let $\Pi = \{p\}$

and $\pi(q_1) = \pi(q_4) = \{p\}$ and $\pi(q_2) = \pi(q_3) = \emptyset$. Let $d_1(q) = d_2(q) = 2$ and $\delta(q, 1, 1) = q_1$, $\delta(q, 1, 2) = q_2$, $\delta(q, 2, 1) = q_3$, and $\delta(q, 2, 2) = q_4$. Then $q \not\models \langle\langle 1 \rangle\rangle \bigcirc p$ and $q \models [\![2]\!] \bigcirc p$; that is, in state q, player 1 does not have a strategy to enforce p in the next state, and player 2 does not have a strategy to avoid p in the next state.

Example 3.2 Recall the turn-based synchronous game structure S_{train} from Example 2.2. Recall that in a turn-based synchronous game structure, every state can be labeled with a player that chooses 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, which corresponds to a traditional game played on AND-OR graphs. At a state u, if it is the turn of a player in 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_{train} 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:

$$\langle\langle\ \rangle\rangle \Box\, ((out\_of\_gate \wedge \neg grant) \rightarrow \langle\langle ctr \rangle\rangle \Box\, out\_of\_gate).$$

   For readability, we write *ctr* for the constant 2, and similarly, *train* for the constant 1.

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

$$\langle\langle\ \rangle\rangle \Box\, (out\_of\_gate \rightarrow [\![ctr]\!] \Box\, out\_of\_gate).$$

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

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

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:

$$\langle\langle\ \rangle\rangle \Box\, (out\_of\_gate \rightarrow \langle\langle train \rangle\rangle \Diamond\, (request \wedge (\langle\langle ctr \rangle\rangle \Diamond\, grant) \wedge (\langle\langle ctr \rangle\rangle \Box\, \neg grant))).$$

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

$$\langle\langle\ \rangle\rangle \Box\, (in\_gate \rightarrow \langle\langle ctr \rangle\rangle \bigcirc out\_of\_gate).$$

These natural requirements involve unbounded alternations between universal and existential path quantification and cannot be stated in CTL or CTL*. Consider the first two ATL formulas. They provide more information than the CTL formula

$$\forall \Box\, (out\_of\_gate \rightarrow \exists \Box\, out\_of\_gate).$$

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. (As the train and the controller are the only players in this example, the third ATL formula is, over S_{train}, equivalent to the CTL formula

$$\forall\Box\ (out\_of\_gate \rightarrow \exists\Diamond\ in\_gate).$$

The third ATL formula, however, does not have an equivalent CTL formula over game structures with more than two players, because in general the path quantifier $\langle\langle 1,2\rangle\rangle$ is not equivalent to the existential path quantifier of CTL.)
□

Turn-based game structures

It is worth noting that in the special case of turn-based synchronous as well as turn-based asynchronous game structures, the players in A can enforce a set Λ of computations iff the players in $\Sigma \setminus A$ cannot avoid Λ. This is the property of *determinedness* for turn-based games [BL69, GH82]: in each state, either the players in A can win with objective Λ, or the players not in A can win with the complementary objective. Therefore, for all states q of a turn-based synchronous or asynchronous game structure, $q \models \langle\langle A\rangle\rangle\psi$ iff $q \models [\![\Sigma \setminus A]\!]\psi$, or equivalently, $[\![A]\!] = \langle\langle\Sigma \setminus A\rangle\rangle$. It follows that $\langle\langle A\rangle\rangle\bigcirc\varphi = [\![\Sigma \setminus A]\!]\bigcirc\varphi = \neg\langle\langle\Sigma \setminus A\rangle\rangle\bigcirc\neg\varphi$ over turn-based synchronous and asynchronous game structures. Furthermore, over turn-based game structures we can define the temporal operator $\Box$ from $\Diamond$, namely, $\langle\langle A\rangle\rangle\Box\,\varphi = [\![\Sigma \setminus A]\!]\Box\,\varphi = \neg\langle\langle\Sigma \setminus A\rangle\rangle\Diamond\neg\varphi$.

Single-player structures

Recall that a Kripke structure is a concurrent game structure with a single player, that is, $k = 1$. In this case, which is also a special case of turn-based synchronous, there are only two path quantifiers: $\langle\langle 1\rangle\rangle = [\![\ ]\!]$ and $\langle\langle\ \rangle\rangle = [\![1]\!]$. Then each set $out(q, \{f_1\})$ of outcomes, for some player-1 strategy f_1, contains a single q-computation, and each set $out(q, \emptyset)$ of outcomes contains all q-computations. Accordingly, the path quantifiers $\langle\langle 1\rangle\rangle$ 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 Kripke structures, ATL is identical to CTL. We write, over arbitrary game structures, $\exists$ for the path quantifier $\langle\langle\Sigma\rangle\rangle$, and $\forall$ for the path quantifier $[\![\Sigma]\!]$. This is because, regarding $q \models \exists\psi$, all players can cooperate to enforce a condition ψ iff there exists a q-computation that satisfies ψ. Similarly, regarding $q \models \forall\psi$, all players cannot cooperate to avoid ψ iff all q-computations satisfy ψ.

3.3 Fair ATL

Fairness constraints rule out certain computations. Consequently, in the presence of fairness constraints, 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 protagonist to satisfy all fairness constraints for players in A, and we require the antagonist to satisfy all fairness constraints for players in $\Sigma \setminus A$. This leads us to the following definition. Consider a fairness condition Γ for a game structure S. A strategy f_a for player a is Γ-*fair* if for every computation $\lambda \in out(q, \{f_a\})$ and every fairness constraint of the form $\langle a, \gamma \rangle \in \Gamma$, the computation λ is $\langle a, \gamma \rangle$-fair.

The logic Fair ATL has the same syntax as ATL. The formulas of Fair ATL are interpreted over a concurrent game structure 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 the 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 (both S and Γ are omitted from the satisfaction relation for brevity):

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

Example 3.3 Consider the game structure S_{train} from Example 2.2. 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 Γ containing the single fairness constraint $\langle 2, \gamma_c \rangle$, which imposes fairness on the decisions of the controller (player 2) in state q_1, namely, $\gamma_c(q_1) = \{1\}$ and $\gamma_c(q_0) = \gamma_c(q_2) = \gamma_c(q_3) = \emptyset$. If we interpret Γ as a strong-fairness condition, 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 as a weak-fairness condition, 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 $\langle 2, \gamma_c \rangle$-fair computation $q_0, q_1, q_0, q_1, q_0, q_1, \ldots$
□

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 players 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 players. Additional Boolean connectives and temporal operators are defined from $\neg$, $\vee$, $\bigcirc$, and $\mathcal{U}$ in the usual manner; in particular, $\Diamond\psi = \mathit{true}\,\mathcal{U}\psi$ 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 a concurrent game structure S. We write $S, \lambda \models \psi$ to indicate that the computation λ of the structure S satisfies the path formula ψ. 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 player in A, such that for all computations $\lambda \in \mathit{out}(q, F_A)$, 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

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

asserts that player a has a strategy to enforce computations in which either only finitely many requests are sent, or infinitely many grants are given. Such a requirement can be expressed neither in CTL* nor in ATL.[2]. Since both weak and strong fairness conditions can be expressed within ATL* (provided appropriate propositions are available; see Section 4.2), there is no need for Fair ATL*.

Remark 3.4 In the definitions of ATL and ATL*, the strategy of a player may depend on an unbounded amount of information, namely, the full history of the game up to the current position. As we consider *finite* game structures, all involved games are ω-regular. Therefore, the existence of a winning strategy implies the existence of a winning *finite-state* strategy [BL69, Rab72], which depends only on a finite amount of information about the history of the game. It follows that the semantics of ATL and ATL* (over finite game structures) 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, again, finite game structures. Indeed, for the finite reachability 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 Fair ATL, which gives rise to games with conjunctions of Büchi conditions, nor 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 a game structure $S = \langle k, Q, \Pi, \pi, d, \delta \rangle$ and an ATL formula φ, for the set of states in Q that satisfy φ. We denote the desired set of states by $[\varphi]_S$, or simply by $[\varphi]$ if the game structure S is understood. As usual, let $\Sigma = \{1, \ldots, k\}$. Figure 3 shows a *symbolic* algorithm for ATL model checking, which manipulates state sets of S. The control structure of the algorithm is identical to symbolic algorithms for CTL model checking [BCM$^+$90], but the pre-image operator on Kripke structures used for CTL model checking is replaced by a *Pre* operator on game structures. More precisely, the algorithm uses the following primitive operations:

[2]To see that χ cannot be expressed in ATL, note that in the case of a single-player structure, the formula χ is equivalent to the CTL* formula $\exists((\Diamond \Box \neg req) \vee (\Box \Diamond grant))$, which cannot be expressed in CTL.

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** $\rho := \tau$; $\tau := Pre(A, \rho) \cap [\theta]$ **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 $[\varphi]$.

Figure 3: ATL symbolic model checking.

- The function Sub, when given an ATL formula φ, returns a queue of syntactic 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 set of states in Q that satisfy p.

- The function Pre, when given a set $A \subseteq \Sigma$ of players and a set $\rho \subseteq Q$ of states, returns the set of states q such that from q, the players in A can cooperate and enforce the next state to lie in ρ. Formally, $Pre(A, \rho)$ contains state $q \in Q$ if for every player $a \in A$, there exists a move $j_a \in \{1, \ldots, d_a(q)\}$ such that for all players $b \in \Sigma \setminus A$ and moves $j_b \in \{1, \ldots, d_b(q)\}$, we have $\delta(q, j_1, \ldots, j_k) \in \rho$.

- Union, intersection, difference, and inclusion test for state sets. Note also that we write $[true]$ for the set Q of all states, and $[false]$ for the empty set of states.

Partial correctness of the algorithm can be proved by induction on the structure of the input formula φ. Termination is guaranteed, because the state space Q is finite.

If each state is a valuation for a set X of Boolean variables, then a state set ρ can be encoded by a Boolean expression $\underline{\rho}(X)$ over the variables in X. For Kripke structures that arise from descriptions of closed systems with Boolean state variables, the symbolic operations necessary for CTL model checking have standard implementations. In this case, a transition relation R

on states can be encoded by a Boolean expression $\underline{R}(X, X')$ over X and X', where X' is a copy of X that represents the values of the state variables after a transition. Then the *pre-image* of ρ under R —i.e., the set of states that have R-successors in ρ— can be computed as

$$(\exists X')(\underline{R}(X, X') \wedge \underline{\rho}(X')).$$

Based on this observation, symbolic model checkers for CTL, such as SMV [McM93], typically use ordered binary-decision diagrams (OBDDs) [Bry92] to represent Boolean expressions, and implement the Boolean and pre-image operations on state sets by manipulating OBDDs. In the special case that the game structure S is turn-based synchronous, the symbolic computation of *Pre* is also straightforward. Recall that in this case, the move and transition functions of S can be replaced by a map σ and a transition relation R, such that for every state $q \in Q$, it is the turn of player $\sigma(q)$. Then, when given a set A of players and a set ρ of states, the function *Pre* returns the set of states q such that either $\sigma(q) \in A$ and some R-successor of q lies in ρ, or $\sigma(q) \notin A$ and all R-successors of q lie in ρ. Suppose that $\underline{A}(X)$ is a Boolean expression that encodes the set of states q such that $\sigma(q) \in A$. Then $Pre(\rho)$ can be computed as

$$\begin{array}{l}(\underline{A}(X) \wedge (\exists X')(\underline{R}(X, X') \wedge \underline{\rho}(X'))) \vee \\ (\neg\underline{A}(X) \wedge (\forall X')(\underline{R}(X, X') \rightarrow \underline{\rho}(X')))\end{array}$$

using standard operations on OBDDs.

For general game structures, the situation is more delicate. Typically a game structure arises from a description of an open system with Boolean state variables in a particular system description language. A language for describing open systems has a specific semantics for the parallel composition of open systems. The language, together with its composition semantics, determines a possibly restricted class of game structures that need to be considered, and often suggests a natural symbolic representation for these game structures, i.e., an encoding of the move function and the transition function using Boolean expressions. The symbolic implementation of the *Pre* operator, then, depends on the chosen representation of game structures. Several interesting cases of description languages for open systems with synchronous composition, as well as the computation of the corresponding *Pre* operators, are discussed in [dAHM00, dAHM01b]. One of these languages is Reactive Modules [AH96], for which a symbolic ATL model checker based on OBDDs has been implemented in the verification tool suite MOCHA [AHM+98, AdAG+01].

4.2 Fair ATL Symbolic Model Checking

The *model-checking problem for Fair ATL* asks, given a game structure $S = \langle k, Q, \Pi, \pi, d, \delta\rangle$, a fairness condition Γ for S, and a Fair ATL formula φ, to compute the set of states in Q that fairly satisfy φ with respect to Γ. As will be explained in Section 5.2, Fair ATL model checking can be reduced to model checking of ATL* formulas that have a special form. Here, we consider the special case of turn-based asynchronous game structures. Recall that in a turn-based asynchronous game structure, in every state, the scheduler selects one of the players, and

the selected player determines the next state. A given turn-based asynchronous game structure with players $\Sigma = \{1, \ldots, k\}$, where player $k \geq 2$ is the scheduler, can be viewed as a tuple $S = \langle Q, \Pi, \pi, R_1, \ldots, R_{k-1} \rangle$ with $k-1$ transition relations. A computation $\lambda = q_0, q_1, q_2, \ldots$ is an infinite sequence of states such that for all positions $i \geq 0$, we have $R_a(q_i, q_{i+1})$ for some player $1 \leq a < k$. For $1 \leq a < k$, the (weak) fairness constraint $\langle k, \gamma_a \rangle$ on the scheduler enforces that player a is selected infinitely often, and thus, the computation λ is $\langle k, \gamma_a \rangle$-fair if $R_a(q_i, q_{i+1})$ holds for infinitely many positions $i \geq 0$.

To facilitate Fair ATL model checking, from S we define another turn-based asynchronous game structure, $S^f = \langle Q^f, \Pi^f, \pi^f, R_1^f, \ldots, R_{k-1}^f \rangle$, as follows:

- $Q^f = Q \times \{1, \ldots, k\}$.
- $\Pi^f = \Pi \cup \{done\}$.
- $\pi^f(\langle q, a \rangle) = \pi(q)$ for $1 \leq a < k$, and $\pi^f(\langle q, k \rangle) = \pi(q) \cup \{done\}$.
- For $1 \leq a < k$, the relation R_a^f contains $(\langle q, j \rangle, \langle q', j' \rangle)$ iff $R_a(q, q')$ and either (1) $j = k$ and $j' = 1$, or (2) $j = a$ and $j' = a + 1$, or (3) $1 \leq j < k$ and $j \neq a$ and $j' = j$.

Intuitively, a state of S^f keeps a counter. If the counter is a, then it is incremented when a transition in R_a is taken (i.e., when the scheduler selects the player a). The counter is reset to 1 when it reaches k. The new proposition *done* is true precisely when the counter is k. Thus, the requirement that the scheduler selects every player infinitely often corresponds to the proposition *done* being true infinitely often.

Proposition 4.1 *A state q of the turn-based asynchronous game structure S fairly satisfies an Fair ATL formula of the form $\langle\langle A \rangle\rangle \psi$, where A is a set of players of S, and $\psi = \Box p$ or $\psi = p_1 \mathcal{U} p_2$ for propositions p, p_1, and p_2, with respect to the weak-fairness condition Γ_w of S iff the state $\langle q, 1 \rangle$ of the extended game structure S^f satisfies the ATL* formula $\langle\langle A \rangle\rangle(\Box\Diamond done \rightarrow \psi)$.*

This proposition allows us to develop a symbolic model-checking algorithm for Fair ATL. We consider here only the sample formula $\langle\langle A \rangle\rangle \Diamond p$, for a set $A \subseteq \Sigma$ of players and a proposition p. Consider the following game on the structure S^f, with the players in A being the protagonist, and the players in $\Sigma \setminus A$ the antagonist. Suppose that the current state of the game is q. If the scheduler belongs to A (that is, $k \in A$), then the protagonist either updates the state to q' such that $R_a^f(q, q')$ for some $1 \leq a < k$ and $a \in A$, or the protagonist picks an agent $a \notin A$, and then the antagonist updates the state to q' such that $R_a^f(q, q')$. If the scheduler does not belong to A (that is, $k \notin A$), then the antagonist either updates the state to q' such that $R_a^f(q, q')$ for some $1 \leq a < k$ and $a \notin A$, or the antagonist picks an agent $a \in A$, and then the protagonist updates the state to q' such that $R_a^f(q, q')$. 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 fair, i.e., if the proposition *done* is true only finitely often. The winning condition for the protagonist can therefore be specified by the LTL formula $(\Diamond p) \vee \Diamond \Box \neg done$, or equivalently,

$\rho := [true]$; $\tau := [\neg p]$;
while $\rho \not\subseteq \tau$ **do**
  $\rho := \tau$;
  $\rho' := [false]$; $\tau' := [\rho] \cap [done]$;
  while $\tau' \not\subseteq \rho'$ **do** $\rho' := \rho' \cup \tau'$; $\tau' := Pre^f(\Sigma \setminus A, \rho') \cap [\neg p]$ **od**;
  $\tau := Pre^f(\Sigma \setminus A, \rho') \cap [\neg p]$
od;
return $\hat{\rho} := [true] \setminus \rho$.

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

$\Diamond (p \vee \Box \neg done)$. Since the game is turn-based, and thus determined, the winning condition for the antagonist is obtained by negation, as $\Box (\neg p \wedge \Diamond done)$. This is a Büchi game, and the set of winning states in such a game can be computed using nested fixed points. Note that the CTL* formula $\exists\Box(p_1 \wedge \Diamond p_2)$ can be computed symbolically as the greatest fixpoint

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

Similarly, the algorithm of Figure 4 computes the set $\hat{\rho} \subseteq Q^f$ of winning states for the protagonist.

For a turn-based asynchronous game structure, the set $Pre(A, \rho)$, can be computed as follows. If the scheduler belongs to A, then $Pre(A, \rho)$ contains all states q such that (1) for some $1 \leq a < k$ with $a \in A$, we have $R_a(q, q')$ for some state $q' \in \rho$, or (2) for some $1 \leq a < k$ with $a \notin A$, if $R_a(q, q')$, then $q' \in \rho$. If the scheduler does not belong to A, then $Pre(A, \rho)$ contains all states q such that both (1) for all $1 \leq a < k$ with $a \in A$, we have $R_a(q, q')$ for some state $q' \in \rho$, and (2) for all $1 \leq a < k$ with $a \notin A$, if $R_a(q, q')$, then $q' \in \rho$. In other words, the function Pre can be easily encoded from the encodings of the component transition relations R_a. The function Pre^f is like Pre, but operates on the extended game structure S^f.

5 Model-checking Complexity

We measure the complexity of a model-checking problem in two different ways: the (*joint*) *complexity* of model checking considers the complexity in terms of both the game structure and the formula; the *structure complexity* of model checking considers the complexity in terms of the game structure only, assuming the formula is fixed. Since the game structure is typically much larger than the formula, and its size is the most common computational bottleneck, the structure-complexity measure is of particular practical interest [LP85]. For Fair ATL model checking, the fairness condition is considered together with the game structure, and thus the structure complexity of Fair ATL model checking depends on the size of both the game structure and the fairness condition.

5.1 ATL Model-checking Complexity

The essential subroutines for solving the ATL model-checking problem concern the solution of games with reachability and invariance objectives played on game structures. These games can be solved in linear time on turn-based synchronous game structures [Bee80]. We therefore reduce games played on general game structures to games played on turn-based synchronous game structures. Consider a game structure $S = \langle k, Q, \Pi, \pi, d, \delta \rangle$ and a set $A \subseteq \Sigma$ of players, where $\Sigma = \{1, \ldots, k\}$ as usual. For a state $q \in Q$, an *A-move* c at q is a function that maps each player $a \in A$ to a natural number $c(a) \leq d_a(q)$. The A-move c represents a possible combination of moves at q for the players in A. A state $q' \in Q$ is a *c-successor* of q if there is a move vector $\langle j_1, \ldots, j_k \rangle \in D(q)$ such that (1) $j_a = c(a)$ for all $a \in A$, and (2) $q' = \delta(q, j_1, \ldots, j_k)$. We write $C(A, q)$ for the set of A-moves at q, and $\mathcal{C}(A) = \bigcup_{q \in Q} C(A, q)$ for the set of all A-moves. We build the following 2-player turn-based synchronous game structure $S_A = \langle 2, Q_A, \Pi_A, \pi_A, \sigma_A, R_A \rangle$:

- There is a state for every state of S, and a new state for every A-move: $Q_A = Q \cup \mathcal{C}(A)$.
- There is a special proposition *aux* that identifies the new states: $\Pi_A = \Pi \cup \{aux\}$.
- $\pi_A(q) = \pi(q)$ for all $q \in Q$, and $\pi_A(c) = \{aux\}$ for all $c \in \mathcal{C}(A)$.
- At states $q \in Q$ it is the turn of player 1, and at A-moves $c \in \mathcal{C}(A)$ it is the turn of player 2; that is, $\sigma_A(q) = 1$ for all $q \in Q$, and $\sigma_A(c) = 2$ for all $c \in \mathcal{C}(A)$.
- There is a transition from a state $q \in Q$ to an A-move $c \in \mathcal{C}(A)$ if c is an A-move at q, and there is a transition from c to a state $q' \in Q$ if q' is a c-successor of q. Formally, $R(u, u')$ iff either (1) $u \in Q$ and $u' \in C(A, u)$, or (2) there exists a state $q \in Q$ such that $u \in C(A, q)$ and $u' \in Q$ and u' is a u-successor of q.

If the original game structure S has m transitions, then the turn-based synchronous structure S_A has $O(m)$ states and transitions.

Proposition 5.1 *Let S be a game structure with state space Q, let A be a set of players of S, and let p be a proposition of S. Then, $[\langle\langle A \rangle\rangle \Diamond p]_S = [\langle\langle 1 \rangle\rangle \Diamond p]_{S_A} \cap Q$ and $[\langle\langle A \rangle\rangle \Box p]_S = [\langle\langle 1 \rangle\rangle \Box (p \vee aux)]_{S_A} \cap Q$.*

In other words, in order to solve a reachability or invariance game on S, we can solve a corresponding game on the 2-player turn-based synchronous structure S_A. This gives the following result.

Theorem 5.2 *The model-checking problem for ATL is PTIME-complete, and can be solved in time $O(m \cdot \ell)$ for a game structure with m transitions and an ATL formula of length ℓ. The problem is PTIME-hard even for a fixed formula, and even in the special case of turn-based synchronous game structures.*

Proof. Consider a game structure S with m transitions and an ATL formula φ of length ℓ. We claim that an algorithm that follows the outer loop of Figure 3 can be implemented in time $O(m \cdot \ell)$. The size of $Sub(\varphi)$ is bounded by ℓ. Thus it suffices to show that each case statement can be executed in time $O(m)$. The interesting cases are $\langle\langle A\rangle\rangle \Box\varphi$ and $\langle\langle A\rangle\rangle \varphi_1 \mathcal{U} \varphi_2$. To compute $[\langle\langle A\rangle\rangle \Box\varphi]$ from $[\varphi]$, we apply the second part of Proposition 5.1, choosing a new proposition p with $[p] = [\varphi]$. The resulting invariance game on S_A can be solved in time linear in the size of S^A, that is, in time $O(m)$ [Bee80]. To compute $[\langle\langle A\rangle\rangle \varphi_1 \mathcal{U} \varphi_2]$ from $[\varphi_1]$ and $[\varphi_2]$, we first restrict the game structure S to the states in $[\varphi_1] \cup [\varphi_2]$, and then apply the first part of Proposition 5.1, choosing p such that $[p] = [\varphi_2]$. The resulting reachability game can again be solved in time $O(m)$. This concludes the upper bound.

Since reachability in AND-OR graphs is PTIME-hard [Imm81], and can be specified using the fixed ATL formula $\langle\langle a\rangle\rangle \Diamond p$ interpreted over turn-based synchronous game structures, the lower bounds are immediate.
□

It is interesting to compare the model-checking complexities of ATL and CTL over turn-based synchronous game structures. While both problems can be solved in time $O(m \cdot \ell)$ (for CTL, see [CES86]), the structure complexity of CTL model checking is only NLOGSPACE [KVW00]. This is because CTL model checking is related to graph reachability, whereas ATL model checking is related to AND-OR graph reachability.

5.2 Fair ATL Model-checking Complexity

Consider a game structure $S = \langle k, Q, \Pi, \pi, d, \delta\rangle$ and a fairness condition Γ for S. We need to restrict attention to the computations of S that satisfy all fairness constraints in Γ. To determine which fairness constraints are satisfied by a computation, we augment the state space by adding new propositions that indicate for each fairness constraint $\langle a, \gamma\rangle \in \Gamma$, whether or not $\langle a, \gamma\rangle$ is enabled, and whether or not $\langle a, \gamma\rangle$ is taken. For this purpose, we define the following extended game structure, $S^F = \langle k, Q^F, \Pi^F, \pi^F, d^F, \delta^F\rangle$:

- $Q^F = \{\langle \bot, q\rangle \mid q \in Q\} \cup \{\langle q', q\rangle \mid q', q \in Q$ and q is a successor of q' in $S\}$. Intuitively, a state of the form $\langle \bot, q\rangle$ of S^F corresponds to the game structure S being in state q at the beginning of a computation, and a state of the form $\langle q', q\rangle$ corresponds to S being in state q during a computation whose previous state was q'.
- For each fairness constraint $\langle a, \gamma\rangle \in \Gamma$, there is a new proposition $\langle a, \gamma, \mathit{enabled}\rangle$ and a new proposition $\langle a, \gamma, \mathit{taken}\rangle$; that is, $\Pi^F = \Pi \cup (\Gamma \times \{\mathit{enabled}, \mathit{taken}\})$.
- For each state $\langle \bot, q\rangle \in Q^F$, we have $\pi^F(\langle \bot, q\rangle) = \pi(q)$. For each state $\langle q', q\rangle \in Q^F$, we

have

$$\begin{array}{rcl}\pi^F(\langle q', q\rangle) & = & \pi(q) \cup \\ & & \{\langle a, \gamma, \mathit{enabled}\rangle \mid \gamma(q') \neq \emptyset\} \cup \\ & & \{\langle a, \gamma, \mathit{taken}\rangle \mid \text{ there is a move vector } \langle j_1, \ldots, j_k\rangle \in D(q') \text{ such that} \\ & & \qquad j_a \in \gamma(q') \text{ and } \delta(q', j_1, \ldots, j_k) = q\}.\end{array}$$

- For each player $a \in \Sigma$ and each state $\langle \cdot, q\rangle \in Q^F$, we have $d_a^F(\langle \cdot, q\rangle) = d_a(q)$.
- For each state $\langle \cdot, q\rangle \in Q^F$ and each move vector $\langle j_1, \ldots, j_k\rangle \in D(q)$, we have $\delta^F(\langle \cdot, q\rangle, j_1, \ldots, j_k) = \delta(q, j_1, \ldots, j_k)$.

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 $\Gamma \times \{\mathit{enabled}, \mathit{taken}\}$ 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 as follows.

Proposition 5.3 *A state q of the game structure S fairly satisfies a Fair ATL formula of the form $\langle\langle A\rangle\rangle\psi$, where A is a set of players of S, and $\psi = \Box p$ or $\psi = p_1 \mathcal{U} p_2$ for propositions p, p_1, and p_2, with respect to the weak-fairness condition Γ_w iff the state $\langle \bot, q\rangle$ of the extended game structure S^F satisfies the following ATL* formula:*

$$\begin{array}{l}\langle\langle A\rangle\rangle(\bigwedge_{a\in A, \langle a,\gamma\rangle\in\Gamma_w} \Box \Diamond (\neg\langle a, \gamma, \mathit{enabled}\rangle \vee \langle a, \gamma, \mathit{taken}\rangle) \wedge \\ \quad (\bigwedge_{a\in\Sigma\setminus A, \langle a,\gamma\rangle\in\Gamma_w} \Box \Diamond (\neg\langle a, \gamma, \mathit{enabled}\rangle \vee \langle a, \gamma, \mathit{taken}\rangle) \ \rightarrow \ \psi))\end{array}$$

Moreover, q fairly satisfies $\langle\langle A\rangle\rangle\psi$ with respect to the strong-fairness condition Γ_s iff $\langle \bot, q\rangle$ satisfies the following ATL* formula:

$$\begin{array}{l}\langle\langle A\rangle\rangle(\bigwedge_{a\in A, \langle a,\gamma\rangle\in\Gamma_s} (\Box \Diamond \langle a, \gamma, \mathit{enabled}\rangle \rightarrow \Box \Diamond \langle a, \gamma, \mathit{taken}\rangle) \wedge \\ \quad (\bigwedge_{a\in\Sigma\setminus A, \langle a,\gamma\rangle\in\Gamma_s} (\Box \Diamond \langle a, \gamma, \mathit{enabled}\rangle \rightarrow \Box \Diamond \langle a, \gamma, \mathit{taken}\rangle) \ \rightarrow \ \psi))\end{array}$$

The ATL* formulas that need to be model checked by the above reduction are of a special form, and the corresponding complexity bounds are much lower than those for general ATL* model checking. Let us consider first weak fairness. The *model-checking problem for Weakly-Fair ATL* assumes that the fairness condition on the game structure is a weak-fairness condition.

Theorem 5.4 *The model-checking problem for Weakly-Fair ATL is PTIME-complete, and can be solved in time $O(m^2 \cdot w^3 \cdot \ell)$ for a game structure with m transitions, w weak-fairness constraints, and a Fair ATL formula of length ℓ. Furthermore, for a turn-based asynchronous game structure with n states, m transitions, k players, and a Fair ATL formula of length ℓ, the model-checking problem can be solved in time $O(n \cdot m \cdot k^2 \cdot \ell)$.*

Proof. Consider a game structure S with m transitions and w weak-fairness constraints. Let φ be a Fair ATL formula. The algorithm labels each state of S with all subformulas of φ, starting with the innermost subformulas. Let us consider the case corresponding to a subformula of the form $\langle\langle A\rangle\rangle\psi$. As described earlier, we first construct the extended game structure S^F, and the truth of $\langle\langle A\rangle\rangle\psi$ can be evaluated by solving a game on S^F. The number of states and transitions of S^F is $O(m)$.

We will further augment S^F to simplify the winning condition of the game. Let us partition the weak-fairness constraints in Γ into two sets: Γ_1 contains all constraints of the form $\langle a, \gamma\rangle$ with $a \in A$, and Γ_2 contains the remaining constraints. Suppose that Γ_1 contains the constraints $\langle a^1_1, \gamma^1_1\rangle, \ldots, \langle a^1_{w_1}, \gamma^1_{w_1}\rangle$, and Γ_2 contains the constraints $\langle a^2_1, \gamma^2_1\rangle, \ldots, \langle a^2_{w_2}, \gamma^2_{w_2}\rangle$. We define the game structure S^f_A from S^F by adding two counters. The two counters take their values from the sets $\{1, \ldots, w_i{+}1\}$, for $i = 1, 2$ respectively, and are used to simplify the respective conjunctions $\bigwedge_{\langle a,\gamma\rangle\in\Gamma_i} \Box\Diamond(\neg\langle a, \gamma, \mathit{enabled}\rangle \vee \langle a, \gamma, \mathit{taken}\rangle)$. The states of S^f_A have the form $\langle u, c_1, c_2\rangle$, where u is a state of S^F, and the c_i's are the two counter values. The state component u determines the available moves and the labeling with propositions. The counters c_1 and c_2 are updated deterministically: if c_i equals $w_i + 1$, then it is reset to 1; if $1 \leq c_i \leq w_i$ and u satisfies $\neg\langle a^i_{c_i}, \gamma^i_{c_i}, \mathit{enabled}\rangle \vee \langle a^i, \gamma^i_{c_i}, \mathit{taken}\rangle$, then c_i is incremented; otherwise c_i stays unchanged. Thus, c_i reaches $w_i{+}1$ infinitely often iff each fairness constraint in the corresponding set Γ_i is infinitely often disabled or taken. Consequently, Proposition 5.3 can be restated as: a state q of the game structure S fairly satisfies a Fair ATL formula of the form $\langle\langle A\rangle\rangle\psi$ with respect to the weak-fairness condition Γ iff the state $\langle\bot, q, 1, 1\rangle$ of the extended game structure S^f_A satisfies the formula

$$\langle\langle A\rangle\rangle(\Box\Diamond(c_1 = w_1 + 1) \wedge (\Diamond\Box(c_2 \leq w_2) \vee \psi)).$$

Since the truth of ψ can be encoded in the structure (by doubling the states), the winning condition is a single Rabin pair. The game structure S^f_A has $O(m \cdot w^2)$ states and transitions. The number of states that satisfy the condition $\Box\Diamond(c_1 = w_1 + 1)$ is $O(m \cdot w)$. Using the complexity bounds for solving games with a single Rabin pair [Jur00, EWS01], we get the overall complexity of $O(m^2 \cdot w^3)$. While these bounds are for turn-based games, the reasoning described in Proposition 5.1 can be used to obtain the same bounds also for concurrent game structures.

In the case of turn-based asynchronous structures, recall the construction from Section 4.2. The structure S^f obtained by adding the counter corresponding to the fairness constraints on the scheduler has $O(n \cdot k)$ states and $O(m \cdot k)$ transitions. Checking $\langle\langle A\rangle\rangle\psi$ reduces to solving a (co)Büchi game with the winning condition $\Box\Diamond \mathit{done} \rightarrow \psi$. Since this can be done in time proportional to the product of the number of states and the number of transitions (use the nested fixed-point computation of Figure 4), the cost of processing a temporal operator is $O(n \cdot m \cdot k^2)$.
□

Now let us consider the general case of strong fairness. Proposition 5.3 shows how to reformulate the Fair ATL model-checking problem for a game structure S as an ATL* model-checking

problem for the extended game structure S^F. While such games do not admit a polynomial-time solution, the worst-case bound of 2EXPTIME for ATL* model checking does not apply.

Theorem 5.5 *The model-checking problem for Fair ATL is PSPACE-complete, and can be solved in time $m^{O(w)} \cdot \ell$ for a game structure with m transitions, w fairness constraints, and a Fair ATL formula of size ℓ. The problem is PSPACE-hard even for a fixed formula. For a bounded number of fairness constraints, the problem is PTIME-complete.*

Proof. As usual, the model-checking algorithm labels each state of the extended game structure S^F with all subformulas of the given Fair ATL formula φ, starting with the innermost subformulas. The interesting case corresponds to subformulas of the form $\langle\langle A\rangle\rangle\psi$. This requires solving a game on S^F with the winning condition of the form given by the second part of Proposition 5.3. In [ALM02], it is shown that turn-based games whose condition is a Boolean combination of formulas of the form $\Box\Diamond p$, for propositions p, can be solved in PSPACE, or in time m^n, where m is the size of the game structure and n is the size of the formula. In our case, the size of the winning condition is $O(w)$, where w is the number of fairness constraints. Consequently, each temporal operator can be processed in time $m^{O(w)}$, leading to the overall complexity bound.

For the lower bounds, the construction of [ALM02] can be modified to reduce the satisfaction of a given quantified Boolean formula ϕ to Fair ATL model checking of a fixed formula of the form $\langle\langle a\rangle\rangle\Box p$, for a player a and proposition p, over a 2-player turn-based synchronous game structure (i.e., an AND-OR graph) of size $O(|\phi|)$ with $O(|\phi|)$ strong-fairness constraints.
□

5.3 ATL* Model-checking Complexity

We have seen that the transition from CTL to ATL does not involve a substantial computational price. 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, ATL and ATL*. To see this, we consider the *model-checking problem for ATL**, which asks, given a game structure S and an ATL* (state) formula φ, for the set of states of S that satisfy φ.

Before we discuss ATL* model checking, let us briefly recall CTL* model checking [EL85]. We follow the automata-theoretic approach to model checking. For the definition of word and tree automata on infinite objects, see [Tho90]. 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 (i.e., the players in A).

In order to solve the model-checking problem for ATL*, we first define the notion of execution trees. Consider a game structure $S = \langle k, Q, \Pi, \pi, d, \delta \rangle$, a set $A \subseteq \Sigma$ of players, and a set $F_A = \{f_a \mid a \in A\}$ of strategies for the players in A. For a state $q \in Q$, the set $out(q, F_A)$ of q-computations is fusion-closed, and therefore induces a tree $exec_S(q, F_A)$. Intuitively, the tree $exec_S(q, F_A)$ 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 $exec_S(q, F_A)$ 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 such that there is a move vector $\langle j_1, \ldots, j_k \rangle \in D(q')$ such that (1) $j_a = f_a(\lambda \cdot q')$ for all players $a \in A$, and (2) $\delta(q', j_1, \ldots, j_k) = q''$.

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

Theorem 5.6 *The model-checking problem for ATL* is 2EXPTIME-complete, even in the special case of turn-based synchronous game structures. For ATL* formulas of bounded size, the model-checking problem is PTIME-complete.*

Proof. Consider a game structure 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 [KVW01] 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 that satisfy 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 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 that satisfy $\forall\psi$. Recall that $q \models \langle\langle A \rangle\rangle \psi$ iff there is a set F_A of strategies for the players in A so that all q-computations of S that are outcomes of F_A satisfy ψ. Since each $\langle q, A \rangle$-execution tree corresponds to a set F_A of strategies, it follows that $q \models \langle\langle A \rangle\rangle \psi$ iff the product automaton is nonempty.

The automaton $\mathcal{A}_\psi$ has $2^{2^{O(|\psi|)}}$ states and $2^{O(|\psi|)}$ Rabin pairs [ES84]. In order to define the automaton $\mathcal{A}_{S,q,A}$, we first define, for a state q' and a set A of players, the set $Post(A, q')$ of minimal state sets $\rho \subseteq Q$ so that the players in A can cooperate to ensure that the successor

of q' is a member of ρ. Formally, $\rho \in Post(A, q')$ if (1) for every player $a \in A$, there exists a move $j_a \in \{1, \ldots, d_a(q')\}$ such that for all players $b \in \Sigma \setminus A$ and moves $j_b \in \{1, \ldots, d_b(q')\}$, we have $\delta(q', j_1, \ldots, j_k) \in \rho$, and (2) ρ is minimal, in the sense that no proper subset of ρ satisfies requirement (1). Assume there is an order on the states in Q. Then, we can refer to $Post(A, q')$ as a set of tuples of length at most Q. The tree automaton $\mathcal{A}_{S,q,A}$ has the input alphabet 2^Π, the state set Q, the initial state q, and the nondeterministic transition function η such that for every state $q' \in Q$, we have $\eta(q', \pi(q')) = Post(A, q')$, and $\eta(q', p) = \emptyset$ for all $p \neq \pi(q')$; that is, each set in $Post(A, q)$ determines a set of possible successor states on input $\pi(q')$. All states in Q are Büchi acceptance states. Note that, using the terminology of Section 5.1, each tuple $\rho \in Post(A, q')$ corresponds to an A-move c at q', and ρ contains exactly the c-successors of q'. Thus, the number of tuples in $Post(A, q')$ is $\Pi_{a \in A} d_a(q)$. It follows that the size of the automaton $\mathcal{A}_{S,q,A}$ is bounded by the size of the game structure S. Since the tree automaton $\mathcal{A}_\psi$ is obtained by expanding a deterministic word automaton into a tree automaton, the fact that we regard $Post(A, q')$ as a set of tuples with a single order on the states does not affect the nonemptiness of the product of $\mathcal{A}_\psi$ with $\mathcal{A}_{S,q,A}$. The nonemptiness problem for a Rabin tree automaton of size n with r Rabin pairs can be solved in time $O(n \cdot r)^{3r}$ [EJ88, PR89a]. Hence, labeling a single state with φ' requires at most time $(|S| \cdot 2^{2^{O(|\psi|)}})^{2^{O(|\psi|)}} = |S|^{2^{O(|\psi|)}}$. Since there are $|Q|$ states and at most $|\varphi|$ subformulas, membership in 2EXPTIME follows.

For the lower bound, we reduce the realizability problem for LTL [PR89a], which is 2EXPTIME-hard [Ros92], to ATL* model checking. An LTL formula ψ over a set Π of propositions is realizable iff there exists a 2-player turn-based synchronous game structure S of the following form:

1. The transitions in S alternate between states at which it is the turn of player 1, called player-1 states, and states at which it is the turn of player 2.
2. Every player-1 state has 2^Π successors, each labeled by a different subset of 2^Π.
3. Some state of S satisfies the ATL* formula $\langle\langle 2 \rangle\rangle \psi$.

Intuitively, a state of S that satisfies $\langle\langle 2 \rangle\rangle \psi$ witnesses a strategy for player 2 to satisfy ψ irrespective of how player 1 updates the truth values of propositions. Let S_Π be the maximal 2-player turn-based synchronous game structure over Π that alternates between player-1 states and player-2 states: in transition-relation form,

$$S_\Pi = \langle 2, 2^\Pi \times \{1, 2\}, \Pi, \pi, \sigma, ((2^\Pi \times \{1\}) \times (2^\Pi \times \{2\})) \cup ((2^\Pi \times \{2\}) \times (2^\Pi \times \{1\})) \rangle$$

such that for all $u \subseteq \Pi$, we have $\pi(\langle u, 1 \rangle) = \pi(\langle u, 2 \rangle) = u$ and $\sigma(\langle u, 1 \rangle) = 1$ and $\sigma(\langle u, 2 \rangle) = 2$. Then ψ is realizable iff there exists some state in S_Π that satisfies $\langle\langle 2 \rangle\rangle \psi$. Since the 2EXPTIME lower bound holds for the realizability of LTL formulas with a fixed number of propositions, the size of S_Π is fixed, and the lower bound for the joint complexity of ATL* model checking follows. The lower bound for the structure complexity follows from Theorem 5.2, and the upper bound from fixing $|\psi|$ in the analysis of the joint complexity above.

□

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 game structure). 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-time operator $\bigcirc$, each occurrence parameterized by a set of players, 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 $\Sigma = \{1, \ldots, k\}$ of players, an AMC formula is one of the following:

- p, for propositions $p \in \Pi$.
- X, for 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 players 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 [Koz83], only that the next-time operator $\bigcirc$ is parameterized by sets of players 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 the greatest fixed-point operator ν, the dual next-time 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. As in the μ-calculus, the *alternation depth* of an AMC formula is the maximal length of a chain of nested alternating least and greatest fixed-point operators. In particular, 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 a bound 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 a game structure $S = \langle k, Q, \Pi, \pi, d, \delta\rangle$, 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 $\rho \subseteq Q$ of states, we denote by $\mathcal{V}[X := \rho]$ the valuation that maps X to ρ 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 $\mathcal{V}$ in the structure S. 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 every player $a \in A$, there exists a move $j_a \in \{1, \ldots, d_a(q)\}$ such that for all players $b \in \Sigma \setminus A$ and moves $j_b \in \{1, \ldots, d_b(q)\}$, we have $\delta(q, j_1, \ldots, j_k) \in \varphi^S(\mathcal{V})\}$.
- $(\mu X.\varphi)^S(\mathcal{V}) = \bigcap\{\rho \subseteq Q \mid \varphi^S(\mathcal{V}[X := \rho]) \subseteq \rho\}$.

Consider an AMC formula of the form $\mu X.\varphi$. Given a valuation $\mathcal{V}$, the subformula φ can be viewed as a function $h^S_{\varphi,\mathcal{V}}$ that maps each state set $\rho \subseteq Q$ to the state set $\varphi^S(\mathcal{V}[X := \rho])$. 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 $\rho \subseteq \rho'$, then $h^S_{\varphi,\mathcal{V}}(\rho) \subseteq h^S_{\varphi,\mathcal{V}}(\rho')$. Consequently, by standard fixed-point theory, the function $h^S_{\varphi,\mathcal{V}}$ has a least fixed point, namely, $\bigcap\{\rho \subseteq Q \mid \varphi^S(\mathcal{V}[X := \rho]) \subseteq \rho\}$. The least fixed point can be computed by iterative approximation:

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

As the game structure S has finitely many states, the union 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. A sentence φ defines the same state set $\varphi^S(\mathcal{V})$ for any and all valuations $\mathcal{V}$. Therefore, for a state q of S and a sentence φ, we write $S, q \models \varphi$ ("state q satisfies the formula φ in the structure S") if $q \in \varphi^S(\mathcal{V})$ for any valuation $\mathcal{V}$. 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. The μ-calculus is more expressive than CTL*, and the alternation-free μ-calculus is more expressive than CTL [Eme90, Dam94]. The relationships between AMC, alternation-free AMC, ATL, and ATL* are analogous.

Theorem 6.1 *AMC is more expressive than ATL*. The alternation-free fragment of AMC is more expressive than ATL.*

Proof. First, we define a function G from ATL formulas to 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:

- $G(p) = p$, for propositions $p \in \Pi$.
- $G(\neg\varphi) = \neg G(\varphi)$.
- $G(\varphi_1 \vee \varphi_2) = G(\varphi_1) \vee G(\varphi_2)$.
- $G(\langle\langle A\rangle\rangle \bigcirc \varphi) = \langle\langle A\rangle\rangle \bigcirc G(\varphi)$.
- $G(\langle\langle A\rangle\rangle \Box\, \varphi) = \nu X.(G(\varphi) \wedge \langle\langle A\rangle\rangle \bigcirc X)$.
- $G(\langle\langle A\rangle\rangle \varphi_1 \,\mathcal{U}\, \varphi_2) = \mu X.(G(\varphi_2) \vee (G(\varphi_1) \wedge \langle\langle A\rangle\rangle \bigcirc X))$.

Second, in [dAHM01a] it is shown how a formula of the form $\langle\langle A\rangle\rangle\psi$, where ψ is an LTL formula, can be translated into an equivalent formula $G(\langle\langle A\rangle\rangle\psi)$ of AMC.[3] The function G can be inductively extended to all ATL* formulas. Consider an ATL* formula $\langle\langle A\rangle\rangle\psi$, where ψ is an arbitrary path formula. Let ψ' be the LTL formula that results from ψ by replacing every state subformula φ with a new proposition p_φ. Let $\chi = G(\langle\langle A\rangle\rangle\psi')$ be the AMC formula that is equivalent to $\langle\langle A\rangle\rangle\psi'$. Finally, define $G(\langle\langle A\rangle\rangle\psi)$ to be the result of replacing each new proposition p_φ in χ by the AMC formula $G(\varphi)$. Then $G(\langle\langle A\rangle\rangle\psi)$ is an AMC formula that is equivalent to the ATL* formula $\langle\langle A\rangle\rangle\psi$. This establishes that the alternation-free fragment of AMC is as expressive as ATL, and that AMC is as expressive as ATL*.

To see that AMC is more expressive than ATL*, and that alternation-free AMC is more expressive than ATL, note that for 1-player game structures, (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.
□

Remark 6.2 Parikh has defined a *propositional logic of games* [Par83]. Parikh's logic extends the dynamic logic 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. Each atomic game is a subset of $W \times 2^W$, where W is a set of worlds. Thus, an atomic game corresponds to a single step of a game played on a game structure, with each player choosing a move (or a set of moves) depending on the state of the game structure. Cooperation between players, as well as

[3] It should be noted that if a μ-calculus formula φ is equivalent to $\exists\psi$, for an LTL formula ψ, then it is not necessarily the case that by replacing every occurrence of $\exists\bigcirc$ in φ with $\langle\langle A\rangle\rangle\bigcirc$ we obtain an AMC formula that is equivalent to $\langle\langle A\rangle\rangle\psi$ [dAHM01a].

the iteration of atomic games, are specified in Parikh's logic by standard PDL operations, such as disjunction and iteration. For example, the AMC formula $\mu X.(p \vee \langle\langle a, b\rangle\rangle \bigcirc X)$ corresponds to the formula $\langle (a \vee b)^* \rangle p$ of Parikh's logic. In [Par83], Parikh's logic is shown to be decidable, and a complete set of axioms is given. Our work is motivated by the verification of open systems. Accordingly, we have defined the game versions of logics that are popular for the specification of closed systems, such as temporal logics and the μ-calculus, and unlike [Par83], we focus on the model-checking problem.
□

AMC model checking

The *model-checking problem for AMC* asks, given a game structure S and an AMC sentence φ, for the set of states of S that satisfy φ. The only difference between the classical μ-calculus and AMC is the next-time operator, which has a game interpretation in AMC. Hence, symbolic algorithms and tools for μ-calculus model checking can be modified to handle AMC by implementing the *Pre* function, which computes the next-time operator $\langle\langle\cdot\rangle\rangle\bigcirc$ of AMC (see the discussion in Section 4.1 on possible implementations). From a computational point of view, the μ-calculus and alternation-free μ-calculus model-checking algorithms of [EL86, CS91] can be modified to handle AMC and alternation-free AMC within the same complexity bounds.

We first consider the alternation-free case. The algorithm of [CS91] is based on a bottom-up evaluation of subformulas. Consider a game structure with m transitions and an alternation-free μ-calculus formula φ of length ℓ. The formulas in [CS91] are given in *equational form*; that is, each formula is presented as a set of equational blocks and a propositional variable, called root. An equational block has two forms, $\nu\{E\}$ or $\mu\{E\}$, where E is a list of equations of the form $X_i = \varphi_i$, each φ_i is a μ-calculus formula without fixed-point operators, and the X_i are propositional variables. We assume that the equational blocks are simple; that is, each φ_i contains at most one non-propositional subformula. By adding new propositional variables, we can turn a block that is not simple into a simple block, at a cost that is linear in the length of the formula. If a propositional variable X that appears in the right-hand side of an equation in some block B also appears in the left-hand side of an equation in some *other* block B', then B depends on B'. The formula is alternation-free if the dependencies are not circular. The algorithm in [CS91] processes each equational block in order to evaluate its left-hand side variables. The algorithm proceeds from the minimal blocks, which depend on no other blocks, following a linearization of the dependencies until the root variable is evaluated. Thus, when it reaches a block with an equation $X_i = \varphi_i$, then each propositional variable that appears in φ_i appears either in the left-hand side of another equation of the current block, or it has already been evaluated. The processing of a block requires the repeated application of the *Pre* function: first all left-hand side variables are initialized to true, in case of a ν block, or to false, in case of a μ block, and then the equations are evaluated repeatedly until a fixed point is reached. If the block contains e equations, then the fixed-point computation requires time $O(e \cdot m)$. As ℓ bounds the number of equations in all blocks, the overall complexity $O(m \cdot \ell)$ follows.

In the case of the classical μ-calculus, each calculation of *Pre* is simple, as it corresponds to a universal or an existential next-time operator. In the case of AMC, we need to be more careful in establishing the $O(e \cdot m)$ bound for the repeated application of the *Pre* function in the processing of a block with e equations. Recall that the equational blocks are simple. Thus each equation $X_i = \varphi_i$ is such that φ_i contains at most one subformula of the form $\langle\langle A \rangle\rangle \bigcirc \varphi'$. Following the reasoning described in Proposition 5.1, the repeated evaluations of X_i can be done with respect to the turn-based game structure S_A defined there. Note that φ_i may contain a propositional variable X_j that appears in the left-hand side of the current block, and that φ_j may contain a subformula of the form $\langle\langle B \rangle\rangle \bigcirc \varphi''$ for B different from A. The repeated evaluation of X_i then uses the intermediate values of X_j, which are computed on the turn-based game structure S_B. Still, the repeated calculations of *Pre* required for each equation, which monotonically shrink or grow the left-hand side variable, can be completed in total time $O(m)$. Thus, the evaluation of a block with e equations requires time $O(e \cdot m)$, yielding an overall complexity of $O(m \cdot \ell)$ for model checking the alternation-free fragment of AMC.

Now consider the general case. The algorithm of [EL86] for μ-calculus model checking proceeds also bottom-up, and the evaluation of each subformula involves again repeated applications of the *Pre* function. Here, however, each evaluation may depend on intermediate values of outer subformulas. In particular, the value of a subformula may be updated nonmonotonically, and the number of such updates is bounded by the alternation level of the subformula, i.e., by the number of alternations of fixed-point operators in whose scope the subformula occurs. This leads, for a formula φ of alternation depth d, to a complexity of $O((m \cdot \ell)^{d+1})$ for μ-calculus model checking. In the case of AMC, each repeated monotonic application of the *Pre* function for evaluating a subformula φ' can be performed as in the alternation-free case, in time $O(m \cdot |\varphi'|)$, yielding the overall complexity of $O((m \cdot \ell)^{d+1})$ as well.

Theorem 6.3 *The model-checking problem for the alternation-free fragment of AMC can be solved in time $O(m \cdot \ell)$ for a game structure with m transitions and a formula of size ℓ. The model-checking problem for AMC can be solved in time $O((m \cdot \ell)^{d+1})$ for a game structure with m transitions and a formula of length ℓ and alternation depth $d \geq 1$.*

6.2 Game Logic

The parameterized path quantifier $\langle\langle A \rangle\rangle$ first stipulates the *existence* of strategies for the players in A, and then *universally* quantifies over the outcomes of the stipulated strategies. One may generalize ATL and ATL* by separating the two concerns into strategy quantifiers and path quantifiers, say, by writing $\exists A. \forall$ instead of $\langle\langle A \rangle\rangle$ (read $\exists A$ as "there exist strategies for the players in A"). Then, for example, the formula $\hat{\varphi} = \exists A. (\exists \Box \varphi_1 \wedge \exists \Box \varphi_2)$ asserts that the players in A have strategies such that for some behavior of the remaining players, φ_1 is always true, and for some possibly different behavior of the remaining players, 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 a given game structure 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, given a set Π of propositions and a set Σ of players, 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 players 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 with respect to a game structure S. We write $S, q \models \varphi$ to indicate that the state q of the structure S satisfies the state formula φ; we write $S, t \models \theta$ to indicate that the execution tree t of the structure S satisfies the tree formula θ; and we write $S, t, \lambda \models \psi$ to indicate that the rooted infinite path λ of the execution tree t of the structure S satisfies the path formula ψ (note that in this case, λ is a computation of S). If t is an execution tree of S, and x is a node of t, we write $t(x)$ for the subtree of t with root x. 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.
- $S, q \models \exists\!\!\exists A.\, \theta$ iff there exists a set F_A of strategies, one for each player in A, such that $S, \mathit{exec}_S(q, F_A) \models \theta$.
- $S, t \models \varphi$ for a state formula φ iff $S, q \models \varphi$, where q is the root of the execution tree t.

- $S, t \models \exists\psi$ for a path formula ψ iff there exists a rooted infinite path λ in t such that $S, t, \lambda \models \psi$.
- $S, t, \lambda \models \theta$ for a tree formula θ iff $S, t \models \theta$.
- $S, t, \lambda \models \bigcirc\psi$ iff $S, t(\lambda[0,1]), \lambda[1,\infty] \models \psi$.
- $S, t, \lambda \models \psi_1 \,\mathcal{U}\, \psi_2$ iff there exists a position $i \geq 0$ such that $S, t(\lambda[0,i]), \lambda[i,\infty] \models \psi_2$ and for all positions $0 \leq j < i$, we have $S, t(\lambda[0,j]), \lambda[j,\infty] \models \psi_1$.

Note that whenever a strategy quantifier is applied, the tree formula in its scope is evaluated with respect to execution trees of S, even when the strategy quantifier is in a scope of another strategy quantifier. Thus, for example, the GL formula $\exists A_1.\, \exists A_2.\, \theta$ is equivalent to the GL formula $\exists A_2.\, \theta$. This is analogous to the semantics of nested path quantifiers in CTL*, where, for example, $\exists\forall\psi$ is equivalent to $\forall\psi$.

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 $\exists A.\, \forall$ is equivalent to $\langle\langle A\rangle\rangle$). In particular, the formula $\hat{\varphi}' = \exists\{a\}.\, (\exists\Box\, p \wedge \exists\Box\, q)$, for a player a and two different propositions p and q, is not equivalent to any ATL* formula. It follows that GL is more expressive than ATL*. GL and AMC are incomparable generalizations of ATL*: the GL formula $\hat{\varphi}'$ is not equivalent to any AMC formula; over 1-player game structures, GL is the same as CTL*, and thus not as expressive as the (alternation-free) μ-calculus.

Theorem 6.4 *GL is more expressive than ATL* but not as expressive as AMC. AMC is not as expressive as GL.*

A syntactic fragment of GL different from ATL/ATL* is studied in *module checking* [KVW01]. There, one considers formulas of the form $\exists A.\, \theta$, with a single outermost strategy quantifier followed by a CTL or CTL* formula θ. The GL formula $\langle\langle a\rangle\rangle\Diamond\langle\langle b\rangle\rangle\Diamond\, p$ involves strategies for two different players, a and b, and is not equivalent to any formula with a single outermost strategy quantifier. It follows that GL is more expressive than module checking. Thus, from an expressiveness point of view, alternating-time temporal logics and module checking identify incomparable fragments of game logic. In [KVW01], it is shown that the module-checking problem is EXPTIME-complete for CTL and 2EXPTIME-complete for CTL*, and the structure complexity of both problems is PTIME. Hence, from a computational point of view, ATL is advantageous.

GL model checking

The *model-checking problem for GL* asks, given a game structure S and a GL (state) formula φ, for the set of states of S that satisfy φ. 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 [KVW01]. The complexity of CTL* module checking then implies the following.

Theorem 6.5 *The model-checking problem for GL is 2EXPTIME-complete. For GL formulas of bounded size, the model-checking problem is PTIME-complete.*

7 Incomplete Information

According to our definition of ATL, every player has complete information about the state of a game structure. In certain modeling situations it may be appropriate, however, to assume that a player can observe only a subset of the propositions. Then, the strategy of the player can depend only on the observable part of the history of the game. In this section we study such players 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 players with incomplete information. We present this result for the special case of turn-based synchronous game structures.

7.1 Game Structures with Incomplete Information

A *turn-based synchronous game structure with incomplete information* is a pair $\langle S, P\rangle$ that consists of a turn-based synchronous game structure $S = \langle k, Q, \Pi, \pi, \sigma, R\rangle$ and a vector $P = \{\Pi_a \mid 1 \leq a \leq k\}$ of k sets $\Pi_a \subseteq \Pi$ of propositions, one for each player. Recall that σ is a map from the states Q to the players $\Sigma = \{1, \ldots, k\}$ such that at state q, it is the turn of player $\sigma(q)$, and $R \subseteq Q \times Q$ is a total transition relation. The *observability vector* P defines for each player $a \in \Sigma$ the set Π_a of propositions *observable* by a. For each player $a \in \Sigma$, we assume that there is a proposition $p_a \in \Pi_a$ such that $[p_a] = \{q \mid \sigma(q) = a\}$. Thus, player a can observe when it is its turn, but it might not observe $\sigma(q)$ for states q with $\sigma(q) \neq a$. Consider a player $a \in \Sigma$. A subset $v \subseteq \Pi_a$ is called an *a-view*, and the set of a-views is denoted $V_a = 2^{\Pi_a}$. The function π_a maps each state $q \in Q$ to the corresponding a-view $\pi_a(q) = \pi(q) \cap \Pi_a$, and the function $\pi_{\tilde{a}}$ maps each state $q \in Q$ to the set $\pi_{\tilde{a}}(q) = \pi(q) \setminus \Pi_a$ of propositions that hold in q but a cannot observe. The function π_a is extended to computations in the natural way: if $\lambda = q_0, q_1, q_2, \ldots$, then $\pi_a(\lambda) = \pi_a(q_0), \pi_a(q_1), \pi_a(q_2), \ldots$ We require that whenever it is player a's turn, then the transition relation can influence only propositions that a can observe, and is independent of propositions that a cannot observe. An exception are propositions of the form p_b, for $b \in \Sigma$, which may became valid in the target state. Formally, we require that the following two conditions hold for all players $a \in \Sigma$ and all states $q_1, q_1', q_2 \in Q$:

1. If $\sigma(q_1) = a$ and $R(q_1, q_1')$, then $\pi_{\tilde{a}}(q_1) = \pi_{\tilde{a}}(q_1') \setminus \{p_{\sigma(q_1')}\}$.

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 $R(q_2, q_2')$ for all states q_2' with $\pi_a(q_2') = \pi_a(q_1')$ and $\pi_{\tilde{a}}(q_2') \setminus \{p_{\sigma(q_2')}\} = \pi_{\tilde{a}}(q_2)$.

In other words, we can view the transition relation R as a vector of *player transition relations* $R_a \subseteq V_a \times V_a$, one for each player $a \in \Sigma$. The player transition relation R_a specifies for each a-view a set of possible successor a-views: for all a-views v and v', we have $R_a(v, v')$ iff for all pairs $\langle q, q' \rangle$ of states with $\sigma(q) = a$ and $\pi_a(q) = v$ and $\pi_a(q') = v'$ and $\pi_{\tilde{a}}(q) = \pi_{\tilde{a}}(q') \setminus \{p_{\sigma(q')}\}$, we have $R(q, q')$.

7.2 ATL with Incomplete Information

When we specify properties of a game structure 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 a proposition $p \notin \Pi_a$ that cannot be observed by player a. The formula requires player a to have a strategy to reach a state in which the proposition p is true. As a cannot observe p, this requirement makes no sense. Consequently, whenever a set of players is supposed to attain a certain task, we require that each player 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 players appearing in the task. Consider, for example, the ATL formula $\langle\langle a \rangle\rangle \Diamond \langle\langle b \rangle\rangle \Diamond p$, for two players a and b. If a cannot observe all propositions that b can observe, then a cannot determine whether its task, to reach a state that satisfies $\langle\langle b \rangle\rangle \Diamond p$, is accomplished. Hence we require that $\Pi_b \subseteq \Pi_a$, as well as $p \in \Pi_b$.

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:

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

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 player $a \in A$, 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 player $a \in A$, 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 player in A can observe all propositions that are observable by players that appear in ψ, but it may not be able to observe some propositions that are observable by other players in A.

When we interpret an ATL formula φ over a turn-based synchronous game structure $\langle S, P\rangle$ with incomplete information, we require that φ is well-formed with respect to P. The definition of the satisfaction relation is the same as in the case of complete information (see Section 3.2), except for the following definitions of strategies and outcomes. Now, a *strategy* for player $a \in \Sigma$ is a function f_a: $V_a^+ \to V_a$ that maps every nonempty, finite sequence κ of a-views to an a-view such that if the last a-view of κ is v, then $R_a(v, f_a(\kappa))$. Thus, the strategy f_a looks at the a-views of a finite computation prefix and suggests an a-view for the next state (the suggestion is followed if it is the turn of player a). Given a state $q \in Q$, a set $A \subseteq \Sigma$ of players, and a set $F_A = \{f_a \mid a \in A\}$ of strategies, one for each player 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 $\pi_a(q_{i+1}) = f_a(\pi_a(\lambda[0, i]))$ 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 $q' \models \varphi$.

The *model-checking problem for turn-based synchronous ATL with incomplete information* asks, given a turn-based synchronous game structure $\langle S, P\rangle$ with incomplete information, and an ATL formula φ that is well-formed with respect to P, for the set of states of S that satisfy φ.

Theorem 7.1 *The model-checking problem for turn-based synchronous ATL with incomplete information is undecidable.*

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 game structure with incomplete information.

□

We note that for Fair ATL over turn-based asynchronous game structures with weak-fairness constraints for the scheduler, and incomplete information for the other players, proving undecidability is easier, and follows from undecidability results on asynchronous multi-player games with incomplete information [PR79, PR90].

7.3 Single-player ATL with Incomplete Information

Single-player ATL is the fragment of ATL in which every path quantifier is parameterized by a singleton set of players. In this case, players cannot cooperate, and 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 7.2 *The model-checking problem for single-player turn-based synchronous ATL with incomplete information is EXPTIME-complete. The problem is EXPTIME-hard even for a fixed formula.*

Proof. We start with the upper bound. Given a turn-based synchronous game structure $\langle S, P\rangle$ with incomplete information and a single-player ATL formula φ that is well-formed with respect to P, we label the states of S with subformulas of φ, starting as usual from the innermost subformulas. For subformulas generated by the rules (S1–2), the labeling procedure is straightforward. For subformulas φ' generated by (S3), we proceed as follows.

Let $S_a = \langle 2, V_a, \Pi_a, id, \sigma_a, R_a\rangle$ be the game structure S as observed by player a: id is the identity function, and for all states $v \in V_a$, define $\sigma_a(v) = 1$ if $p_a \in v$, and $\sigma_a(v) = 2$ otherwise. Note that S_a is a 2-player game structure —player 1 corresponds to player a, and player 2 corresponds to all other players— and player 1 has complete information in this game. We construct the extended game structure $S'_a = \langle 2, V_a, \Pi'_a, \pi'_a, \sigma_a, R_a\rangle$ by adding the following new propositions. If $\varphi' = \langle\langle a\rangle\rangle \bigcirc \theta$ or $\varphi' = \langle\langle a\rangle\rangle \Box\, \theta$, then $\Pi'_a = \Pi_a \cup \{p_\theta\}$ with $[p_\theta]_{S'_a} = [\theta]_{S_a}$; that is, the new proposition p_θ represents the label that corresponds to the proper subformula θ. Similarly, if $\varphi' = \langle\langle a\rangle\rangle \theta_1 \,\mathcal{U}\, \theta_2$, then $\Pi'_a = \Pi_a \cup \{p_{\theta_1}, p_{\theta_2}\}$ with $[p_{\theta_1}]_{S'_a} = [\theta_1]_{S_a}$ and $[p_{\theta_2}]_{S'_a} = [\theta_2]_{S_a}$. Since φ is well-formed with respect to P, player 1 can solve the ATL model-checking problems for the subformulas θ, θ_1, and θ_2 in S_a, and thus can observe the propositions in Π'_a. In particular, if θ, θ_1, or θ_2 contain a path quantifier $\langle\langle b\rangle\rangle$ with $b \neq a$, then $\Pi_b \subseteq \Pi_a$, and the game structure S_a refines the game structure S_b. If $\varphi' = \langle\langle a\rangle\rangle \bigcirc \theta$, let $\varphi'' = \langle\langle 1\rangle\rangle \bigcirc p_\theta$; if $\varphi' = \langle\langle a\rangle\rangle \Box\, \theta$, let $\varphi'' = \langle\langle 1\rangle\rangle \Box\, p_\theta$; and if $\varphi' = \langle\langle a\rangle\rangle \theta_1 \,\mathcal{U}\, \theta_2$, let $\varphi'' = \langle\langle 1\rangle\rangle p_{\theta_1} \,\mathcal{U}\, p_{\theta_2}$. Then, $q \models \varphi'$ in S iff $\pi_a(q) \models \varphi''$ in S'_a. In particular, given a winning strategy for player a from state q in S, we can construct a winning strategy for player 1 from state $\pi_a(q)$ in S_a, and vice versa. Since the size of S_a is exponential in the size of S, membership in EXPTIME follows from Theorem 5.2.

For the lower bound, we observe that the model-checking problem for the ATL formula $\langle\langle 1\rangle\rangle \Diamond\, p$ on a turn-based synchronous game structure with two players and incomplete information is identical to the outcome problem for 2-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 [KVW01], that is, to a search for winning strategies. While existing methods

	Closed Systems	Open Systems
ATL joint complexity	PTIME [CES86]	PTIME $O(m \cdot \ell)$
ATL structure complexity	NLOGSPACE [KVW00]	PTIME
Weakly-Fair ATL joint complexity	PTIME [CES86]	PTIME $O(m^2 \cdot w^3 \cdot \ell)$
Weakly-Fair ATL structure complexity	NLOGSPACE [KV95]	PTIME
Strongly-Fair ATL joint complexity	PTIME [CES86]	PSPACE $m^{O(w)} \cdot \ell$
Strongly-Fair ATL structure complexity	PTIME [KV98]	PSPACE
ATL* joint complexity	PSPACE [CES86]	2EXPTIME $m^{2^{O(\ell)}}$
ATL* structure complexity	NLOGSPACE [KVW00]	PTIME

Table 1: Model-checking complexity results.

for the verification of open systems do not avoid 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 so far, there is still a great deal of interesting reasoning about open systems that can be performed efficiently.

While closed systems are naturally modeled as Kripke structures (labeled transition systems), a general model for open systems, which can accommodate a wide variety of notions of composition, is the concurrent game structure. Closed systems correspond to the special case of a single player. In this case, game structures degenerate to Kripke structures, ATL degenerates to CTL, Fair ATL to Fair CTL, 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 structure, w is the number of fairness constraints, and ℓ is the length of the formula.

Acknowledgments. We thank Luca de Alfaro, Kousha Etessami, Salvatore La Torre, P. Madhusudan, Amir Pnueli, Moshe Vardi, Thomas Wilke, and Mihalis Yannakakis for helpful discussions. We also thank Freddy Mang for comments on a draft of this manuscript.

References

[AdAG+01] R. Alur, L. de Alfaro, R. Grosu, T.A. Henzinger, M. Kang, C.M. Kirsch, R. Majumdar, F.Y.C. Mang, and B.Y. Wang. JMOCHA: A model-checking tool that exploits design structure. In *Proceedings of the 23rd International Conference on Software Engineering*, pages 835–836. IEEE Computer Society Press, 2001.

[AH96] R. Alur and T.A. Henzinger. Reactive modules. *Formal Methods in System Design*, 15(1):7–48, 1999.

[AHM+98] R. Alur, T.A. Henzinger, F.Y.C. Mang, S. Qadeer, S.K. Rajamani, and S. Tasiran. MOCHA: Modularity in model checking. In *Proceedings of the 10th International Conference on Computer-Aided Verification*, volume 1427 of Lecture Notes in Computer Science, pages 521–525. Springer-Verlag, 1998.

[AL95] M. Abadi and L. Lamport. Conjoining specifications. *ACM Transactions on Programming Languages and Systems*, 17(3):507–534, 1995.

[ALM02] R. Alur, S. La Torre, and P. Madhusudan. *Playing Games with Boxes and Diamonds*. Technical report, University of Pennsylvania, 2002.

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

[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. *Information and Computation*, 98(2):142–170, 1992.

[Bee80] C. Beeri. On the membership problem for functional and multi-valued dependencies in relational databases. *ACM Transactions on on Database Systems*, 5:241–259, 1980.

[BL69] J.R. Büchi and L.H. Landweber. Solving sequential conditions by finite-state strategies. *Transactions of the AMS*, 138:295–311, 1969.

[Bry92] R.E. Bryant. Symbolic boolean manipulation with ordered binary-decision diagrams. *ACM Computing Surveys*, 24(3):293–318, 1992.

[CE81] E.M. Clarke and E.A. Emerson. Design and synthesis of synchronization skeletons using branching-time temporal logic. In *Proceedings of the International Workshop on Logic of Programs*, volume 131 of Lecture Notes in Computer Science, pages 52–71. Springer-Verlag, 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 ACM*, 28(1):114–133, 1981.

[CS91] R. Cleaveland and B. Steffen. A linear-time model-checking algorithm for the alternation-free modal μ-calculus. In *Proceedings of the Third International Conference on Computer Aided-Verification*, volume 575 of Lecture Notes in Computer Science, pages 48–58. Springer-Verlag, 1991.

[dAHM00] L. de Alfaro, T.A. Henzinger, and F.Y.C. Mang. The control of synchronous systems. In *Proceedings of the 11th International Conference on Concurrency Theory*, volume 1877 of Lecture Notes in Computer Science, pages 458–473. Springer-Verlag, 2000.

[dAHM01a] L. de Alfaro, T.A. Henzinger, and R. Majumdar. From verification to control: Dynamic programs for omega-regular objectives. In *Proceedings of the 16th Annual Symposium on Logic in Computer Science*, pages 279–290. IEEE Computer Society Press, 2001.

[dAHM01b] L. de Alfaro, T.A. Henzinger, and F.Y.C. Mang. The control of synchronous systems, part II. In *Proceedings of the 12th International Conference on Concurrency Theory*, volume 2154 of Lecture Notes in Computer Science, pages 566–580. Springer-Verlag, 2001.

[Dam94] M. Dam. CTL* and ECTL* as fragments of the modal μ-calculus. *Theoretical Computer Science*, 126:77–96, 1994.

[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 *Proceedings of the 29th international Symposium on Foundations of Computer Science*, pages 328–337. IEEE Computer Society Press, 1988.

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

[EL86] E.A. Emerson and C.-L. Lei. Efficient model checking in fragments of the propositional μ-calculus. In *Proceedings of the First International Symposium on Logic in Computer Science*, pages 267–278. IEEE Computer Society Press, 1986.

[Eme90] E.A. Emerson. Temporal and modal logic. In J. Van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B, pages 997–1072. Elsevier, 1990.

[ES84] E.A. Emerson and A. P. Sistla. Deciding branching-time logic. In *Proceedings of the 16th International Symposium on Theory of Computing*, pages 14–24. ACM Press, 1984.

[EWS01] K. Etessami, T. Wilke, and R. A. Schuller. Fair simulation relations, parity games, and state-space reduction for Büchi automata. In *Proceedings of the 28th International Colloquium on Automata, Languages, and Programming*, volume 2076 of Lecture Notes in Computer Science, pages 694–707. Springer-Verlag, 2001.

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

[GH82] Y. Gurevich and L. Harrington. Trees, automata, and games. STOC 1982: In *Proceedings of the 14th International Symposium on Theory of Computing*, pages 60–65, ACM Press, 1984.

[GSSL94] R. Gawlick, R. Segala, J. Sogaard-Andersen, and N.A. Lynch. Liveness in timed and untimed systems. In *Proceedings of the 21st International Colloquium on Automata, Languages, and Programming*, volume 820 of Lecture Notes in Computer Science, pages 166–177. Springer-Verlag, 1994.

[HF89] J.Y. Halpern and R. Fagin. Modeling knowledge and action in distributed systems. *Distributed Computing*, 3(4):159–179, 1989.

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

[Hol97] G.J. Holzmann. The model checker Spin. *IEEE Transactions on Software Engineering*, 23(5):279–295, 1997.

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

[Jur00] M. Jurdzinski. Small progress measures for solving parity games. In *Proceedings of the 17th International Symposium on Theoretical Aspects of Computer Science*, volume 1770 of Lecture Notes in Computer Science, pages 290–301. Springer-Verlag, 2000.

[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 *Proceedings of the Sixth International Conference on Concurrency Theory*, volume 962 of Lecture Notes in Computer Science, pages 408–422. Springer-Verlag, 1995.

[KV98] O. Kupferman and M.Y. Vardi. Verification of fair transition systems. *Chicago Journal of Theoretical Computer Science*, volume 1998(2), March 1998.

[KVW00] O. Kupferman, M.Y. Vardi, and P. Wolper. An automata-theoretic approach to branching-time model checking. *Journal of the ACM*, 47(2):312–360, 2000.

[KVW01] O. Kupferman, M.Y. Vardi, and P. Wolper. Module checking. *Information and Computation*, 164:322–344, 2001.

[LP85] O. Lichtenstein and A. Pnueli. Checking that finite-state concurrent programs satisfy their linear specification. In *Proceedings of the 12th International Symposium on Principles of Programming Languages*, pages 97–107. ACM Press, 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 *Proceedings of the 24th International Symposium on Foundations of Computer Science*, pages 195–200. IEEE Computer Society Press, 1983.

[Pnu77] A. Pnueli. The temporal logic of programs. In *Proceedings of the 18th International Symposium on Foundations of Computer Science*, pages 46–57. IEEE Computer Society Press, 1977.

[PR79] G.L. Peterson and J.H. Reif. Multiple-person alternation. In *Proceedings of the 20th International Symposium on Foundations of Computer Science*, pages 348–363. IEEE Computer Society Press, 1979.

[PR89a] A. Pnueli and R. Rosner. On the synthesis of a reactive module. In *Proceedings of the 16th International Symposium on Principles of Programming Languages*, pages 179–190. ACM Press, 1989.

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

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

[QS81] J.P. Queille and J. Sifakis. Specification and verification of concurrent systems in CESAR. In *Proceedings of the Fifth International Symposium on Programming*, volume 137 of Lecture Notes in Computer Science, pages 337–351. Springer-Verlag, 1981.

[Rab72] M.O. Rabin. *Automata on Infinite Objects and Church's Problem*, volume 13 of the Regional Conference Series in Mathematics. American Mathematical Society, 1972.

[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, 1992.

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

[Sha53] L.S. Shapley. Stochastic games. In *Proceedings of the National Academy of Science*, volume 39, 1953.

[Tho90] W. Thomas. Automata on infinite objects. In J. Van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B, pages 165–191. Elsevier, 1990.

[Tho95] W. Thomas. On the synthesis of strategies in infinite games. In *Proceedings of the 12th International Symposium on Theoretical Aspects of Computer Science*, volume 900 of Lecture Notes in Computer Science, pages 1–13. Springer-Verlag, 1995.

[Yan97] M. Yannakakis. *Synchronous Multi-Player Games with Incomplete Information are Undecidable*. Personal communication, 1997.