Proving ATL* Properties of Infinite-State Systems

Matteo Slanina, Henny B. Sipma, and Zohar Manna[⋆]

Stanford University
{matteo, sipma, manna}@cs.stanford.edu

Abstract. Alternating temporal logic (ATL*) was introduced to prove properties of multi-agent systems in which the agents have different objectives and may collaborate to achieve them. Examples include (distributed) controlled systems, security protocols, and contract-signing protocols. Proving ATL* properties over finite-state systems was shown decidable by Alur et al., and a model checker for the sublanguage ATL implemented in MOCHA.

In this paper we present a sound and complete proof system for proving ATL* properties over infinite-state systems. The proof system reduces proofs of ATL* properties over systems to first-order verification conditions in the underlying assertion language. The verification conditions make use of predicate transformers that depend on the system structure, so that proofs over systems with a simpler structure, e.g., turn-based systems, directly result in simpler verification conditions. We illustrate the use of the proof system on a small example.

1 Introduction

ATL* [1] is a logic used to specify properties of computing systems in which different agents have different goals. It allows reasoning about temporal properties that players can achieve in cooperation or competition with each other.

Alur et al. [1] showed that the verification of ATL* properties over *finite-state* systems is decidable, and they proposed several model-checking algorithms. Model checking of ATL (a restricted form of ATL*) properties over finite-state alternating systems was implemented in MOCHA [2]. MOCHA has since been applied to the analysis of a wide variety of systems, extending to such diverse realms as security and contract-signing protocols [3,4,5], or mechanism design [6]. Although in some of these analyses the restriction to finite-state systems was not a problem, in general this is not the case. For example, the analysis of the multi-party contract-signing protocol of [4,5], which is parameterized by the number of participating parties, was limited to small instances with three or four parties. Thus there is a need for methods for verifying ATL properties over infinite-state systems.

[⋆] This research was supported in part by NSF grants CCR-01-21403, CCR-02-20134, CCR-02-09237, CNS-0411363, and CCF-0430102, by ARO grant DAAD19-01-1-0723, and by NAVY/ONR contract N00014-03-1-0939.

In this paper we present a sound and complete proof system for proving ATL* properties over infinite-state alternating systems.

Proof systems for program logics come in two flavors. The first approach [7] reduces proofs of system properties to proofs of validities in the program logic. To prove a property φ over a system $\mathcal{S}$, the system is encoded in a formula $\Phi_{\mathcal{S}}$ in the program logic, and $\Phi_{\mathcal{S}} \rightarrow \varphi$ is proved valid. A complete proof system of this kind for propositional ATL was developed by Goranko *&* al. [8]. The second approach [9,10,11] reduces proofs of system properties to proofs of first-order validities by means of rules that act on the system representation directly. The proof system proposed in this paper follows the second approach.

Our proof system consists of proof rules that reduce the verification of an ATL* property over an alternating system to a set of first-order *verification conditions* in the underlying assertion language of the system. The verification conditions are expressed in terms of a *controllable predecessor* predicate transformer (cpre). The advantage of parameterizing the proof rules by cpre is that the rules are independent of the system structure, but the resulting verification conditions for different types of systems – e.g., turn-based systems – can be simplified by instantiating cpre with the version that exploits the more constrained system structure. The proof rules are constructive: a proof of the verification conditions can be used to construct controllers for the original property proved.

Our proof system incrementally converts temporal formulas into finite automata that are then composed with the system. This technique of lifting automata-theoretic results to proof systems was first proposed by Vardi and applied to LTL [12]. Later a similar approach was applied to CTL [13] and CTL* [11]. Our approach is most closely related to that in [11].

The rest of the paper is organized as follows. Section 2 presents our model of computation. Section 3 defines ATL*. Section 4 describes the proof system and Section 5 concludes. The models and proof rules are illustrated with a small example. Proofs of soundness and completeness can be found in [14].

2 Alternating Discrete Systems

As computational model we use *alternating discrete systems* (ADS), based on the fair discrete systems of Kesten and Pnueli [11]. An ADS is a general first-order representation of alternating structures, that generalizes turn-based, synchronous and asynchronous concurrency models of [1] and recursive programming languages. States and fairness conditions are represented as value assignments to a finite set of typed variables. To enable a first-order representation of the next-state relation, the player's available actions are represented by special *action variables.* The formal definitions are as follows.

An alternating discrete system (ADS) is a tuple

$$\mathcal{S} = \langle \Omega, V_S, V_\Omega, \xi, \chi, \mathcal{F} \rangle \ ,$$

where:

- Ω is a finite set of players.
- V_S is a finite set of typed system variables; a *state* is a typed value assignment to the variables in V_S; the set of all states is denoted by Σ.
- $V_\Omega = \langle V_a \mid a \in \Omega \rangle$ provides each player with a finite set of typed action variables. An a-action is a typed value assignment to the variables in V_a; the set of all a-actions is denoted by Γ_a. An A-action for a set of players $A \subseteq \Omega$ is a typed value assignment to the variables in $V_A = \bigcup_{a \in A} V_a$; the set of all A-actions is denoted by Γ_A. We write Γ for Γ_Ω.
- $\xi = \langle \xi_a \mid a \in \Omega \rangle$ associates to each player a a first-order formula over variables V_S and V_a that restricts the actions player a can choose at each state: at state V_S, player a can choose only actions such that $\xi_a(V_S, V_a)$ holds. The extension of ξ to a set of players $A \subseteq \Omega$ is defined as $\xi_A(V_S, V_A) \equiv \bigwedge_{a \in A} \xi_a(V_S, V_a)$.
- χ is a first-order formula over V_S, V_Ω, V'_S; χ represents the game matrix: $\chi(V_S, V_\Omega, V'_S)$ expresses that the system can move from state V_S to state V'_S when the players' choices are V_Ω.
- $\mathcal{F} : \Omega \to \mathbb{B}(\infty\,\mathrm{QF}(V_S))$ assigns to each player a fairness condition, represented as a Boolean formula over atoms of the form ∞p (read "infinitely many times p"), where p is an assertion (quantifier-free formula) over V_S. For example, $\infty(x = 2 \wedge y > x) \to \infty(y \geq z^2)$.

We assume that an ADS has no blocking states, i.e., states from which a player has no legal action, or from which there is no available successor state for certain choices of the players. Clearly, the property of being non-blocking can be expressed by a simple set of verification conditions, of the following forms:

$$\forall V_S\, \exists V_a.\, \xi_a(V_S, V_a) \text{ for all } a \in \Omega\text{, and}$$
$$\forall V_S\, \forall V_\Omega.\, \xi_\Omega(V_S, V_\Omega) \to \exists V'_S.\, \chi(V_S, V_\Omega, V'_S) \ .$$

The non-blocking assumption is not restrictive, since we can add a new state and make all previously blocking actions move to it; from there, all actions would then lead back to the same state. Thus we can assume without mentioning that these conditions hold in any system under consideration.

Some of the proof rules that we shall describe modify the underlying ADS. In those cases, if the original ADS is non-blocking, then the modified one is non-blocking too.

Example 1. As an illustration of the computational model of ADS, consider the model of PROCESSOR, a simple system consisting of a processor that must be scheduled to execute multiple processes, shown in Fig. 1. In the model, processes are stored in a queue, represented by the system variable qu, and the processor is either active or not active, represented by the boolean system variable pa. When the processor is inactive, a new process, represented by the environment action variable np, can enter and is inserted at the end of the queue. The environment may choose not to enter a new process by setting np to $\perp$. When the processor becomes active, the process at the head of the queue is therefrom

$$
\begin{array}{ll}
\Omega: & \{Env, Sched\} \\
V_S: & \{qu : \text{list of process}, pa : boolean, xp : \text{process}_\bot\} \\
V_{Env}: & \{np : \text{process}_\bot, te : \{no, yes, cont\}\} \\
V_{Sched}: & \{pos : \mathbb{N}\} \\
\xi_{Env}: & \mathrm{T} \\
\xi_{Sched}: & pa \rightarrow (0 \leq pos \leq |qu|) \\
\chi: & (\neg pa \wedge np = \bot \wedge pres\{pa, xp, qu\}) \quad \vee \\
 & (\neg pa \wedge np \neq \bot \wedge qu' = append(qu, np) \wedge pres\{pa, xp\}) \quad \vee \\
 & (\neg pa \wedge np = \bot \wedge qu \neq empty \wedge pa' \wedge qu = cons(xp', qu')) \quad \vee \\
 & (pa \wedge te = no \wedge pres\{pa, xp\}) \quad \vee \\
 & (pa \wedge te = yes \wedge \neg pa' \wedge xp' = \bot \wedge pres\{qu\}) \quad \vee \\
 & (pa \wedge te = cont \wedge \neg pa' \wedge xp' = \bot \wedge qu' = insert(qu, pos, xp)) \\
\mathcal{F}_{Env}: & \infty\neg pa \wedge \infty pa \\
\mathcal{F}_{Sched}: & \mathrm{T}
\end{array}
$$

Fig. 1. ADS for PROCESSOR: *pres*{...} means the values are preserved by the transition; *append* adds an element to the end of a list, *insert* adds an element at a certain position, &c

removed and becomes the *executing process*, represented by the system variable *xp*. When the process releases the processor, it may or may not need to continue later, represented by the environment action variable *te*. If it needs to continue, the scheduler reinserts it in the queue at the position determined by its action variable *pos*. It is assumed that all executing processes eventually release the processor and that there is an unlimited supply of processes to be executed, represented by the environment fairness condition. An informal representation of the model is shown in Fig. 2.

$$\begin{tikzcd}
pa \arrow[loop left, "no"] \arrow[r, "yes"] \arrow[r, bend left=60, "cont"] & \neg pa \arrow[l, bend left=60] \arrow[loop above, "np=\bot"] \arrow[loop below, "np\neq\bot"]
\end{tikzcd}$$

Fig. 2. Informal representation of ADS for PROCESSOR

Given an ADS $\mathcal{S}$, a run consists of the following game played ad infinitum: At each state $s \in \Sigma$ assigning values to variables V_S, every player $a \in \Omega$, independently of the others, picks an action by choosing values for the local variables V_a so that $\xi_a(V_S, V_a)$ holds. Then, the next state is nondeterministically chosen among the assignments to V_S' such that $\chi(V_S, V_\Omega, V_S')$ holds. Notice that our assumption of non-blocking guarantees that such an assignment always exists. The formal definitions are as follows.

A sequence $\pi \in \Sigma^\omega$ is a *run* of $\mathcal{S}$ from $s \in \Sigma$, with *choices* $\rho \in \Gamma^\omega$, if $\pi[0] = s$ and

$$\xi_a(\pi[n], \rho[n]_a) \qquad\qquad \chi(\pi[n], \rho[n], \pi[n+1])$$

for all $n < \omega$ and $a \in \Omega$. A run from $X \subseteq \Sigma$ is a run from any state $s \in X$. We omit the initial state if it is irrelevant or clear from the context. A run π is *fair* to player a, written $\pi \models \mathcal{F}_a$, if $\mathcal{F}_a$ evaluates to true under the interpretation of atoms ∞p as "p holds at $\pi[n]$ for infinitely many n".

A player $a \in \Omega$ can make its choices ρ in accordance with a *strategy*, a function

$$f_a : \Sigma^+ \to \Gamma_a$$

such that $\xi_a(s, f_a(ws))$ holds for all $w \in \Sigma^*$ and $s \in \Sigma$. A run π is *compatible* with strategy f_a for player a if its choices ρ satisfy

$$\rho[n]_a = f_a(\pi[0 \ldots n])$$

for all $n < \omega$. A run is compatible with strategies f_A (denoting the sequence $\langle f_a \mid a \in A \rangle$), for $A \subseteq \Omega$, if it is compatible with f_a for all $a \in A$. The set of all runs compatible with f_A starting at a certain state $s \in \Sigma$ is called the set of *outcomes* of f_A from s and denoted

$$out_{\mathcal{S}}(s, f_A) \ ,$$

or $out(s, f_A)$ when $\mathcal{S}$ is clear from the context.

The fundamental operator to describe properties of discrete structures is the *controllable predecessors* operator cpre_A. Given a set of states $X \subseteq \Sigma$ and a set of players $A \subseteq \Omega$, $\mathrm{cpre}_A(X)$ denotes the set of states from which the players in A have a collaborative action with which they can ensure that the game will be in X at the next state. Formally,

$$\begin{aligned}\mathrm{cpre}_A(\varphi)(V_S) \equiv{} & \exists V_A .\, \xi_A(V_S, V_A) \land \\ & \forall V_{\Omega \setminus A} .\, \xi_{\Omega \setminus A} \to \forall V'_S .\, \chi(V_S, V_\Omega, V'_S) \to \varphi(V'_S) \ . \end{aligned} \tag{1}$$

Dual to cpre_A is the *uncontrollable predecessors* operator upre_A, defined as

$$\mathrm{upre}_A(X) = \Sigma \setminus \mathrm{cpre}_A(\Sigma \setminus X) \ ,$$

or, explicitly, as

$$\begin{aligned}\mathrm{upre}_A(\varphi)(V_S) \equiv{} & \forall V_A .\, \xi_A(V_S, V_A) \to \\ & \exists V_{\Omega \setminus A} .\, \xi_{\Omega \setminus A} \land \exists V'_S .\, \chi(V_S, V_\Omega, V'_S) \land \varphi(V'_S) \ . \end{aligned} \tag{2}$$

For classes of ADS with special properties, the cpre transformers have simpler forms [1,15]. To take advantage of these simpler forms, we express our verification conditions in terms of these transformers as much as possible.

Example 2. As an illustration of how the cpre operator is affected by the system structure, consider the following asynchronous game structure $\mathcal{S}$, consisting of two agents a and b. The state space of $\mathcal{S}$ is partitioned so that, from any given state, either a or b has complete control of the next state, represented by the formulas $turn_a(V_S)$ and $turn_b(V_S) \equiv \neg turn_a(V_S)$. Furthermore, the next-state relation is represented by the formulas $\chi_a(V_S, V_S')$ and $\chi_b(V_S, V_S')$, meaning when it is a's turn, agent a can choose any state in V_S' such that $\xi_a(V_S, V_S')$ holds, and similarly for agent b. For this game structure, $\mathrm{cpre}_a(\varphi)$ can be simplified to

$$\mathrm{cpre}_a(\varphi)(V_S) \equiv \begin{pmatrix} turn_a(V_S) \rightarrow \exists V_S'.\, \chi_a(V_S, V_S') \wedge \varphi(V_S) \\ \wedge \\ \neg turn_a(V_S) \rightarrow \forall V_S'.\, \chi_b(V_S, V_S') \rightarrow \varphi(V_S') \end{pmatrix}$$

As we shall see, cpre always appears in verification conditions in the consequent of a universally quantified implication, and thus the corresponding verification conditions for this game structure can always be split into two simpler ones.

3 The Logic ATL*

ATL* (Alternating Temporal Logic) was proposed by Alur *&* al. to allow selective quantification over runs that are the possible outcomes of games [1]. For convenience we use a version of ATL* with a few more connectives. (The expressive power is not affected.)

3.1 Syntax

ATL* formulas come in two types, state formulas and path formulas, defined by mutual induction.

A *(state) formula* is one of:

– an assertion (first-order formula) in the underlying state language,
– a Boolean combination of state formulas,
– $\langle\!\langle A\rangle\!\rangle\varphi$, $[\![A]\!]\varphi$, $\langle\!\langle A\rangle\!\rangle_f\varphi$, or $[\![A]\!]_f\varphi$, for A a set of players and φ a path formula.

A *path formula* is one of:

– a state formula,
– a Boolean combination of path formulas, or
– an LTL temporal operator applied to path formulas.

For LTL operators we use the notation of [9, 10]: $\Box$ for *always in the future*, $\Diamond$ for *eventually in the future*, &c.

The operators $\langle\!\langle A\rangle\!\rangle, [\![A]\!], \langle\!\langle A\rangle\!\rangle_f, [\![A]\!]_f$ are called *alternating quantifiers.* The most basic one is $\langle\!\langle A\rangle\!\rangle$, stating that A have a strategy to make a path formula true in all runs starting in the current state. The dual operator $[\![A]\!]$ is defined as $[\![A]\!]\varphi \equiv \neg\langle\!\langle A\rangle\!\rangle\neg\varphi$: we usually say that A *cannot avoid* φ from happening. The *fair* alternating quantifiers $\langle\!\langle A\rangle\!\rangle_f$ and $[\![A]\!]_f$ are similar, but interpreted over all fair runs instead of all runs.

3.2 Semantics

Let $\mathcal{S}$ be an ADS. We define truth relations

$$\mathcal{S}, s \models \varphi \qquad\qquad \mathcal{S}, \pi \models \psi$$

for a state formula φ at a state s and for a path formula ψ over a path π, by mutual induction on the structure of the formula. Recall that $out_{\mathcal{S}}(s, f_A)$ denotes the set of runs of $\mathcal{S}$ starting at s and compatible with strategies f_A.

- $\mathcal{S}, s \models p$, for p an assertion, if $s \models p$ in the assertion language;
- Boolean operators distribute over $\models$ in the natural way, both for state and path formulas;
- $\mathcal{S}, s \models \langle\!\langle A \rangle\!\rangle \psi$ if there exist strategies f_A such that, for all $\pi \in out(s, f_A)$, we have $\mathcal{S}, \pi \models \psi$;
- $\mathcal{S}, s \models [\![A]\!] \psi$ if, for all strategies f_A, there exists an outcome $\pi \in out(s, f_A)$ such that $\mathcal{S}, \pi \models \psi$ holds;
- $\mathcal{S}, s \models \langle\!\langle A \rangle\!\rangle_f \psi$ if there exist strategies f_A such that, for all outcomes $\pi \in out(s, f_A)$ such that $\pi \models \mathcal{F}_{\Omega \setminus A}$, we have also $\pi \models \mathcal{F}_A$ and $\mathcal{S}, \pi \models \psi$;
- $\mathcal{S}, s \models [\![A]\!]_f \psi$ if, for all strategies f_A, there is at least an outcome $\pi \in out(s, f_A)$ such that $\pi \models \mathcal{F}_{\Omega \setminus A}$ and, if $\pi \models \mathcal{F}_A$, then also $\mathcal{S}, \pi \models \psi$;
- LTL operators are evaluated over path formulas in the usual way.

When $\mathcal{S}$ is clear from the context, we simply write $s \models \varphi$ and $\pi \models \psi$. We say that

$$\mathcal{S} \models p \Rightarrow \varphi$$

when $\mathcal{S}, s \models \varphi$ for all states $s \in \Sigma$ satisfying p.

Example 3. Reconsider the system modeled in Example 1. We want to prove that the scheduler has a strategy that allows it to be fair: from a state where a process x is in the queue, the scheduler can play its choices in such a way that x will eventually be executed. We model this requirement with the ATL* formula

$$x \in qu \Rightarrow \langle\!\langle Sched \rangle\!\rangle_f \diamondsuit\, exec(x) \ ,$$

where x is a free variable and $exec(x)$ is an abbreviation for $pa \wedge xp = x$.

4 Proof System

4.1 Overview

Our proof system operates on statements of the form

$$\mathcal{S} \models p \Rightarrow (\!(A)\!)\varphi \ ,$$

where $\mathcal{S}$ is an ADS, p is an assertion, $(\!(A)\!)$ is an alternating quantifier, and φ is a path formula in positive normal form. (Every ATL* formula can be put in positive normal form, where all negations have been pushed to the assertion level, in the same way used for propositional logic and LTL, and rewriting $\neg\langle\!\langle A \rangle\!\rangle\varphi$ to $[\![A]\!]\neg\varphi$ &c.) When clear from the context, we omit $\mathcal{S}$ and simply write $p \Rightarrow (\!(A)\!)\varphi$. The rules of the proof system can be classified into four groups:

1. A basic state rule, which reduces all statements to the form $p \Rightarrow \langle\!\langle A \rangle\!\rangle \varphi$, where φ is an LTL formula.
2. A basic path rule, which reduces φ (an LTL formula) to an assertion while extending the system $\mathcal{S}$ by synchronously composing it with an automaton for φ.
3. A history rule, which augments the system with extra history variables such that, in the new system, A can win φ with memoryless strategies from its winning set.
4. Assertion rules, which reduce the validity of statements about winnability with memoryless strategies to assertional verification conditions.

The application of these rules results in a set of verification conditions to be proved in the underlying theory of the system plus the cpre predicate transformers. The advantage of parameterizing the underlying language to the cpre's is that, as illustrated in Example 2, in most practical cases the alternating system has specific properties that can be exploited by defining a simpler version of cpre than the generic form for ADS's shown in (1).

Completeness of the proof system is relative to validities in the first-order logic, with fixpoints and cpre, of the underlying theory – the same as required for relative completeness for LTL, or program termination, proof systems [9]. Proofs of soundness and completeness of all proof rules can be found in [14].

4.2 Basic State Rule

For ψ a state formula appearing with positive polarity in $\varphi(\psi)$,

$$\begin{array}{l}\text{BASIC-STATE:}\\ p \Rightarrow \varphi(q)\\ q \Rightarrow \psi\\ \hline p \Rightarrow \varphi(\psi)\end{array}$$

This rule says that, in order to prove $p \Rightarrow \varphi(\psi)$, where ψ is a state formula appearing with positive polarity in φ, we guess an assertion q underapproximating the set of states on which ψ holds and substitute q for ψ in φ. The two premises require us to prove that q is indeed an underapproximation ($q \Rightarrow \psi$) and that the formula after the substitution holds in the system ($p \Rightarrow \varphi(q)$). Notice that there is an implicit "$\mathcal{S} \models$ " at the left of every line in the rule.

Example 4. Reconsider the system from Example 1. In trying to prove

$$\neg pa \Rightarrow \langle\!\langle Env \rangle\!\rangle \Diamond \langle\!\langle Sched \rangle\!\rangle_f \Diamond\, exec(x)$$

over this system, we can apply rule BASIC-STATE with $\psi \equiv \langle\!\langle Sched \rangle\!\rangle_f \Diamond\, exec(x)$ and $q \equiv x \in qu$ to obtain the subgoals $\neg pa \Rightarrow \langle\!\langle Env \rangle\!\rangle \Diamond (x \in qu)$ and $x \in qu \Rightarrow \langle\!\langle Sched \rangle\!\rangle_f \Diamond\, exec(x)$.

4.3 Basic Path Rule

Augmentation. The Basic Path Rule applies to statements of the form $\mathcal{S} \models p \Rightarrow \langle\!\langle A \rangle\!\rangle \varphi$ where φ is an LTL formula. The first step in the application of this rule is the synchronous composition of an automaton for φ with $\mathcal{S}$ [12,11].

Let φ be a (quantifier-free) LTL formula over variables V_S. Let

$$\mathcal{A}_\varphi = \langle Q, q_0, \delta, \mathcal{F} \rangle$$

be a deterministic Muller automaton accepting the language of φ, where:

- Q is a finite set of states;
- $q_0 \in Q$ is the initial state;
- $\delta : Q \times \Sigma \to Q$ is a full deterministic transition function;
- $\mathcal{F} \in 2^{2^Q}$ is the Muller acceptance condition.

The definition suggests that the alphabet of $\mathcal{A}_\varphi$ is Σ, the – generally infinite – set of states of the underlying ADS. Only a finite quotient of Σ, however, is necessary, determined by the values assumed by the states on the atoms of φ. We use Muller acceptance condition for simplicity of notation. In practice, Streett is of course preferable (or, if possible, an even simpler acceptance condition).

The role of $\mathcal{A}_\varphi$ is to act as a temporal tester [11], that is, to observe the evolution of φ on the ADS. To achieve this, we construct a synchronous composition of $\mathcal{A}_\varphi$ and the ADS $\mathcal{S}$ and introduce a new player a_φ with the fairness conditions of $\mathcal{A}_\varphi$. The requirement that $\mathcal{A}_\varphi$ be deterministic ensures that no player gains power by the composition with $\mathcal{A}_\varphi$. In particular, the new player a_φ has only one choice of action at all times. The formal definition is as follows.

Let $\mathcal{S} = \langle \Omega, V_S, V_\Omega, \xi, \chi, \mathcal{F} \rangle$ be an ADS, and $\mathcal{A} = \langle Q, q_0, \delta, \mathcal{F}_\mathcal{A} \rangle$ a deterministic Muller (or Büchi, or Streett ...) automaton on alphabet Σ, as defined above. The *synchronous composition* of $\mathcal{S}$ and $\mathcal{A}$, denoted $\mathcal{S} \mathbin{|||} \mathcal{A}$, is the ADS

$$\hat{\mathcal{S}} = \langle \hat{\Omega}, \hat{V}_S, \hat{V}_\Omega, \hat{\xi}, \hat{\chi}, \hat{\mathcal{F}} \rangle \ ,$$

where:

- $\hat{\Omega} = \Omega \cup \{a_\mathcal{A}\}$, where $a_\mathcal{A}$ is a new player;
- $\hat{V}_S = V_S \cup \{q\}$, where q is a new variable of type Q;
- $\hat{V}_a = V_a$ if $a \in \Omega$;
  $\hat{V}_{a_\mathcal{A}} = \emptyset$;
- $\hat{\xi}_a(\hat{V}_S, V_a) \equiv \xi_a(V_S, V_a)$ if $a \in \Omega$;
  $\hat{\xi}_{a_\mathcal{A}}(\hat{V}_S, \emptyset) \equiv \mathrm{T}$;
- $\hat{\chi}(\hat{V}_S, \hat{V}_\Omega, \hat{V}'_S) \equiv \chi(V_S, V_\Omega, V'_S) \wedge q' = \delta(q, V_S)$;
- $\hat{\mathcal{F}} \equiv \mathcal{F} \wedge \mathcal{F}_A$, where $\mathcal{F}_\mathcal{A}$ is an expression of $\mathcal{A}$'s acceptance condition.

The Basic-Path Rule. For an LTL formula φ,

BASIC-PATH:

$$\frac{\mathcal{S} \mathbin{|||} \mathcal{A}_\varphi \models p \wedge q = q_0 \Rightarrow \langle\!\langle A, a_\varphi \rangle\!\rangle_f \mathrm{T}}{\mathcal{S} \models \qquad p \Rightarrow \langle\!\langle A \rangle\!\rangle_f \varphi}$$

where $\mathcal{A}_\varphi$ is a deterministic automaton on infinite words accepting φ, $|||$ is synchronous composition, $(\!(A)\!)_f$ is either $\langle\!\langle A\rangle\!\rangle_f$ or $[\![A]\!]_f$, and a_φ stands for $a_{\mathcal{A}_\varphi}$. Notice that we require the alternating quantifiers to be fair. If this is not the case, we simply remove all fairness conditions from the system before applying the rule.

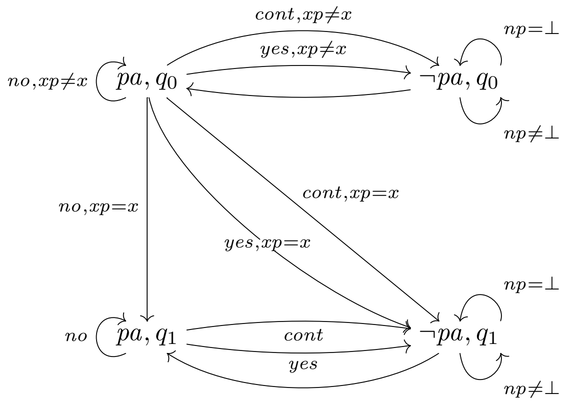
Example 5. Returning to our example, we apply BASIC-PATH using the following deterministic automaton $\mathcal{A}$ for $\Diamond\, exec(x)$:

$$\begin{tikzcd} q_0 \arrow[loop left, "\neg exec(x)"] \arrow[r, "exec(x)"] & q_1 \arrow[loop right, "\top"] \end{tikzcd}$$

with initial state q_0 and fairness condition ∞q_1. Then $\mathcal{S} \mathbin{|||} \mathcal{A}$ becomes:

$$\hat{\Omega} : \{Env, Sched, a\} \qquad \hat{V}_S : V_S \cup \{q : \{q_0, q_1\}\}$$
$$\hat{V}_{Env} : V_{Env} \qquad \hat{V}_{Sched} : V_{Sched} \qquad \hat{V}_a : \varnothing$$
$$\hat{\xi}_{Env} : \xi_{Env} \qquad \hat{\xi}_{Sched} : \xi_{Sched} \qquad \hat{\xi}_a : \top$$
$$\hat{\chi} : \chi \wedge \big((q = q_0 \wedge \neg exec(x) \rightarrow q' = q_0) \wedge (q = q_1 \vee exec(x) \rightarrow q' = q_1)\big)$$
$$\hat{\mathcal{F}}_{Env} : \infty pa \wedge \infty\neg pa \qquad \hat{\mathcal{F}}_{Sched} : \top \qquad \hat{\mathcal{F}}_a : \infty(q = q_1)$$

The following picture summarizes the game matrix $\hat{\chi}$ for the augmented ADS:



The property to prove on this system is now

$$x \in qu \wedge q = q_0 \Rightarrow \langle\!\langle Sched, a\rangle\!\rangle_f \top\ . \tag{3}$$

4.4 History Rule

The purpose of the history rule is to allow for memoryless strategies in all the games of interest. Consider a property of the form $\mathcal{S} \models p \Rightarrow \langle\!\langle A\rangle\!\rangle\varphi$, where φ is an LTL property. It is known (see, for example, [16]) that if we partition the states of $\mathcal{S}$ into two sets, W_1 and W_2, such that players A have a winning strategy (can ensure φ) from every state in W_1, but not from any state in W_2, then players A have a *finite memory* winning strategy from every state in W_1.

The application of the assertion rules requires that A have a *memoryless* strategy. To achieve this, we add to the structure some new variables, called *history variables*, and a new player h (for *history*). We let h play in coalition with A and give it the task of maintaining the history variables: at every step, h will make a deterministic choice for the history variables, and the game matrix χ will simply copy these choices into the next stage of the game.

History Augmentations. Let h be a new player and V_h a new set of *history* variables. We define the history augmentation of $\mathcal{S}$ with history V_h, denoted $\mathcal{S}[h, V_h]$, to be the ADS

$$\hat{\mathcal{S}} = \langle \hat{\Omega}, \hat{V}_S, \hat{V}_\Omega, \hat{\xi}, \hat{\chi}, \hat{\mathcal{F}} \rangle \ ,$$

where:

- $\hat{\Omega} = \Omega \cup \{h\}$;
- $\hat{V}_S = V_S \cup V_h^*$, where V_h^* is a copy of V_h;
- $\hat{V}_a = V_a$ if $a \in \Omega$;
  $\hat{V}_h = V_h$;
- $\hat{\xi}_a(\hat{V}_S, V_a) \equiv \xi_a(V_S, V_a)$ if $a \in \Omega$;
  $\hat{\xi}_h(\hat{V}_S, V_h) \equiv \mathrm{T}$;
- $\hat{\chi}(\hat{V}_S, \hat{V}_\Omega, \hat{V}_S') \equiv \chi(V_S, V_\Omega, V_S') \wedge V_h^{*\prime} = V_h$;
- $\hat{\mathcal{F}} \equiv \mathcal{F}$.

The History Rule. For an LTL formula φ,

$$\text{HISTORY:}\quad \frac{\mathcal{S}[h, V_h] \models p \Rightarrow (\!(A, h)\!)\varphi}{\mathcal{S} \models p \Rightarrow (\!(A)\!)\varphi}$$

where $\mathcal{S}[h, V_h]$ is a history augmentation of $\mathcal{S}$.

4.5 Assertion Rules

After applying the previous rules as much as possible, we are left with a set of statements of the form

$$\mathcal{S} \models p \Rightarrow (\!(A)\!)q \ ,$$

where $\mathcal{S}$ is an ADS, p and q are assertions, and $(\!(A)\!)$ is one of the four alternating quantifiers $\langle\!\langle A \rangle\!\rangle$, $[\![A]\!]$, $\langle\!\langle A \rangle\!\rangle_f$, $[\![A]\!]_f$. In this section we show how to reduce each of these to first-order validities. First, we transform the fair quantifiers into unfair ones by making the fairness conditions explicit. This reintroduces temporal operators in the scope of the alternating quantifier. The resulting temporal formula, however, is of a special form that is dealt with directly by the assertion rule.

Making Fairness Conditions Explicit. Recall that every fairness condition $\mathcal{F}_a$ is a Boolean combination of atoms of the form ∞p, where p is an assertion. We write ∞p instead of $\Box\Diamond p$ to make it clear that we are now dealing with a special case and not with arbitrary LTL formulas.

From the definitions of the semantics of the alternating quantifiers it follows that $p \Rightarrow \langle\!\langle A \rangle\!\rangle_f q$ can be rewritten as the conjunction of the two statements

$$p \wedge q \Rightarrow \langle\!\langle A \rangle\!\rangle (\mathcal{F}_{\Omega \setminus A} \rightarrow \mathcal{F}_A) \qquad\qquad p \wedge \neg q \Rightarrow \langle\!\langle A \rangle\!\rangle \neg \mathcal{F}_{\Omega \setminus A}$$

and $p \Rightarrow [\![A]\!]_f q$ as the conjunction of

$$p \wedge q \Rightarrow [\![A]\!] \mathcal{F}_{\Omega \setminus A} \qquad\qquad p \wedge \neg q \Rightarrow [\![A]\!] (\mathcal{F}_{\Omega \setminus A} \wedge \neg \mathcal{F}_A)$$

These statements are all of the form

$$p \Rightarrow (\!(A)\!) \mathcal{F} \enspace ,$$

where $(\!(A)\!)$ is either $\langle\!\langle A \rangle\!\rangle$ or $[\![A]\!]$ and $\mathcal{F}$ is a Boolean combination of ∞ atoms, and can be rewritten as

$$\mathcal{S} \models p \Rightarrow (\!(A)\!) \bigwedge_i \big(\infty J^i_1 \wedge \ldots \wedge \infty J^i_k \rightarrow \infty q_i \big) \enspace .$$

(Technically, the number k of antecedents is different for every i. Without loss of generality, we drop this distinction to lighten the notation.) Below we present proof rules to reduce these particular forms to assertional verification conditions.

The Positive Assertion Rule. This rule applies to formulas of the form

$$p \Rightarrow \langle\!\langle A \rangle\!\rangle \bigwedge_i \big(\infty J^i_1 \wedge \ldots \wedge \infty J^i_k \rightarrow \infty q_i \big) \enspace . \tag{4}$$

To apply the rule, we guess intermediate assertions r^i and r^j_i (for $i \in \{1, \ldots, n\}$ and $j \in \{1, \ldots, k\}$) and ranking functions δ^j_i on a well-founded domain $\langle A, \prec \rangle$.

POS-ASSERTION:

$$p \Rightarrow \textstyle\bigwedge_{i=1}^n r^i$$

For every $i \in \{1, \ldots, n\}$:

$$r^i \Rightarrow \textstyle\bigvee_{j=1}^k r^i_j$$

For every choice of $\{j_i \mid i \in \{1, \ldots, n\}\}$:

$$\frac{\displaystyle\bigwedge_{i=1}^n (r^i_{j_i} \wedge \delta^i_{j_i} = a_i) \Rightarrow \text{cpre}_A \bigwedge_{i=1}^n \begin{bmatrix} (r^i \wedge q_i) \\ \vee \\ \bigvee_{l=1}^k (r^i_l \wedge \delta^i_l \prec a_i) \\ \vee \\ (r^i_{j_i} \wedge \delta^i_{j_i} \preceq a_i \wedge \neg J^i_{j_i}) \end{bmatrix}}{p \Rightarrow \langle\!\langle A \rangle\!\rangle \bigwedge_{i=1}^n \big(\infty J^i_1 \wedge \ldots \wedge \infty J^i_k \rightarrow \infty q_i \big)}$$

The intuition behind this rule is similar to that for the analogous rules for LTL [9]. The ranking functions enforce progress towards realizing the q_i, and the

r_i^j denote regions inside which the ranking functions are constant. The verification conditions assure that, assuming fairness of the adversaries, the players A can eventually force the game out of these regions, and thus decrease the ranking.

This proof rule is sound and relatively complete to prove properties of the form (4). Relative completeness means that, if (4) holds of a system, then there exist assertions r^i, r_j^i, δ_j^i that are expressible in the language and satisfy the premises.

Example 6. Returning to our running example, making the fairness conditions of (3) explicit results in

$$x \in qu \wedge q_0 \Rightarrow \langle\langle Sched, a\rangle\rangle(\hat{\mathcal{F}}_{Env} \rightarrow \hat{\mathcal{F}}_{Sched} \wedge \hat{\mathcal{F}}_a) \ ,$$

or, equivalently

$$x \in qu \wedge q_0 \Rightarrow \langle\langle Sched, a\rangle\rangle(\infty\neg pa \wedge \infty pa \rightarrow \infty q_1) \ ,$$

where we abbreviate $q = q_0$ with q_0 and $q = q_1$ with q_1.

To apply rule POS-ASSERTION, with $n = 1$ and $k = 2$ (since $n = 1$, we drop the superscripts), we need to find assertions r, r_1, r_2 and ranking functions δ_1, δ_2 (index 1 corresponds to the $\infty\neg pa$ requirement, index 2 to ∞pa) and then prove the following verification conditions:

$$\begin{aligned}
x \in qu \wedge q_0 &\rightarrow r \\
r &\rightarrow r_1 \vee r_2 \\
r_1 \wedge \delta_1 = d &\rightarrow \mathrm{cpre}_{Sched,a}\begin{bmatrix}(r \wedge q_1) \vee (r_1 \wedge \delta_1 \prec d) \\ \vee (r_2 \wedge \delta_2 \prec d) \vee (r_1 \wedge \delta_1 \preceq d \wedge pa)\end{bmatrix} \\
r_2 \wedge \delta_2 = d &\rightarrow \mathrm{cpre}_{Sched,a}\begin{bmatrix}(r \wedge q_1) \vee (r_1 \wedge \delta_1 \prec d) \\ \vee (r_2 \wedge \delta_2 \prec d) \vee (r_2 \wedge \delta_2 \preceq d \wedge \neg pa)\end{bmatrix}
\end{aligned}$$

We choose the following:

$$r : ((x \in qu \vee x = xp) \wedge q_0) \vee q_1$$

$$\begin{array}{ll}
r_1 : r \wedge pa & r_2 : r \wedge \neg pa \\
\delta_1 : \langle g, depth(x, qu), 1\rangle & \delta_2 : \langle g, depth(x, qu), 0\rangle
\end{array}$$

where

$$g = \begin{cases} 0 & \text{if } x = xp \\ 1 & \text{otherwise} \end{cases} ,$$

and $depth(x, qu)$ is the distance from the head of the first occurrence of x in qu, or 0 if x is not in qu. The domain of the ranking functions is $\{0, 1\} \times \mathbb{N} \times \{0, 1\}$ with the standard lexicographic order. The main part of the ranking functions is the *depth* term – its value decreases as we remove items from the head of qu, provided the scheduler reinserts processes far enough back in the queue. The other two components are adjustments needed for the cases of going from active to not active (third component) and for the boundary case of x having left the queue and being executed (first component).

The Negative Assertion Rule. The negative assertion rule is used for formulas of the form

$$p \Rightarrow [\![A]\!] \bigwedge_i \left(\infty J_1^i \wedge \ldots \wedge \infty J_k^i \rightarrow \infty q_i \right) .$$

The rule and its properties are identical to the positive version, except that it uses the predicate transformer upre instead of cpre.

5 Conclusions and Future Work

We have presented a sound and complete proof system for proving ATL* properties over infinite-state alternating structures. The proof system can be used as a basis for the construction of special-purpose proof systems for alternating systems with a specific structure, e.g., turn-based or lock-step [1], or for proving specific properties, e.g., invariants or reachability. We expect that our proof system will be particularly beneficial in the verification of security and contract-signing protocols, which often have a very specific structure that can be exploited to simplify the cpre predicate transformers.

Our proof system may also contribute to the construction of abstraction-based verification methods. The foundations for proving ATL* properties over infinite-state alternating systems using abstraction were laid in [17]. However, methods for finding a suitable abstraction function and proving its correctness, which for infinite-state systems must necessarily rely on deduction, still require investigation. We expect that the proof rules presented here will provide valuable insights in proving that a proposed abstraction is sound, since the corresponding verification conditions are of the same form as those generated by our proof system.

Other areas for further investigation include the development of approximations and heuristics for special cases, e.g., automatic generation of ranking functions; the construction of efficient decision procedures, tailored to the verification conditions produced, e.g., for simple $\forall\exists$ formulas over program types; and the representation of proofs by diagrams, similar to verification diagrams [18], which allow to reduce the complexity of the premises of the assertion rules, by making use of user-provided structure.

Acknowledgments

We thank Aaron Bradley and the anonymous referees for their comments on a previous draft of this paper.

References

1. Alur, R., Henzinger, T.A., Kupferman, O.: Alternating-time temporal logic. Journal of the ACM **49**(5) (2002) 672–713
2. Alur, R., Henzinger, T., Mang, F., Qadeer, S., Rajamani, S., Tasiran, S.: MOCHA: Modularity in model checking. In: Proc. 10th Intl. Conference on Computer Aided Verification. Volume 1427 of LNCS., Springer (1998) 516–520

3. Kremer, S., Raskin, J.F.: A game-based verification of non-repudiation and fair exchange protocols. Journal of Computer Security **11**(3) (2003) 399–429
4. Kremer, S., Raskin, J.F.: Game analysis of abuse-free contract signing. In: Computer Security Foundations Workshop (CSFW), IEEE Computer Society (2002)
5. Chadha, R., Kremer, S., Scedrov, A.: Formal analysis of multi-party contract signing. Journal of Automated Reasoning (2006) To appear.
6. Pauly, M., Wooldridge, M.: Logic for mechanism design—A manifesto. In: Proceedings of the 2003 Workshop on Game Theory and Decision Theory in Agent-Based Systems (GTDT-2003), Melbourne, Australia (2003)
7. Lamport, L.: Specifying Systems. Addison-Wesley (2002)
8. Goranko, V., van Drimmelen, G.: Complete axiomatization and decidability of alternating-time temporal logic. Theoretical Computer Science **353**(1–3) (2006) 93–117
9. Manna, Z., Pnueli, A.: Completing the temporal picture. In Ausiello, G., Dezani-Ciancaglini, M., Ronchi Della Rocca, S., eds.: 16th International Colloquium on Automata, Languages, and Programming. Volume 372 of LNCS., Springer (1989) 534–558
10. Manna, Z., Pnueli, A.: Temporal Verification of Reactive Systems: Safety. Springer (1995)
11. Kesten, Y., Pnueli, A.: A compositional approach to CTL* verification. Theoretical Computer Science **331** (2005) 397–428
12. Vardi, M.Y.: Verification of concurrent programs: The automata-theoretic framework. Journal of Pure and Applied Logic **51**(1–2) (1991) 79–98
13. Fix, L., Grumberg, O.: Verification of temporal properties. J. Logic Computat. **6**(3) (1996) 343–361
14. Slanina, M., Sipma, H.B., Manna, Z.: Proving ATL* properties of infinite-state systems. Technical Report REACT-TR-2006-02, Stanford University, Computer Science Department, REACT Group (2006) Avaliable at `http://react.stanford.edu/TR/`.
15. Slanina, M.: Control rules for reactive system games. In Fischer, B., Smith, D.R., eds.: Logic-Based Program Synthesis: State of the Art and Future Trends. AAAI Spring Symposium, The American Association for Artificial Intelligence, AAAI Press (2002) 95–104 Available from AAAI as Technical Report SS-02-05.
16. Zielonka, W.: Infinite games on finitely coloured graphs with applications to automata on infinite trees. Theoretical Computer Science **200** (1998) 135–183
17. Henzinger, T.A., Majumdar, R., Mang, F., Raskin, J.F.: Abstract interpretation of game properties. In: Proc. 7th Intern. Static Analysis Symp. (SAS). Volume 1824 of LNCS., Springer (2000) 220–239
18. Manna, Z., Pnueli, A.: Temporal verification diagrams. In: Proc. International Symposium on Theoretical Aspects of Computer Software. Volume 789 of LNCS., Springer (1994) 726–765