ATL Satisfiability is Indeed ExpTime-Complete

Dirk Walther[1] Carsten Lutz[2] Frank Wolter[1] Michael Wooldridge[1]

[1] University of Liverpool
UK
{dirk,frank,mjw}@csc.liv.ac.uk

[2] TU Dresden
Germany
lutz@tcs.inf.tu-dresden.de

Abstract

The Alternating-time Temporal Logic (ATL) of Alur, Henzinger, and Kupferman is being increasingly widely applied in the specification and verification of open distributed systems and game-like multi-agent systems. In this paper, we investigate the computational complexity of the satisfiability problem for ATL. For the case where the set of agents is fixed in advance, this problem was settled at ExpTime-complete in a result of van Drimmelen. If the set of agents is not fixed in advance, then van Drimmelen's construction yields a 2ExpTime upper bound. In this paper, we focus on the latter case and define three natural variations of the satisfiability problem. Although none of these variations fixes the set of agents in advance, we are able to prove containment in ExpTime for all of them by means of a type elimination construction—thus improving the existing 2ExpTime upper bound to a tight ExpTime one.

1 Introduction

Alternating-time Temporal Logic (ATL) is a logic of *strategic ability*, intended to support reasoning about the abilities of agents and coalitions of agents in *open systems* (i.e., game-like multi-agent systems) [3]. Introduced in 1997 [1], ATL appears to be rapidly gaining acceptance as a key formalism for reasoning about multi-agent systems. There are several reasons for the intense interest in ATL. From a language point of view, ATL can be seen as an elegant generalisation and extension of Computation Tree Logic (CTL), one of the most successful and widely applied formalisms for reasoning about reactive systems [5]. While in CTL, one is essentially restricted to stating that some property is either inevitable or possible, in ATL, one can also express *adversarial* properties, such as "agents 1 and 2 can cooperate to ensure that, no matter what the other agents do, the system will not enter an invalid state" (written: $\langle\langle 1,2\rangle\rangle \Box \mathit{valid}$). From the point of view of semantics, ATL is based on models (called *Alternating Transition Systems* – ATSs) that emphasise the game-like nature of distributed computing, thus reflecting current opinion on the semantics of multi-process systems. And finally, from the verification point of view, model checking in ATL has been shown to be no more complex than that of model checking in its counterpart CTL: the ATL model checking problem can be solved in time $O(m \cdot n)$, where m is the size of the model, and n is the size of the formula to be checked. As with CTL, this has enabled the development of practical model checking tools for ATL [4].

One interesting aspect of ATL is that its model checking problem subsumes a number of other important computational problems, such as *program synthesis*, *module checking*, and *controllability* [3, p.676]. However, the problem of *social procedure design* or *mechanism design* in ATL is best understood as a (constructive) *satisfiability checking* problem: we are given a specification of a social mechanism (such as a voting procedure [11]), expressed as a formula φ of ATL, and asked whether or not there exists a procedure that satisfies specification φ; and if so, we are asked to exhibit it. As such a procedure corresponds to a model of φ, social procedure design can be viewed as a proof of the satisfiability of φ that is constructive in the sense that an actual model is generated.

Although the complexity of the model checking problem for ATL was classified in the very first publications on ATL [1], the complexity of the satisfiability problem was not addressed. The fact that ATL is a generalisation of CTL immediately gives an ExpTime lower bound, but the question of whether or not ATL satisfiability was in ExpTime was left open.[1]

For the case where the set of relevant agents is fixed in advance, the complexity of the satisfiability problem for ATL was settled in 2003 with the publication by van Drimmelen of an automata-based ExpTime decision procedure [14]. More precisely, the approach was to show that ATL satisfiability can be reduced to the nonemptiness problem for alternating Büchi tree automata. For the overall decision procedure to have exponential running time, the branching degree of the constructed trees has to be polynomial in the size of the input formula. In van Drimmelen's proof, the constructed trees have branching degree k^n, where n is the number of agents and k is polynomial in the size of the input formula. Thus, a polynomial branching degree, and hence an overall ExpTime upper bound, is only obtained if the number of agents allowed to appear in input formulas is fixed beforehand, rather than being regarded as part of the input. Thus, the obtained ExpTime result can be stated as follows.

Theorem 1.1 (VanDrimmelen) *Suppose Σ is a fixed, finite set of agents. Then satisfiability of ATL-formulas based on Σ in an ATS over Σ is* ExpTime*-complete.*

[1]This is a simplification. As we shall see later, some variants of the ATL satisfiability problem do not inherit ExpTime-hardness from CTL in an immediate way.

If input formulas may contain arbitrarily many agents, then n, the number of agents, *is dependent on the input formula*. In this case, the branching degree of the constructed trees becomes exponential, and the decision procedure only yields a 2ExpTime upper bound. Thus, if we do not fix the set of agents in advance, the complexity of satisfiability in ATL is still open – between ExpTime and 2ExpTime. Note that van Drimmelen's approach cannot be generalised by choosing a better tree construction: as we will see later, in ATL it is possible to devise a formula that enforces a branching degree exponential in the number of agents.

Considering ATL with an unbounded supply of agents, we find there are several different ways of framing the satisfiability problem with respect to the agents that can appear in both the formula and the structure that satisfies the formula. In particular, there are different possibilities for the number of agents that occur in an ATS over which a formula is to be interpreted. With this observation in mind, consider the following three formulations of the ATL satisfiability problem:

(a) Given a finite set Σ of agents and a formula φ over Σ, is φ satisfiable in an ATS over Σ?

(b) Given a formula φ, is there a finite set Σ of agents (containing the agents referred to in φ) such that φ is satisfiable in an ATS over Σ?

(c) Given a formula φ, is φ satisfiable in an ATS over exactly the agents which occur in φ?

Van Drimmelen's construction does not give an ExpTime upper bound for any of these three variations. The main contribution of the present paper is to prove that, still, all three variations are ExpTime-complete.

The remainder of this paper is structured as follows. To begin, in Section 2, we introduce ATL both by way of some simple examples and its formal syntax and semantics. In Section 3, we discuss various possible formulations of the satisfiability problem for ATL, and how these problems relate to one-another. Our main result is proved in Section 4, and we present some conclusions in Section 5.

Author's note: The authors of the present paper would like to point out that we greatly appreciate Govert van Drimmelen's work on ATL complexity and the results of [14], which we regard as a an important milestone on the road to understanding this exciting new logic. We are very grateful to Govert for providing us with copies of his MSc thesis, upon which [14] was based, and for answering our queries in the very best spirit of scientific openness. We would also like to express our profound thanks to his supervisor Valentin Goranko, for the open and valuable discussions we have enjoyed on ATL satisfiability and related topics.

2 Alternating-time Temporal Logic (ATL)

ATL is a logic of *strategic cooperative ability.* It is intended to support reasoning about the abilities of agents and coalitions of agents in open distributed systems (also increasingly known as *multi-agent systems*). The syntactic expressions that facilitate such reasoning in ATL are known as *cooperation modalities.* A cooperation modality has the general form $\langle\langle C \rangle\rangle T$, where C is a set of agents (which intuitively correspond to the autonomously acting components of a system), and T is a temporal logic expression, of the form $\bigcirc\varphi$ ("next φ"), $\Box\varphi$ ("always φ"), $\Diamond\varphi$ ("eventually, φ"), or $\varphi\,\mathcal{U}\,\psi$ ("φ until ψ"). The meaning of a cooperation modality $\langle\langle C \rangle\rangle T$ is that the coalition C can cooperate to ensure that T is true; more precisely, that there exists a collective strategy for the system components in C such that, if these components act in the manner defined by this collective strategy, then T is guaranteed to be satisfied. In ATL, as in its ancestor CTL, we are not allowed to arbitrarily intermingle cooperation expressions and temporal logic formulas: every temporal expression ($\bigcirc$, $\Box$, $\mathcal{U}$, or $\Diamond$) must be preceded by a cooperation expression of the form $\langle\langle C \rangle\rangle$, for some set of agents C.[2]

To better understand ATL, here are some example formulas (see [3]).

$$\langle\langle C \rangle\rangle \Box\varphi$$

This formula asserts the *controllability* of the overall system by some coalition C with respect to property φ. That is, it states that the coalition C can cooperate to ensure that the property φ *always* holds in the system, no matter how the components of the system outside C behave.

$$\langle\langle a \rangle\rangle \Diamond see_b(msg)$$

This formula says that agent a can guarantee that agent b eventually sees the message msg (where $see_b(msg)$ is an atomic proposition).

$$\langle\langle a \rangle\rangle \Box\neg see_b(msg)$$

This formula says that agent a can ensure that agent b never sees the message msg.

One area of interest to the authors of the present paper is the use of ATL in the specification, verification, and synthesis of *social procedures* such as voting protocols [11]. Consider the following example (adapted from [11]).

> *Two agents, A and B, must choose between two outcomes, p and q. We want a mechanism that will allow them to choose, which will satisfy the*

[2]The variation of ATL that permits arbitrarily intermingled temporal logic and cooperation expressions is known as ATL*, and bears the same family relationship to ATL that the logic CTL* does to CTL [5].

following requirements. First, whatever happens, we definitely want an outcome to result – that is, we want either p or q to be selected. Second, we really do want the agents to be able to collectively choose an outcome. However, we do not want them to be able to bring about both outcomes simultaneously. Similarly, we do not want either agent to dominate: we want them both to have equal power.

We can elegantly capture these requirements using ATL, as follows.

(1) $\langle\langle\emptyset\rangle\rangle\bigcirc(p \vee q)$

(2) $(\langle\langle A, B\rangle\rangle\bigcirc p) \wedge (\langle\langle A, B\rangle\rangle\bigcirc q)$

(3) $\neg\langle\langle A, B\rangle\rangle\bigcirc(p \wedge q)$

(4) $(\neg\langle\langle A\rangle\rangle\bigcirc p) \wedge (\neg\langle\langle B\rangle\rangle\bigcirc p)$

(5) $(\neg\langle\langle A\rangle\rangle\bigcirc q) \wedge (\neg\langle\langle B\rangle\rangle\bigcirc q)$

The first requirement states that an outcome *must* result: this will happen inevitably, whatever the agents do. Requirement (2) states that the two agents can choose between the two outcomes: they have a collective strategy such that, if they follow this strategy, outcome x will occur, where x is either p or q. Requirement (3), however, says that the agents cannot choose *both* outcomes. Requirements (4) and (5) state that neither agent can bring about an outcome alone.

Now it is easy to see that there exists a voting protocol that satisfies these requirements. Consider the following mechanism, (from [12]), intended to permit the agents to select between the outcomes in accordance with these requirements.

The two agents vote on the outcomes, i.e., they each choose either p or q. If there is a consensus, then the consensus outcome is selected; if there is no consensus, (i.e., if the two agents vote differently), then an outcome p or q is selected non-deterministically.

Notice that, given this simple mechanism, the agents really can collectively choose the outcome, by cooperating. If they do not cooperate, however, then an outcome is chosen for them.

We give a precise description of this mechanism in Figure 1; the mechanism is specified in the REACTIVE MODULES language of the MOCHA model checking system for ATL [4]. Thus, the protocol contains three agents, which in MOCHA terminology are called `modules`. (Note that defining the protocol in this way requires an agent that was not named in the specification formulae, above.)

- `AgentA` and `AgentB` correspond to the A and B in our scenario. Each agent `controls` (i.e., has exclusive write access to) a variable that is used to record

```
-- voteA == false ...  agent A votes for outcome P
-- voteA == true ...  agent A votes for outcome Q
module AgentA
   interface voteA : bool
   atom controls voteA
   init update
        [] true -> voteA' := false
        [] true -> voteA' := true
   endatom
endmodule

-- voteB == false ...  agent B votes for outcome P
-- voteB == true ...  agent B votes for outcome Q
module AgentB
   interface voteB : bool
   atom controls voteB
   init update
        [] true -> voteB' := false
        [] true -> voteB' := true
   endatom
endmodule

-- outcome == false ...  P is selected
-- outcome == true ...  Q is selected
module Environment
   interface outcome :  bool
   external voteA, voteB : bool
   atom controls outcome awaits voteA, voteB
   init update
   -- if votes are the same, go with selected outcome
        [] (voteA' = voteB') -> outcome' := (voteA' & voteB')
   -- otherwise select outcome non-deterministically
        [] ~(voteA' = voteB') -> outcome' := true
        [] ~(voteA' = voteB') -> outcome' := false
   endatom
endmodule -- Environment

System := (AgentA || AgentB || Environment)
```

Figure 1: A simple social choice mechanism, defined in the ReactiveModules language of the MOCHA model checker.

their vote. Thus `voteA` records the vote of `AgentA`, where a value of `false` in this variable means voting for outcome `P`, while `true` implies voting for `Q`. The "program" of each agent is made up of two remaining guarded commands, which simply present the agent with a choice of voting either way.

- The `Environment` module is used to model the mechanism itself. This module simply looks at the two votes, and if they are the same, sets the variable `outcome` to be the consensus outcome; if the two votes are different, then the guarded commands defining `Environment`'s behaviour say that an outcome will be selected non-deterministically.

It is similarly easy to see how ATL can be used, in this way, to specify much more complex voting protocols and other related social procedures. It is important to note that this example is chosen for pedagogic reasons, and realistic protocols will be rather more elaborate. In particular, in realistic applications we will often have a considerably larger number of agents who participate in a protocol than just two. If we fix the set of agents in advance, then we impose an upper bound on the number of participants in the protocols that we can describe and synthesize. Therefore, a natural satisfiability problem for this application appears to be Variant (a) from above in which the set of agents is unbounded and given as part of the input.

Once we have a protocol, we can use an ATL model checker (such as MOCHA [4]), to automatically verify that our implementation of the requirements is correct. However, the problem of *synthesising* a protocol from such a specification corresponds to (constructively) checking the *satisfiability* of the specification—whence our interest in the satisfiability problem.

Let us now move on to consider the formal syntax and semantics of ATL.

Definition 2.1 [ATL Syntax] Let $\mathbf{\Pi}$ be a countable infinite set of *atomic propositions* and $\mathbf{\Sigma}$ a countable infinite set of *agents*. A *coalition* is a finite set $A \subseteq \mathbf{\Sigma}$ of agents. The set of ATL-formulas is the smallest set containing the following:

- p for $p \in \mathbf{\Pi}$;
- $\neg\varphi$, $\varphi \vee \psi$ for ATL-formulas φ and ψ;
- $\langle\langle A\rangle\rangle\bigcirc\varphi$, $\langle\langle A\rangle\rangle\Box\varphi$, $\langle\langle A\rangle\rangle\varphi\,\mathcal{U}\,\psi$ for a coalition A and ATL-formulas φ and ψ;

$\lhd$

The modality $\langle\langle\ \rangle\rangle$ is called a *path quantifier*, $\bigcirc$ ("next"), $\Box$ ("always"), and $\mathcal{U}$ ("until") *temporal operators*. Logical truth ($\top$) and the Boolean connectives ($\wedge$, $\rightarrow$, and $\leftrightarrow$) are defined as usual. The operator $\langle\langle A\rangle\rangle\Diamond\varphi$ used in the examples is defined as $\langle\langle A\rangle\rangle\top\,\mathcal{U}\,\varphi$. Observe that, unlike the $\mathsf{A}\Box$ and $\mathsf{E}\Box$ operators of CTL, the ATL operator $\langle\langle A\rangle\rangle\Box$ cannot be expressed in terms of the other operators.

Several versions of the semantics of ATL have been presented in the literature. We choose to work with *alternating transition systems* as introduced in [2].

Definition 2.2 [Alternating Transition Systems] An *alternating transition system (ATS)* for Σ is a tuple $\mathcal{S} = \langle\Pi, \Sigma, Q, \pi, \delta\rangle$ with $n \geq 1$ where

- $\Pi \subseteq \mathbf{\Pi}$ is a finite, non-empty set of *atomic propositions*,
- $\Sigma = \{a_1, \ldots, a_n\} \subseteq \mathbf{\Sigma}$ is a (finite) set of n *agents*,
- Q is a finite, non-empty set of *states*,

- $\pi : Q \rightarrow \mathbf{2}^\Pi$ is a *valuation function* which assigns to every state a set of propositions which are true there, and
- $\delta : Q \times \Sigma \rightarrow \mathbf{2}^{\mathbf{2}^Q}$ is a *transition function* which maps a state $q \in Q$ and an agent $a \in \Sigma$ to a set of *choices* $\delta(q, a)$ available to a at q such that the following condition is satisfied: for every state $q \in Q$ and every set $Q_{a_1}, \ldots, Q_{a_n}$ of choices $Q_{a_i} \in \delta(q, a_i)$, $1 \leq i \leq n$, the intersection $Q_{a_1} \cap \cdots \cap Q_{a_n}$ is a singleton set.

$\lhd$

Intuitively, $\delta(q, a)$ describes the *a-choices* available in q: when in state q, agent a chooses a set from $\delta(q, a)$ to ensure that the "next state" will be among those in the chosen set. It is natural to generalize this notion to *A-choices* for coalitions A: let $\mathcal{S} = \langle \Pi, \Sigma, Q, \pi, \delta \rangle$ be an ATS. For each state $q \in Q$ and each coalition $A \subseteq \Sigma$, set

$$\delta(q, A) := \begin{cases} \{Q_A \subseteq Q \mid Q_A = \bigcap_{a \in A} Q_a \text{ where for each } a \in A,\ Q_a \in \delta(q, a)\} & \text{if } A \neq \emptyset \\ \{\bigcup \delta(q, \Sigma)\} & \text{if } A = \emptyset \end{cases}.$$

When in state q, the coalition A may jointly choose a set from $\delta(q, A)$ to ensure that the next state is from this set. Clearly, $\delta(q, \Sigma)$ is a set of singletons. The states appearing in singletons in $\delta(q, \Sigma)$ are the *successors* of q, i.e., whatever choices the individual agents make, the next state of the system will be from $\bigcup \delta(q, \Sigma)$. This explains the definition of $\delta(q, \emptyset)$: the empty set of agents cannot influence the behaviour of the system, so the only choice that the empty coalition has is the set of all successors.

An infinite sequence $\lambda = q_0 q_1 q_2 \cdots \in Q^\omega$ of states is a *computation* if, for all positions $i \geq 0$, there is a choice $\{q_{i+1}\} \in \delta(q_i, \Sigma)$ (i.e., q_{i+1} is a successor of q_i). As a notational convention for any finite or infinite sequence $\lambda = \lambda_0 \lambda_1 \cdots$ and any $i \geq 0$, denote with $\lambda[i]$ the i-th component λ_i in λ and with $\lambda[0, i]$ the initial sequence $\lambda_0 \cdots \lambda_i$ of λ.

A *strategy* for an agent $a \in \Sigma$ is a mapping $f_a : Q^+ \rightarrow \mathbf{2}^Q$ such that for all $\lambda \in Q^*$ and all $q \in Q$, $f_a(\lambda \cdot q) \in \delta(q, a)$. Note that a strategy f_a maps each finite sequence $\lambda \cdot q$ of states to a choice in $\delta(q, a)$ available to agent a at the state q. A *strategy* for agents in a coalition $A \subseteq \Sigma$ is a set of strategies $F_A = \{f_a \mid a \in A\}$, one for each agent in A.

The set $\mathsf{out}(q, F_A)$ of *outcomes* of a strategy F_A starting at a state $q \in Q$ is the set of all computations $\lambda = q_0 q_1 q_2 \cdots \in Q^\omega$ such that $q_0 = q$ and $q_{i+1} \in \bigcap_{f_a \in F_A} f_a(\lambda[0, i])$ for all $i \geq 0$.

Definition 2.3 [ATL Semantics] Given an ATS $\mathcal{S} = \langle \Pi, \Sigma, Q, \pi, \delta \rangle$, the satisfaction relation $\models$ is inductively defined as follows: For all states q of $\mathcal{S}$, coalitions of agents $A \subseteq \Sigma$, agents $a \in \Sigma$, and ATL-formulas φ, φ_1, and φ_2, it holds that

- $\mathcal{S}, q \models p$ iff $p \in \pi(q)$ for all propositions $p \in \Pi$;

- $\mathcal{S}, q \models \neg\varphi$ iff $\mathcal{S}, q \not\models \varphi$;
- $\mathcal{S}, q \models \varphi_1 \vee \varphi_2$ iff $\mathcal{S}, q \models \varphi_1$ or $\mathcal{S}, q \models \varphi_2$;
- $\mathcal{S}, q \models \langle\langle A\rangle\rangle\bigcirc\varphi$ iff there is a strategy F_A such that for all computations $\lambda \in \mathsf{out}(q, F_A)$, it holds that $\mathcal{S}, \lambda[1] \models \varphi$;
- $\mathcal{S}, q \models \langle\langle A\rangle\rangle\Box\varphi$ iff there is a strategy F_A such that for all computations $\lambda \in \mathsf{out}(q, F_A)$, it holds that $\mathcal{S}, \lambda[i] \models \varphi$ for all positions $i \geq 0$;
- $\mathcal{S}, q \models \langle\langle A\rangle\rangle\varphi_1 \mathcal{U} \varphi_2$ iff there is a strategy F_A such that for all computations $\lambda \in \mathsf{out}(q, F_A)$, there is a position $i \geq 0$ such that $\mathcal{S}, \lambda[i] \models \varphi_2$ and $\mathcal{S}, \lambda[j] \models \varphi_1$ for all positions j with $0 \leq j < i$.

If for some state q of some ATS $\mathcal{S}$ it holds that $\mathcal{S}, q \models \varphi$, then the ATL-formula φ is *true* at q, and $\mathcal{S}$ is called a *model* of φ. An ATL-formula is *satisfiable* if it has a model, and it is *valid* if it is true at all states in any ATS. $\lhd$

Notice that there is an intimate relationship between CTL and ATL. Let Σ be the set of all agents in an ATS $\mathcal{S}$. On $\mathcal{S}$, we can then interpret CTL's existential path quantifier E in ATL as the cooperation expression $\langle\langle\Sigma\rangle\rangle$, while we can interpret CTL's universal path quantifier A in ATL as the cooperation expression $\langle\langle\emptyset\rangle\rangle$. Clearly, this translation only works if the set of agents Σ is finite and known in advance.

The automata-based decision procedure in [14] sketched in the introduction yields a 2ExpTime upper bound because, in the constructed tree models, the branching degree is exponential in the number of agents. To illustrate that this branching cannot easily be reduced, we exhibit a sequence of ATL formulas $(\varphi_i)_{i\in\mathbb{N}}$ such that, for any ATS $\mathcal{S}$, state q, and $i \geq 0$, $\mathcal{S}, q \models \varphi_i$ implies that q has at least 2^i successors in $\mathcal{S}$:

$$\varphi_i := \bigwedge_{1\leq j\leq i} \big(\langle\langle a_j\rangle\rangle\bigcirc p_j \wedge \langle\langle a_j\rangle\rangle\bigcirc\neg p_j\big)$$

As every agent a_i may choose the propositional letter p_i to be true or false at a successor state, jointly the agents $a_1, \ldots, a_k$ may choose any possible valuation of $p_1, \ldots, p_k$ for a successor state. As there are 2^i such valuations, there must be as many successors.

3 Varieties of Satisfiability for ATL

The key syntactic difference between ATL and its predecessor CTL is that formulas of ATL explicitly refer to agents. We must be mindful of the way in which such agents are interpreted in the formulation of the ATL satisfiability problem. To better understand why, consider the following ATL formula (adapted from [11, p.47]).

$$\neg\langle\langle a\rangle\rangle p \land \neg\langle\langle a\rangle\rangle q \land \langle\langle a\rangle\rangle(p \lor q)$$

This formula expresses the fact that agent a cannot make p true, and cannot make q true; but it can make either p or q true. Now, is this ATL formula satisfiable? It depends on the range of ATSs we are prepared to consider. If we admit *arbitrary* ATSs as witness to its satisfiability, then the answer is yes: one can easily construct an ATS containing two or more agents that satisfies it. However, suppose we only consider ATSs that contain at most *one* agent. By virtue of the fact that the formula is well-formed, the agent in the structure must be a. But then the choice sets for this agent must be singletons, and it is then easy to see that the formula could not be satisfied in such a model. So: the agents in the structures we are prepared to consider *are* important in determining the satisfiability or otherwise of a formula, and even unknown agents that are not referred to in a formula can play a part in determining whether or not the formula is satisfied in a structure. Notice that the presence of unknown agents introduces an element of non-determinism, as the agents that occur in a formula cannot completely determine the behaviour of the system anymore.

With these concerns in mind, consider the following three variations of satisfiability for ATL.

(a) *Satisfiability over given sets of agents*:

Given a finite set Σ of agents and a formula φ over Σ, is φ satisfiable in an ATS over Σ?

(b) *Satisfiability over arbitrary sets of agents*:

Given a formula φ, is there a finite set Σ of agents (containing the agents referred to in φ) such that φ is satisfiable in an ATS over Σ?

(c) *Satisfiability over formula-defined sets of agents*:

Given a formula φ, is φ satisfiable in an ATS over exactly the agents which occur in φ?

As already noted, van Drimmelen's construction does not give an ExpTime upper bound for any of these variations. The automata theoretic algorithm presented in [14] yields a 2ExpTime upper bound Cases (a) and (c). It does not a priori give any upper bound for Case (b), although a 2ExpTime bound follows from this algorithm together with Lemma 3.2 below. Our main result is as follows.

Theorem 3.1 *Problems (a), (b), and (c) are* ExpTime*-complete.*

The remainder of this paper is largely devoted to the proof of the ExpTime upper bound. The lower bound will be discussed briefly at the end of the next section. We begin by showing that it suffices to prove the upper bound for Problem (c), as the other two cases may be reduced to this.

Lemma 3.2 *Problems (a) and (c) are polynomially reducible to each other, while Problem (b) is polynomially reducible to (a). In fact, we even have the stronger property that, for each formula φ and each set of agents $\Sigma \supset \Sigma_\varphi$, φ is satisfiable in an ATS for Σ iff it is satisfiable in an ATS for $\Sigma_\varphi \cup \{a\}$, for one fresh agent a.*

Proof. Note that (c) is a special case of (a), where Σ coincides with the set of agents Σ_φ which occur in φ. Conversely, given Σ and φ from (a), conjunctively add to φ any valid formula ψ containing exactly the agents from Σ which do not occur in φ. Then φ is satisfiable in an ATS for Σ iff $\varphi \wedge \psi$ is satisfiable in an ATS for the agents which occur in $\varphi \wedge \psi$. It thus remains to prove the second part, i.e., that for each formula φ and each set of agents $\Sigma \supset \Sigma_\varphi$, φ is satisfiable in an ATS for Σ iff it is satisfiable in an ATS for $\Sigma_\varphi \cup \{a\}$, for one fresh agent a.

"$\Rightarrow$": Suppose $\mathcal{S}$ is an ATS for $\Sigma \supset \Sigma_\varphi$ such that $\mathcal{S}$ satisfies φ. We convert $\mathcal{S}$ into an ATS $\mathcal{S}'$ for $\Sigma_\varphi \uplus \{a\}$ that also satisfies φ. Define the transition function δ' of $\mathcal{S}'$ in terms of δ in $\mathcal{S}$ as follows: for all $q \in Q$,

- $\delta'(q, a') := \delta(q, a')$ for each $a' \in \Sigma_\varphi$, and
- $\delta'(q, a) := \delta(q, \Sigma \setminus \Sigma_\varphi)$.

It is easy to show by structural induction that, for all $q \in Q$ and all formulas ψ using only agents from the set Σ_φ, we have $\mathcal{S}, q \models \psi$ iff $\mathcal{S}', q \models \psi$. Thus, $\mathcal{S}'$ is a model of φ as required.

"$\Leftarrow$": Suppose $\mathcal{S}$ is an ATS for $\Sigma_\varphi \uplus \{a\}$ such that $\mathcal{S}$ satisfies φ. Let $a' \in \Sigma \setminus \Sigma_\varphi$. We convert $\mathcal{S}$ into an ATS $\mathcal{S}'$ for Σ such that $\mathcal{S}'$ still satisfies φ. Define δ' of $\mathcal{S}'$ in terms of δ in $\mathcal{S}$ as follows: for all $q \in Q$,

- $\delta'(q, a'') = \delta(q, a'')$ for each $a'' \in \Sigma_\varphi$,
- $\delta'(q, a') = \delta(q, a)$, and
- $\delta'(q, a'') = \{Q\}$ for each $a'' \in \Sigma \setminus (\Sigma_\varphi \cup \{a'\})$.

Again, it is easy to show by structural induction that, for all $q \in Q$ and all formulas ψ using only agents from the set Σ_φ, we have $\mathcal{S}, q \models \psi$ iff $\mathcal{S}', q \models \psi$. QED

4 The Main Result

This section is devoted to the proof of Theorem 3.1. Our main result is containment in ExpTime of Problem (c) from the previous section, i.e., satisfiability of ATL formulas φ in ATSs over exactly the agents occurring in φ. By Lemma 3.2, this yields ExpTime upper bounds also for problems (a) and (b). The ExpTime lower bounds for (a) and (c) are immediate by reduction of CTL as sketched at the end of Section 2. To establish an ExpTime lower bound for Problem (b), we will reduce the global consequence problem in the modal logic K.

We start with the ExpTime upper bound for Problem (c). Our approach is to use a type elimination construction similar to the one commonly used for CTL [6, 5]. One advantage of this approach is that it is constructive: if the input formula φ is satisfiable, then the proof actually constructs a model of φ.[3] Thus, our algorithm can be used e.g., for the synthesis of social procedures as sketched in Sections 1 and 2. We should note that recently a similar construction has been independently developed by Goranko and van Drimmelen in [8]. However, Goranko and van Drimmelen use their construction to prove completeness of an ATL axiomatization rather than for obtaining upper complexity bounds.

We start the presentation of our decision procedure with a number of definitions.

Definition 4.1 [Extended Closure] Let φ be an ATL-formula. The *extended closure* $\mathsf{ecl}(\varphi)$ of φ is the smallest set which is closed under the following conditions:

- $\varphi \in \mathsf{ecl}(\varphi)$;
- $\mathsf{ecl}(\varphi)$ is closed under subformulas;
- $\mathsf{ecl}(\varphi)$ is closed under single negation;
- if $\langle\langle A\rangle\rangle\Box\psi \in \mathsf{ecl}(\varphi)$, then $\langle\langle A\rangle\rangle\bigcirc\langle\langle A\rangle\rangle\Box\psi \in \mathsf{ecl}(\varphi)$;
- if $\langle\langle A\rangle\rangle\psi\,\mathcal{U}\,\vartheta \in \mathsf{ecl}(\varphi)$, then $\langle\langle A\rangle\rangle\bigcirc\langle\langle A\rangle\rangle\psi\,\mathcal{U}\,\vartheta \in \mathsf{ecl}(\varphi)$.

$\lhd$

It is easy to verify that for a given ATL-formula φ, the cardinality of the extended closure $\mathsf{ecl}(\varphi)$ is linear in the length of φ.

Definition 4.2 [Type] Let φ be an ATL-formula. The set $\Psi \subseteq \mathsf{ecl}(\varphi)$ is a *type for* φ if the following conditions are satisfied:

(T1) $\psi_1 \vee \psi_2 \in \Psi$ iff $\psi_1 \in \Psi$ or $\psi_2 \in \Psi$, for all $\psi_1 \vee \psi_2 \in \mathsf{ecl}(\varphi)$;

(T2) $\psi \in \Psi$ iff $\neg\psi \notin \Psi$, for all $\neg\psi \in \mathsf{ecl}(\varphi)$;

(T3) $\langle\langle A\rangle\rangle\Box\psi \in \Psi$ iff $\{\psi, \langle\langle A\rangle\rangle\bigcirc\langle\langle A\rangle\rangle\Box\psi\} \subseteq \Psi$, for all $\langle\langle A\rangle\rangle\Box\psi \in \mathsf{ecl}(\varphi)$;

(T4) $\langle\langle A\rangle\rangle\psi\,\mathcal{U}\,\vartheta \in \Psi$ iff $\vartheta \in \Psi$ or $\{\psi, \langle\langle A\rangle\rangle\bigcirc\langle\langle A\rangle\rangle\psi\,\mathcal{U}\,\vartheta\} \subseteq \Psi$, for all $\langle\langle A\rangle\rangle\psi\,\mathcal{U}\,\vartheta \in \mathsf{ecl}(\varphi)$.

The set of all types for φ is designated by Γ_φ. $\lhd$

[3]This model is finite and of bounded size: the number of states is at most exponential in the length of φ.

Before continuing, we introduce some convenient notions. With Σ_φ, we denote the set of agents which occur in the formula φ. Moreover, we assume that $|\Sigma_\varphi| = n$ implies $\Sigma_\varphi = \{1, \dots, n\}$, i.e., the agents are numbered and their name coincides with their number. We call ATL-formulas of the form $\langle\!\langle A \rangle\!\rangle \psi_1 \mathcal{U} \psi_2$ or $\neg\langle\!\langle A \rangle\!\rangle \Box\psi$ *eventualities.* A *next-formula* is a formula of the form $\langle\!\langle A \rangle\!\rangle \bigcirc\psi$ or $\neg\langle\!\langle A \rangle\!\rangle \bigcirc\psi$. For each formula φ, assume that all next-formulas in $\mathsf{ecl}(\varphi)$ are linearly ordered and use $\sharp_\psi$ to denote the number of the next-formula $\psi \in \mathsf{ecl}(\varphi)$ (the numbering starts with 0 and the formula φ will be clear from the context). The ordering is such that no negative next-formula occurs before a positive one. Since there are as many positive next-formulas in $\mathsf{ecl}(\varphi)$ as negative ones, we obtain an enumeration $\psi_0, \dots, \psi_{k-1}$ of k next-formulas, with positive next-formulas $\psi_0, \dots, \psi_{k/2-1}$ and negative next-formulas $\psi_{k/2}, \dots, \psi_{k-1}$.

To understand the following, central definition, let us sketch some details of the ATS $\mathcal{S}$ that our algorithm attempts to build as a model for the input formula φ. If $n = |\Sigma_\varphi|$ and k is the number of next-formulas in $\mathsf{ecl}(\varphi)$, then (regardless of some technical details) the states of $\mathcal{S}$ consist of sequences of n-tuples whose components take values from $\{0, \dots, k-1\}$. The set of all such n-tuples is denoted with $[k/n]$, and the states of $\mathcal{S}$ will thus be from $[k/n]^*$. If $q \in [k/n]^*$ is a state of $\mathcal{S}$, then $\{q \cdot \vec{t} \mid \vec{t} \in [k/n]\}$ will be the set of its potential successors, i.e., the choices in $\delta(q, a)$ will be subsets of this set. When constructing $\mathcal{S}$, each state $q \in [k/n]^*$ will have to satisfy a number of positive next-formulas and a number of negative next-formulas. Clearly, having to satisfy a positive next-formula $\langle\!\langle A \rangle\!\rangle \bigcirc\psi$ at q means that there has to be an A-choice $C \in \delta(q, A)$ such that all states in C satisfy ψ. Similarly, having to satisfy a negative next-formula $\neg\langle\!\langle A \rangle\!\rangle \bigcirc\psi$ at q means that all A-choices $C \in \delta(q, A)$ have to contain a state satisfying $\neg\psi$. This can be achieved by defining the transition function and assigning formulas to successors as follows:

(i) For each agent a, we set

$$\delta(q, a) := \{\{q \cdot \vec{t} \in [k/n] \mid \vec{t} = (t_0, \dots, t_{n-1}) \text{ and } t_a = p\} \mid p < k/2\}.^4$$

Intuitively, every agent "owns" a position in $\vec{t}$, and via this position he can make an a-choice in state q by "voting" for a positive next-formula that he wants to be satisfied.

Due to this definition, for coalitions of agents A we can characterize A-choices as follows: A subset $S \subseteq [k/n]$ is called an *A-voting set* if there exists a mapping $\tau : A \to \{0, \dots, k/2 - 1\}$ such that

$$S := \{\vec{t} = (t_0, \dots, t_{n-1}) \mid t_a = \tau(a) \text{ for all } a \in A\}.$$

Then, the elements of $\delta(q, A)$ are exactly the sets $\{q \cdot \vec{t} \mid \vec{t} \in S\}$ with S being an A-voting set.

[4]Recall that we assume agents to be natural numbers.

(ii) To satisfy a positive next-formula $\langle\!\langle A \rangle\!\rangle \bigcirc \psi$ at q, we use the voting set S in which all agents $a \in A$ vote for this formula, i.e.,

$$S = \{\vec{t} = (t_0, \dots, t_{n-1}) \mid t_a = \sharp_{\langle\!\langle A \rangle\!\rangle \bigcirc \psi} \text{ for all } a \in A\} \in \delta(q, A).$$

The ATS $\mathcal{S}$ is constructed such that all states in the corresponding A-choice $\{q \cdot \vec{t} \mid \vec{t} \in S\}$ make ψ true.

(iii) To satisfy a negative next-formula $\neg\langle\!\langle A \rangle\!\rangle \bigcirc \psi$ at q, we have to pick an element from every A-choice $C \in \delta(q, A)$, and then make ψ false at the picked elements.

Note that, in being a member of an A-choice, a picked element will automatically also be a member of an A'-choice for all $A' \subseteq A$. This is fine as $\neg\langle\!\langle A \rangle\!\rangle \bigcirc \psi$ implies $\neg\langle\!\langle A' \rangle\!\rangle \bigcirc \psi$.

However, if $B \not\subseteq A$, then $\neg\langle\!\langle A \rangle\!\rangle \bigcirc \psi$ does not imply $\neg\langle\!\langle B \rangle\!\rangle \bigcirc \psi$. Thus we should be careful that the picked elements from A-choices are not elements of B-choices for any such B. This is implemented by demanding that the element $\vec{t} = (t_0, \dots, t_{n-1})$ picked for an A-choice satisfies $t_a \geq k/2$ for each $a \notin A$.

The description of how exactly we pick elements is given after the next definition.

(iv) A special role is played by negative next-formulas $\neg\langle\!\langle \Sigma_\varphi \rangle\!\rangle \bigcirc \psi$. As we are working with formula-defined sets of agents, Σ_φ is the set of all agents in $\mathcal{S}$. For this reason, such negative next-formulas behave differently from formulas referring to smaller sets of agents. For example, $\neg\langle\!\langle A \rangle\!\rangle \bigcirc \psi$ and $\neg\langle\!\langle A \rangle\!\rangle \bigcirc \psi'$ imply $\neg\langle\!\langle A \rangle\!\rangle \bigcirc (\psi \vee \psi')$ if and only if $A = \Sigma_\varphi$. However, dealing with formulas $\neg\langle\!\langle \Sigma_\varphi \rangle\!\rangle \bigcirc \psi$ is simple: they merely state that no successor of q satisfies ψ.

The whole picture of the ATS construction is somewhat more complicated due to the presence of box formulas and until formulas, which we will address later.

We now introduce a "refutation function" whose purpose is to pick, for states q that have to satisfy a negative next-formula $\neg\langle\!\langle A \rangle\!\rangle \bigcirc \psi$, successors that refute ψ as explained under (iii) above.

Definition 4.3 [Refutation Function] Let φ be an ATL-formula, $n = |\Sigma_\varphi|$, and k the number of next-formulas in $\mathsf{ecl}(\varphi)$. We define a partial function

$$f : [k/n] \times 2^{\Sigma_\varphi} \to \{k/2, \dots, k-1\}$$

mapping vectors and coalitions of agents to (numbers of) negative next-formulas: for each set $A \subset \Sigma_\varphi$ of agents , fix an agent $a_A \in \Sigma_\varphi \backslash A$. Then set, for all $\vec{t} = (t_0, \dots, t_{n-1}) \in [k/n]$ and $A \subseteq \Sigma_\varphi$,

$$f(\vec{t}, A) := \begin{cases} \sharp_{\neg\langle\!\langle A \rangle\!\rangle \bigcirc \psi} & \text{if } t_{a_A} = \sharp_{\neg\langle\!\langle A \rangle\!\rangle \bigcirc \psi} \text{ and for all } a \in \Sigma_\varphi,\ t_a < k/2 \text{ iff } a \in A \\ \text{undefined} & \text{if there is no such } \neg\langle\!\langle A \rangle\!\rangle \bigcirc \psi. \end{cases}$$

$\lhd$

Intuitively, $f(\vec{t}, A) = \sharp_{\neg\langle\langle A\rangle\rangle\bigcirc\psi}$ means that, for every state q satisfying $\neg\langle\langle A\rangle\rangle\bigcirc\psi$, the successor $q \cdot \vec{t}$ has to refute ψ. Note that there may be more than one successor of q refuting ψ: at least one element of each A-choice. Formally, the most important properties of the refutation function are the following:

1. for each formula $\neg\langle\langle A'\rangle\rangle\bigcirc\psi' \in \mathsf{ecl}(\varphi)$ with $A' \subset \Sigma_\varphi$ and each A'-voting set S, there is an element $\vec{t'} \in S$ such that $f(\vec{t'}, A') = \sharp_{\neg\langle\langle A'\rangle\rangle\bigcirc\psi'}$;[5]

2. for all $\vec{t'} = (t'_0, \ldots, t'_{n-1}) \in [k/n]$, $f(\vec{t'}, A') = \sharp_{\neg\langle\langle A'\rangle\rangle\bigcirc\psi'}$ implies $t'_a \geq k/2$ for all $a \in \Sigma_\varphi \setminus A'$ (c.f. Explanation (iii));

3. for each $\vec{t} \in [k/n]$, there is at most a single $A \subseteq \Sigma_\varphi$ with $f(\vec{t}, A)$ defined.

It is easily verified that the function f from Definition 4.3 indeed satisfies these properties. A different function satisfying the properties is given by van Drimmelen in [14]. To determine the satisfiability of an ATL formula φ, the algorithm developed in this section will check for the existence of a model that is composed from certain trees, so-called *φ-trees.*

Definition 4.4 [φ-tree, ϑ-vector, $\bigcirc$-matching] Let φ be an ATL-formula, $n = |\Sigma_\varphi|$, and k the number of next-formulas in $\mathsf{ecl}(\varphi)$. For each next-formula $\vartheta \in \mathsf{ecl}(\varphi)$ and vector $\vec{t} = (t_0, \ldots, t_{n-1}) \in [k/n]$,

- if $\vartheta = \langle\langle A\rangle\rangle\bigcirc\psi$ and $t_a = \sharp_\vartheta$ for each $a \in A$, then $\vec{t}$ is called a *ϑ-vector*;
- if $\vartheta = \neg\langle\langle A\rangle\rangle\bigcirc\psi$ with $A \subset \Sigma_\varphi$, then $\vec{t}$ is called a *ϑ-vector* if $f(\vec{t}, A) = \sharp_\vartheta$;
- if $\vartheta = \neg\langle\langle \Sigma_\varphi\rangle\rangle\bigcirc\psi \in \mathsf{ecl}(\varphi)$, then $\vec{t}$ is called a *ϑ-vector.*

For each type $\Psi \in \Gamma_\varphi$ and each $\vec{t} = (t_0, \ldots, t_{n-1}) \in [k/n]$, let $S_\Psi(\vec{t}) \subseteq \mathsf{ecl}(\varphi)$ be the smallest set such that

(M1) if $\langle\langle A\rangle\rangle\bigcirc\psi \in \Psi$ and $\vec{t}$ is a $\langle\langle A\rangle\rangle\bigcirc\psi$-vector, then $\psi \in S_\Psi(\vec{t})$, and

(M2) if $\neg\langle\langle A\rangle\rangle\bigcirc\psi \in \Psi$ and $\vec{t}$ is a $\neg\langle\langle A\rangle\rangle\bigcirc\psi$-vector, then $\neg\psi \in S_\Psi(\vec{t})$.

Given a set M, a *$\langle M, k, n\rangle$-tree* T is a mapping T from a finite prefix-closed subset of $[k/n]^*$ to M. A *$\langle \Gamma_\varphi, k, n\rangle$-tree* is called a *$\varphi$-tree.* A φ-tree T is called *$\bigcirc$-matching* if, for all $\alpha \in \mathsf{dom}(T)$ and all $\vec{t} \in [k/n]$, $\alpha \cdot \vec{t} \in \mathsf{dom}(T)$ implies $S_{T(\alpha)}(\vec{t}) \subseteq T(\alpha \cdot \vec{t})$. $\lhd$

Intuitively, a vector $\vec{t}$ is a $\langle\langle A\rangle\rangle\bigcirc\psi$-vector if, for all states q satisfying $\langle\langle A\rangle\rangle\bigcirc\psi$, the successor $q \cdot \vec{t}$ has to satisfy ψ, c.f. Explanation (ii) above. The $\neg\langle\langle A\rangle\rangle\bigcirc\psi$-vectors can be understood in an analogous way. This intuition is reflected in (M1) and (M2) and in the definition of $\bigcirc$-matching.

[5]Recall that A'-voting sets correspond to A'-choices; c.f. Explanation (i) above.

Up to this point, we have set up the basic machinery to define ATSs based on the states $[k/n]^*$, and to treat satisfaction of (positive and negative) next-formulas. To deal with eventualities, we introduce φ-trees that witness their satisfaction, so-called *witness trees.* Their definition is largely analogous to the corresponding construction for CTL [5].

Definition 4.5 [Witness Tree] Let φ be an ATL-formula, Γ a set of types for φ, and $\Psi \in \Gamma$. A φ-tree T is called a *witness-tree rooted at* Ψ *in* Γ *for a formula* $\langle\langle A\rangle\rangle\psi\,\mathcal{U}\,\vartheta$ if it satisfies the following properties:

1. for all $\alpha \in \mathsf{dom}(T)$, $T(\alpha) \in \Gamma$;
2. T is $\bigcirc$-matching;
3. $T(\varepsilon) = \Psi$;
4. for all $\alpha \in \mathsf{dom}(T)$, $\langle\langle A\rangle\rangle\psi\,\mathcal{U}\,\vartheta \in T(\alpha)$;
5. for all non-leaf nodes α, $\psi \in T(\alpha)$;
6. for all leaf nodes α, $\vartheta \in T(\alpha)$;
7. if $\alpha \in \mathsf{dom}(T)$, $\langle\langle A\rangle\rangle\bigcirc\langle\langle A\rangle\rangle\psi\,\mathcal{U}\,\vartheta \in T(\alpha)$, $\vartheta \notin T(\alpha)$, and $\vec{t}$ is a $\langle\langle A\rangle\rangle\bigcirc\langle\langle A\rangle\rangle\psi\,\mathcal{U}\,\vartheta$-vector, then $\alpha \cdot \vec{t} \in \mathsf{dom}(T)$.

T is called a *witness-tree rooted at* Ψ *in* Γ for a formula $\neg\langle\langle A\rangle\rangle\Box\psi$ if it satisfies the following properties:

1. for all $\alpha \in \mathsf{dom}(T)$, $T(\alpha) \in \Gamma$;
2. T is $\bigcirc$-matching;
3. $T(\varepsilon) = \Psi$;
4. for all $\alpha \in \mathsf{dom}(T)$, $\neg\langle\langle A\rangle\rangle\Box\psi \in T(\alpha)$;
5. for all leaf nodes α, $\neg\psi \in T(\alpha)$
6. if $\alpha \in \mathsf{dom}(T)$, $\neg\langle\langle A\rangle\rangle\bigcirc\langle\langle A\rangle\rangle\Box\psi \in T(\alpha)$, $\neg\psi \notin T(\alpha)$, and $\vec{t}$ is a $\neg\langle\langle A\rangle\rangle\bigcirc\langle\langle A\rangle\rangle\Box\psi$-vector, then $\alpha \cdot \vec{t} \in \mathsf{dom}(T)$.

$\lhd$

Our decision procedure is based on the following core notion of *realizability.* Intuitively, a type Ψ is realizable in a set of types Γ if it is possible to (a) satisfy all next-formulas in Ψ using only types from Γ, and (b) construct witness trees for all eventualities in Ψ using only types from Γ.

Definition 4.6 [Realizable] Let φ be an ATL-formula and Γ a set of types for φ. A type $\Psi \in \Gamma$ is *realizable in* Γ if the following conditions are satisfied:

1. for each $\vec{t} \in [k/n]$, there is a $\Psi' \in \Gamma$ such that $S_\Psi(\vec{t}) \subseteq \Psi'$;
2. for each $\langle\langle A\rangle\rangle\psi\,\mathcal{U}\,\vartheta \in \Psi$, there is a $\langle\langle A\rangle\rangle\psi\,\mathcal{U}\,\vartheta$-witness tree rooted at Ψ in Γ;
3. for each $\neg\langle\langle A\rangle\rangle\Box\psi \in \Psi$, there is a $\neg\langle\langle A\rangle\rangle\Box\psi$-witness tree rooted at Ψ in Γ.

$\lhd$

We are now ready to describe the decision procedure. The idea is to start with all types for the input formula, then repeatedly eliminate types that are not realizable, and finally check whether there is a type that survived elimination and contains the input formula. Let φ be an ATL-formula whose satisfiability is to be decided. Inductively compute a sequence $\Delta_0, \Delta_1, \cdots$ of sets of types for φ as follows:

$$\begin{array}{lcl} \Delta_0 & := & \Gamma_\varphi \\ \Delta_{i+1} & := & \{\Psi \in \Delta_i \mid \Psi \text{ is realizable in } \Delta_i\}. \end{array}$$

Then $\Delta_{i+1} = \Delta_i$ for some $i \geq 0$. Let $m = i$ for the first such i. The algorithm returns "Yes, φ is satisfiable in an ATS for Σ_φ" if φ is contained in some type in Δ_m, and "No" otherwise.

We proceed as follows: first we show that this procedure is effective by proving that the existence of witness trees is decidable in exponential time. Second, we prove soundness and completeness of the procedure. Finally, we establish that it runs in exponential time.

Lemma 4.7 *Let Γ be a set of types for an ATL-formula φ. Then the existence of witness trees in Γ can be decided in time exponential in the length of φ.*

Proof. Let φ and Γ be as in the lemma, and Ψ_0 a type in Γ. We only show how to check the existence of witness trees for formulas of the form $\langle\langle A\rangle\rangle\psi\mathcal{U}\vartheta$. Witness trees for formulas $\neg\langle\langle A\rangle\rangle\Box\psi$ can be treated analogously. Thus, suppose we want to check the existence of a witness tree for $\langle\langle A\rangle\rangle\psi\mathcal{U}\vartheta \in \Psi_0$ rooted at Ψ_0 in Γ. Start with identifying leaf nodes of possible witness trees by marking all types in Γ which contain $\langle\langle A\rangle\rangle\psi\mathcal{U}\vartheta$ and ϑ. Then identify inner nodes of possible witness trees as follows: For all unmarked types $\Psi \in \Gamma$ such that $\langle\langle A\rangle\rangle\psi\mathcal{U}\vartheta \in \Psi$, mark Ψ if $\psi \in \Psi$ and for all $\langle\langle A\rangle\rangle\bigcirc\langle\langle A\rangle\rangle\psi\mathcal{U}\vartheta$-vectors $\vec{t} \in [k/n]$, there is a type $\Psi' \in \Gamma$ such that:

(i) $S_\Psi(\vec{t}) \subseteq \Psi'$, and

(ii) Ψ' is marked.

Repeatedly apply this procedure until no more types in Γ get marked. Note that this process must terminate since Γ contains only finitely many types.

It is easy to see that a witness tree for $\langle\!\langle A\rangle\!\rangle\psi\mathcal{U}\vartheta$ exists iff Ψ_0 was marked. For the left-to-right direction, suppose a witness tree exists. At the beginning of the marking procedure, all leaf nodes of possible witness trees are marked. In every subsequent round, all inner nodes of possible witness trees in increasing distance from the leaf nodes will be marked. Hence, eventually Ψ_0 will be marked. For the right-to-left direction, assume that Ψ_0 is marked and consider the following informal construction of a witness tree. Take the type Ψ_0 as the root. For each leaf and for all $\langle\!\langle A\rangle\!\rangle\bigcirc\langle\!\langle A\rangle\!\rangle\psi\mathcal{U}\vartheta$-vectors, add a successor type that satisfies (i) and (ii) until a type is reached which was marked at the beginning. Note that this process will not generate any infinite paths since, when choosing a successor type for some type Ψ, we can only use a type that was marked strictly before Ψ was marked. It is readily checked that all properties of a witness tree are fulfilled by the obtained tree.

For the complexity of the algorithm that checks for witness trees consider the following. Let $n = |\varphi|$. Note that the cardinality of the extended closure $\mathsf{ecl}(\varphi)$ is linear in n, i.e., $|\mathsf{ecl}(\varphi)| = c \cdot n$ for some constant $c \geq 1$. Since $\Gamma \subseteq \Gamma_\varphi \subseteq 2^{\mathsf{ecl}(\varphi)}$, it holds that $|\Gamma| \leq 2^{c\cdot n}$. For marking the leaf nodes, maximal $2^{c\cdot n}$ types have to be considered. For the marking of the inner nodes consider the following. Since there are at most $2^{c\cdot n}$ types in Γ and in each marking round at least one type gets marked, there are maximal $2^{c\cdot n}$ marking rounds. In each such round, maximal $2^{c\cdot n}$ yet unmarked types have to be checked. In order to find out whether to mark such a type, not more than n^n vectors have to be considered and for each such vector, tests for conditions (i) and (ii) with maximal $2^{c\cdot n}$ types need to be performed. Altogether this yields an upper bound of $2^{c\cdot n} + 2^{c\cdot n} \cdot 2^{c\cdot n} \cdot n^n \cdot 2^{c\cdot n} = 2^{\mathcal{O}(n^2)}$ steps. Thus the existence of witness trees can be checked in time exponential in the length of φ. QED

Lemma 4.8 *Let φ be an ATL-formula. Then the procedure returns "Yes, the input formula φ is satisfiable in an ATS for Σ_φ" iff this is indeed the case.*

Proof. Suppose φ is given, and let $\Sigma = \Sigma_\varphi$ and $n = |\Sigma_\varphi|$.

"$\Rightarrow$" (Soundness) Assume that the elimination procedure was started on input φ and returns "Yes, the input formula φ is satisfiable". Let $\Gamma = \{\Psi_0, \ldots, \Psi_{m-1}\}$ be the computed set of types. Then all types of Γ are realizable in Γ and there is a type $\Psi \in \Gamma$ with $\varphi \in \Psi$. Our aim is to construct an ATS that is a model of φ.

To this end, enumerate all eventualities in $\mathsf{ecl}(\varphi)$ by $\psi_0, \ldots, \psi_{\ell-1}$. For each i with $i < \ell$ and each j with $j < m$, fix a φ-tree $T_{\langle\psi_i,\Psi_j\rangle}$ as follows:

- If $\psi_i \in \Psi_j$, then fix a ψ_i-witness tree T rooted at Ψ_j in Γ. Supplement all inner nodes of T with missing successors: for each inner node $\alpha \in \mathsf{dom}(T)$ and each

$\vec{t} \in [k/n]$, if $\alpha \cdot \vec{t} \notin \mathsf{dom}(T)$, then add it and set $T(\alpha \cdot \vec{t}) = \Psi$ for some $\Psi \in \Gamma$ such that $S_{T(\alpha)}(\vec{t}) \subseteq \Psi$. Note that such a Ψ must exist by Condition 1 of Definition 4.6. Let $T_{\langle \psi_i, \Psi_j \rangle}$ be the result of augmenting T in this way.

- If $\psi_i \notin \Psi_j$, then let $T_{\langle \psi_i, \Psi_j \rangle}$ be the tree comprised of the nodes $\{\varepsilon\} \cup [k/n]$ such that $T_{\langle \psi_i, \Psi_j \rangle}(\varepsilon) = \Psi_j$ and, for each $\vec{t} \in [k/n]$, $T_{\langle \psi_i, \Psi_j \rangle}(\vec{t}) = \Psi$ for some $\Psi \in \Gamma$ with $S_{\Psi_j}(\vec{t}) \subseteq \Psi$.

It is easy to see that all trees $T_{\langle \psi_i, \Psi_j \rangle}$ are $\bigcirc$-matching. To construct a model of φ, intuitively we do the following: we arrange the selected witness trees in an $\ell \times m$-matrix such that the rows range over the eventualities $\psi_0, \ldots, \psi_{\ell-1}$ and the columns over the types $\Psi_0, \ldots, \Psi_{m-1}$, and then replace all leaf nodes by an 'arrow' from the leaf node's predecessor to the root of some other witness tree.

We now define the ATS $\mathcal{S} = (\Pi, \Sigma, Q, \pi, \delta)$ that we then prove to be a model of φ. Π and Σ are the sets of those propositions and agents that occur in the input formula φ. For defining the set of states Q, fix symbols $\varepsilon_{i,j}$ with $i \leq \ell$ and $j \leq m$. Then set:

$$Q := \{\varepsilon_{i,j} w \mid w \in \mathsf{dom}(T_{\langle \psi_i, \Psi_j \rangle}) \text{ is inner node}\}.$$

Next, the valuation π is easily defined: for $q = \varepsilon_{i,j} w \in Q$, set

$$\pi(q) := T_{\langle \psi_i, \Psi_j \rangle}(w) \cap \Pi.$$

To define the transition function δ, we first define a successor function on Q: for each $q = \varepsilon_{i,j} w \in Q$ and each $\vec{t} \in [k/n]$, set

$$s_{\vec{t}}(q) := \begin{cases} \varepsilon_{s,p} & \text{if } w \cdot \vec{t} \text{ is a leaf node of } T_{\langle \psi_i, \Psi_j \rangle}, \\ & s = i + 1 \bmod \ell, \text{ and } T_{\langle \psi_i, \Psi_j \rangle}(w \cdot \vec{t}) = \Psi_p \\ q \cdot t & \text{if } w \cdot \vec{t} \text{ is an inner node of } T_{\langle \psi_i, \Psi_j \rangle} \end{cases}$$

Now the definition of δ is straightforward: for each $q \in Q$ and $a \in \Sigma$, set

$$\delta(q, a) := \{\{s_{\vec{t}}(q) \mid \vec{t} = (t_0, \ldots, t_{n-1}) \in [k/n] \text{ and } t_a = p\} \mid p < k/2\}.$$

To show that $\mathcal{S}$ is indeed a model of φ, we introduce some auxiliary notions:

For each strategy $F_A = \{f_a \mid a \in A\}$ for a set of agents $A \subseteq \Sigma$ and each sequence of states $\lambda \in Q^+$, we write $F_A(\lambda)$ to denote the set of states $\bigcap_{a \in A} f_a(\lambda)$. Observe that, by definition of strategies for single agents, we have $F_A(\lambda \cdot q) \in \delta(q, A)$ for all $\lambda \in Q^+$ and $q \in Q$.

For each positive next-formula $\langle\!\langle A \rangle\!\rangle \bigcirc \psi$, the *$\langle\!\langle A \rangle\!\rangle \bigcirc \psi$-strategy* is the strategy $F_A = \{f_a \mid a \in A\}$ for the set of agents A that is defined by setting, for each $a \in A$,

$$f_a(\lambda \cdot q) := \{s_{\vec{t}}(q) \mid \vec{t} = (t_0, \ldots, t_{n-1}) \text{ and } t_a = \sharp_{\langle\!\langle A \rangle\!\rangle \bigcirc \psi}\}.$$

It is readily checked that we have

$$F_A(\lambda \cdot q) = \{s_{\vec{t}}(q) \mid \vec{t} \text{ is a } \langle\!\langle A\rangle\!\rangle\bigcirc\psi\text{-vector}\}. \qquad (*)$$

For each negative next-formula $\neg\langle\!\langle A\rangle\!\rangle\bigcirc\psi$, a *$\neg\langle\!\langle A\rangle\!\rangle\bigcirc\psi$-computation for a strategy F_A rooted at a state $q \in Q$* is a computation $\lambda \in \mathsf{out}(q, F_A)$ such that, for all positions $i \geq 0$, $\lambda[i+1] = s_{\vec{t}}(\lambda[i])$ for some $\neg\langle\!\langle A\rangle\!\rangle\bigcirc\psi$-vector $\vec{t} \in [k/n]$.

Claim 1 Let $\neg\langle\!\langle A\rangle\!\rangle\bigcirc\psi$ be a next-formula, F_A a strategy, and $q \in Q$. Then there exists a $\neg\langle\!\langle A\rangle\!\rangle\bigcirc\psi$-computation for F_A rooted at q.

Proof of Claim Let $\neg\langle\!\langle A\rangle\!\rangle\bigcirc\psi$, F_A, and q be as in the claim. Inductively define a $\neg\langle\!\langle A\rangle\!\rangle\bigcirc\psi$-computation $\lambda \in Q^\omega$ for F_A rooted at q as follows:

- $\lambda[0] := q$, and
- for each $i \geq 0$, $\lambda[i+1] := s_{\vec{t}}(\lambda[i])$ for some $\neg\langle\!\langle A\rangle\!\rangle\bigcirc\psi$-vector $\vec{t} \in [k/n]$ such that $s_{\vec{t}}(\lambda[i]) \in F_A(\lambda[i])$.

In order to show that λ is well-defined, it remains to show that for each $i \geq 0$, there is a state $s_{\vec{t}}(\lambda[i]) \in F_A(\lambda[i])$ such that $\vec{t}$ is a $\neg\langle\!\langle A\rangle\!\rangle\bigcirc\psi$-vector. Distinguish two cases:

- $A = \Sigma$. By definition, every vector $\vec{t} \in [k/n]$ is a $\neg\langle\!\langle \Sigma\rangle\!\rangle\bigcirc\psi$-vector. Since $F_A(\lambda[i]) \in \delta(\lambda[i], A)$, $F_A(\lambda[i])$ is non-empty. Thus, any vector from $F_A(\lambda[i])$ is suitable.
- $A \neq \Sigma$. By definition of δ and since $F_A(\lambda[i]) \in \delta(\lambda[i], A)$, $F_A(\lambda[i]) = \{s_{\vec{t}}(\lambda[i]) \mid \vec{t} \in S\}$ for some voting set S. By condition 1 of the function f used in the definition of $\neg\langle\!\langle A\rangle\!\rangle\bigcirc\psi$-vectors, S contains a $\neg\langle\!\langle A\rangle\!\rangle\bigcirc\psi$-vector. Thus, there is a state $s_{\vec{t}}(\lambda[i]) \in F_A(\lambda[i])$ such that $\vec{t}$ is a $\neg\langle\!\langle A\rangle\!\rangle\bigcirc\psi$-vector.

◀

To denote the intended type of a state $q = \varepsilon_{i,j} w \in Q$, we set $t(q) := T_{\langle\psi_i,\Psi_j\rangle}(w)$. Using the construction of $\mathcal{S}$ and property 2 of witness trees, it is straightforward to prove the following claim, which, intuitively, states that our ATS is $\bigcirc$-matching:

Claim 2 For all $q \in Q$ and $\vec{t} \in [k/n]$, $S_{t(q)}(\vec{t}) \subseteq t(s_{\vec{t}}(q))$.

The next claim establishes the property of $\mathcal{S}$ that is crucial for showing that it is a model of φ.

Claim 3 For any state $q \in Q$ and any formula $\psi \in \mathsf{ecl}(\varphi)$, $\psi \in t(q)$ iff $\mathcal{S}, q \models \psi$.

Proof of Claim Let q and ψ be as in the claim. The proof is by induction on the structure of ψ. Since the base case and the Boolean cases are straightforward, we concentrate on path quantifiers:

- $\psi = \langle\langle A\rangle\rangle\bigcirc\psi'$. "$\Rightarrow$": Suppose $\langle\langle A\rangle\rangle\bigcirc\psi' \in t(q)$. Let F_A be the $\langle\langle A\rangle\rangle\bigcirc\psi'$-strategy and $\lambda \in \mathsf{out}(q, F_A)$ a computation. By $(*)$, $\lambda[1] = s_{\vec{t}}(\lambda[0])$ for some $\langle\langle A\rangle\rangle\bigcirc\psi'$-vector $\vec{t}$. From $\langle\langle A\rangle\rangle\bigcirc\psi' \in t(\lambda[0])$ and Claim 2, it follows that $\psi' \in t(\lambda[1])$. The induction hypothesis yields $\mathcal{S}, \lambda[1] \models \psi'$. Hence, $\mathcal{S}, q \models \langle\langle A\rangle\rangle\bigcirc\psi'$ by the semantics.

  "$\Leftarrow$": Suppose $\langle\langle A\rangle\rangle\bigcirc\psi' \notin t(q)$. Then $\neg\langle\langle A\rangle\rangle\bigcirc\psi' \in t(q)$ by (T2). Let F_A be any strategy for the agents in the coalition A and λ a $\neg\langle\langle A\rangle\rangle\bigcirc\psi'$-computation for F_A rooted at q. Note that by Claim 1 there is such a λ. Since $\lambda[1] = s_{\vec{t}}(\lambda[0])$ for some $\neg\langle\langle A\rangle\rangle\bigcirc\psi'$-vector $\vec{t}$, $\neg\psi' \in t(\lambda[1])$ by Claim 2. Condition (T2) yields $\psi' \notin t(\lambda[1])$. Thus $\mathcal{S}, \lambda[1] \not\models \psi'$ by the induction hypothesis. Hence, $\mathcal{S}, q \not\models \langle\langle A\rangle\rangle\bigcirc\psi'$ by the semantics.

- $\psi = \langle\langle A\rangle\rangle\Box\psi'$. "$\Rightarrow$": Suppose $\langle\langle A\rangle\rangle\Box\psi' \in t(q)$. Let F_A be the $\langle\langle A\rangle\rangle\bigcirc\langle\langle A\rangle\rangle\Box\psi'$-strategy and $\lambda \in \mathsf{out}(q, F_A)$ a computation. We show by induction on i that, for $i \geq 0$, the following holds:

  (1) $\langle\langle A\rangle\rangle\Box\psi' \in t(\lambda[i])$;

  (2) $\langle\langle A\rangle\rangle\bigcirc\langle\langle A\rangle\rangle\Box\psi' \in t(\lambda[i])$

  (3) $\psi' \in t(\lambda[i])$.

  For the base case, (1) is immediate. Thus, (T3) yields $\langle\langle A\rangle\rangle\bigcirc\langle\langle A\rangle\rangle\Box\psi' \in t(\lambda[0])$ and $\psi' \in t(\lambda[0])$. For the induction step, the induction hypothesis gives us $\langle\langle A\rangle\rangle\bigcirc\langle\langle A\rangle\rangle\Box\psi' \in t(\lambda[i-1])$. By $(*)$, $\lambda[i] = s_{\vec{t}}(\lambda[i-1])$ for some $\langle\langle A\rangle\rangle\bigcirc\langle\langle A\rangle\rangle\Box\psi'$-vector $\vec{t}$. By Claim 2, it follows that $\langle\langle A\rangle\rangle\Box\psi' \in t(\lambda[i])$. Now we may again use (T3) to infer $\langle\langle A\rangle\rangle\bigcirc\langle\langle A\rangle\rangle\Box\psi' \in t(\lambda[i])$ and $\psi' \in t(\lambda[i])$. This finishes the induction. Finally, (3) and the induction hypothesis yield $\mathcal{S}, \lambda[i] \models \psi'$ for all $i \geq 0$ and we are done.

  "$\Leftarrow$": Suppose $\langle\langle A\rangle\rangle\Box\psi' \notin t(q)$. Then $\neg\langle\langle A\rangle\rangle\Box\psi' \in t(q)$ by (T2). Let F_A be any strategy for the agents in A and λ a $\neg\langle\langle A\rangle\rangle\bigcirc\langle\langle A\rangle\rangle\Box\psi'$-computation for F_A rooted at q. In the following, it is shown that $\neg\psi' \in t(\lambda[i])$ for some $i \geq 0$. Then (T2) yields $\psi' \notin t(\lambda[i])$ and the induction hypothesis $\mathcal{S}, \lambda[i] \not\models \psi'$ Hence, $\mathcal{S}, q \not\models \langle\langle A\rangle\rangle\Box\psi'$ by the semantics as desired.

  Suppose by contradiction that $\neg\psi' \notin t(\lambda[i])$ for all $i \geq 0$. By (T2), $\psi' \in t(\lambda[i])$ for all $i \geq 0$. We show by induction on i that, for $i \geq 0$, the following holds as well:

  (1) $\neg\langle\langle A\rangle\rangle\Box\psi' \in t(\lambda[i])$;

(2) $\neg\langle\!\langle A\rangle\!\rangle\bigcirc\langle\!\langle A\rangle\!\rangle\Box\psi' \in t(\lambda[i])$

For the base case, (1) has already been shown. Since $\psi' \in t(\lambda(0))$, (T3) and (1) imply $\langle\!\langle A\rangle\!\rangle\bigcirc\langle\!\langle A\rangle\!\rangle\Box\psi' \notin t(\lambda[i])$. Thus, (2) follows by (T2). For the induction step, the induction hypothesis gives us $\neg\langle\!\langle A\rangle\!\rangle\bigcirc\langle\!\langle A\rangle\!\rangle\Box\psi' \in t(\lambda[i-1])$. By definition of λ, $\lambda[i] = s_{\vec{t}}(\lambda[i-1])$ for some $\neg\langle\!\langle A\rangle\!\rangle\bigcirc\langle\!\langle A\rangle\!\rangle\Box\psi'$-vector $\vec{t}$. By Claim 2, we thus have $\neg\langle\!\langle A\rangle\!\rangle\Box\psi' \in t(\lambda[i])$. To establish (2), we may argue as in the base case.

By the matrix construction of $\mathcal{S}$, there is a position $i \geq 0$ such that $\tau(\lambda[i])$ is the root of the φ-tree $T_{\langle\vartheta,\Psi\rangle}$ where $\vartheta = \neg\langle\!\langle A\rangle\!\rangle\Box\psi'$ and $\Psi = t(\lambda[i])$. Since $\neg\langle\!\langle A\rangle\!\rangle\Box\psi' \in \Psi$ by Point 1 above, $T_{\langle\vartheta,\Psi\rangle}$ is a witness tree for the eventuality $\neg\langle\!\langle A\rangle\!\rangle\Box\psi'$ rooted at Ψ in Γ. The finiteness of $T_{\langle\vartheta,\Psi\rangle}$ implies that there is a $j \geq i$ such that the type $t(\lambda[j])$ labels one of its leaf nodes $\tau(\lambda[j])$. Hence, $\neg\psi' \in t(\lambda[j])$ by definition of the witness tree $T_{\langle\vartheta,\Psi\rangle}$; a contradiction.

- $\psi = \langle\!\langle A\rangle\!\rangle\psi\,\mathcal{U}\,\vartheta$. This case is similar to the previous one and left to the reader.

◀

Since $\varphi \in \Psi$ for some type $\Psi \in \Gamma$, there is a state $q \in Q$ such that $\psi \in t(q)$. Then it follows from Claim 3 that $\mathcal{S}, q \models \varphi$.

"$\Leftarrow$" (Completeness): Suppose φ is satisfiable in an ATS $\mathcal{S} = \langle \Pi, \Sigma, Q, \pi, \delta\rangle$ in a state $q_\varphi \in Q$. For each state $q \in Q$, let $t(q)$ be the type $\{\psi \in \mathsf{ecl}(\varphi) \mid \mathcal{S}, q \models \psi\}$. Denote with $\mathsf{types}(Q)$ the set of all types associated with some state in Q. We first establish the following claim:

Claim 4 Let $q \in Q$, $\vec{t} \in [k/n]$. Then there is a state $q' \in Q$ such that $S_{\vec{t}}(q) \subseteq t(q')$. Moreover, the following holds: if $\langle\!\langle A\rangle\!\rangle\bigcirc\psi \in t(q)$ and F_A is a strategy such that, for all computations $\lambda \in \mathsf{out}(q, F_A)$ we have $\mathcal{S}, \lambda[1] \models \psi$, we can choose q' such that $q' \in F_A(q)$.

Proof of Claim Let q and $\vec{t}$ be as in the claim. Also, select a formula $\langle\!\langle A\rangle\!\rangle\bigcirc\psi$ and a strategy F_A as in the "moreover" part of the claim. Note first that, by Property 3 of the function f used in the definition of vectors for negative next-formulas, $\vec{t}$ is a vector for at most a single formula $\neg\langle\!\langle A'\rangle\!\rangle\bigcirc\psi'$ with $A' \subset \Sigma_\varphi$. Let

- $\langle\!\langle A_1\rangle\!\rangle\bigcirc\psi_1, \ldots, \langle\!\langle A_\ell\rangle\!\rangle\bigcirc\psi_\ell$ be all positive next-formulas from $t(q)$ for which $\vec{t}$ is a vector; this includes the selected formula $\langle\!\langle A\rangle\!\rangle\bigcirc\psi$;
- $\neg\langle\!\langle A'\rangle\!\rangle\bigcirc\psi'$ be the single negative next-formula from $t(q)$ with $A' \subset \Sigma_\varphi$ for which $\vec{t}$ is a vector, if such a formula exists;
- $\neg\langle\!\langle \Sigma_\varphi\rangle\!\rangle\bigcirc\psi''_1, \ldots, \neg\langle\!\langle \Sigma_\varphi\rangle\!\rangle\bigcirc\psi''_m$ be all negative next-formulas from $t(q)$ quantifying over the set of all agents Σ_φ.

Observe that, by definition, $\vec{t}$ is a vector for all negative next-formulas quantifying over Σ_φ, so that $\psi''_1, \dots, \psi''_m \in S_{t(q)}(\vec{t})$. Next note that, by definition of vectors for positive next-formulas, we have $A_i \cap A_j = \emptyset$ for $1 \le i < j \le \ell$.

For $1 \le i \le \ell$, let F_{A_i} be a strategy such that for all computations $\lambda \in \mathsf{out}(q, F_{A_i})$, $\mathcal{S}, \lambda[1] \models \psi_i$. Such a strategy exists since $\langle\!\langle A_i \rangle\!\rangle \bigcirc \psi_i \in t(q)$. For the selected formula $\langle\!\langle A \rangle\!\rangle \bigcirc \psi$ from the "moreover" part of the claim, choose the selected strategy F_A. Let $B = \bigcup_{1 \le i \le \ell}$ and set $F_B = \bigcup_{1 \le i \le \ell} F_{A_i}$ which is well-defined since $A_i \cap A_j = \emptyset$ for $1 \le i < j \le \ell$. Thus for all $\lambda \in \mathsf{out}(q, F_B)$, we have $\mathcal{S}, \lambda[1] \models \psi_i$ for $1 \le i \le \ell$.

Next, select a computation $\lambda \in \mathsf{out}(q, F_B)$. If there is no negative next-formula in $t(q)$, for which $\vec{t}$ is a vector, then choose an arbitrary element $\lambda \in \mathsf{out}(q, F_B)$ ($\mathsf{out}(q, F_B)$ is non-empty since $\delta(q, A)$ is non-empty for all q and A). Otherwise, choose a $\lambda \in \mathsf{out}(q, F_B)$ such that $\mathcal{S}, \lambda[1] \models \neg\psi'$. Such an element exists since, first, $\neg\langle\!\langle A' \rangle\!\rangle \bigcirc \psi'$ is in $t(q)$ and, second, $\vec{t}$ being a vector for this formula implies (by condition 2 of the definition of such vectors) that $A_i \subseteq A'$ for $1 \le i \le \ell$.

Finally, we have $\mathcal{S}, \lambda[1] \models \neg\psi''_i$ for $1 \le i \le m$ since $\neg\langle\!\langle \Sigma_\varphi \rangle\!\rangle \bigcirc \psi''_i \in t(q)$ implies that $\mathcal{S}, \lambda'[1] \models \neg\psi''_i$ for any computation λ' rooted at q.

Summing up, we have shown that $S_{t(q)}(\vec{t}) \subseteq t(\lambda[1]) \in \mathsf{types}(Q)$. Thus, $\lambda[1]$ is the state whose existence is stated in the claim. ◀

In the following, it is shown that all types in $\mathsf{types}(Q)$ are realizable in $\mathsf{types}(Q)$. Let $q \in Q$ be a state. We have to check that each type $t(q)$ in $\mathsf{types}(Q)$ satisfies conditions 1 to 3 of Definition 4.6.

1. Let $\vec{t} \in [k/n]$. We have to show that $S_{t(q)}(\vec{t}) \subseteq \Psi$ for some $\Psi \in \mathsf{types}(Q)$. Clearly, this is an immediate consequence of Claim 4 (the "moreover" part is not needed).

2. Suppose $\langle\!\langle A \rangle\!\rangle \psi \,\mathcal{U}\, \vartheta \in t(q)$. It is our aim to construct a $\langle\!\langle A \rangle\!\rangle \psi \,\mathcal{U}\, \vartheta$-witness tree rooted at the type $t(q)$ in $\mathsf{types}(Q)$. Since $\mathcal{S}, q \models \langle\!\langle A \rangle\!\rangle \psi \,\mathcal{U}\, \vartheta$, there is a strategy F_A such that for all computations $\lambda \in \mathsf{out}(q, F_A)$, there is a position $i \ge 0$ such that $\mathcal{S}, \lambda[i] \models \vartheta$ and $\mathcal{S}, \lambda[j] \models \psi$ for $j < i$.

   Using the semantics, it is not difficult to prove that F_A satisfies the following property: if $\lambda \in \mathsf{out}(q, F_A)$ and $i \in \mathbb{N}$ is smallest such that $\mathcal{S}, \lambda[i] \models \vartheta$, then

   (a) $\mathcal{S}, \lambda[j] \models \langle\!\langle A \rangle\!\rangle \psi \,\mathcal{U}\, \vartheta$ for $j \le i$;
   (b) $\mathcal{S}, \lambda[j] \models \langle\!\langle A \rangle\!\rangle \bigcirc \langle\!\langle A \rangle\!\rangle \psi \,\mathcal{U}\, \vartheta$ for $j < i$;
   (c) $\mathcal{S}, \lambda[j] \models \psi$ for $j < i$.

   We use F_A to define a $\langle Q, k, n \rangle$-tree T. This tree can then easily be converted into the required witness-tree. For a member α of $[k/n]^*$ denote by $\alpha[0, i]$ the initial segment of α of length $i+1$. In particular, α coincides with $\alpha[0, |\alpha|-1]$, where $|\alpha|$ denotes the length of α. Now, the construction proceeds by induction as follows: In the induction start, set $T(\varepsilon) := q$. In the induction step, let $\alpha \in \mathsf{dom}(T)$ be such such that $T(\alpha)$ is already defined. If $\mathcal{S}, T(\alpha) \models \vartheta$, then α is a leaf and we do not further extend this branch. Otherwise, for each $\langle\langle A\rangle\rangle\bigcirc\langle\langle A\rangle\rangle\psi\,\mathcal{U}\,\vartheta$-vector $\vec{t}$, set $T(\alpha \cdot \vec{t}) := q'$ for some $q' \in Q$ such that

(i) $S_{t(T(\alpha))}(\vec{t}) \subseteq t(q')$, and

(ii) $q' \in F_A(\lambda)$ where $\lambda = T(\alpha[0])\, T(\alpha[0]\alpha[1], 1]) \cdots T(\alpha[0, |\alpha| - 1])$.

The existence of such a q' is a consequence of Claim 4: since $\lambda = \lambda' \cdot T(\alpha)$ for some λ', there exists a strategy F'_A such that $F'_A(T(\alpha)) = F_A(\lambda)$. Take any such strategy. Now apply Claim 4 including the "moreover" part, using the next-formula $\langle\langle A\rangle\rangle\bigcirc\langle\langle A\rangle\rangle\psi\,\mathcal{U}\,\vartheta$ and the strategy F'_A, to get the desired state q'. Note that the prerequisites of the "moreover" part are satisfied:

– $\mathcal{S}, T(\alpha) \models \langle\langle A\rangle\rangle\bigcirc\langle\langle A\rangle\rangle\psi\,\mathcal{U}\,\vartheta$ holds by (b).

– for all computations $\lambda \in \mathsf{out}(\alpha, F'_A)$, we have $\mathcal{S}, q \models \langle\langle A\rangle\rangle\psi\,\mathcal{U}\,\vartheta$ since F'_A is based on F_A and by (a).

We now show that T is finite. Suppose by contradiction that there is an infinite path $\tau \in [k/n]^\omega$ in T. Let $\lambda \in Q^\omega$ be the infinite sequence defined by setting $\lambda[i] := T(\tau[0, i])$. By (ii), $\lambda \in \mathsf{out}(q, F_A)$. Then there is a position $i \geq 0$ such that $\mathcal{S}, \lambda[i] \models \vartheta$. Thus $\vartheta \in t(\lambda[i]) = t(T(\tau[0, i]))$ and the node $\tau[0, i+1]$ is a leaf; a contradiction.

Since the nodes in T are labelled by states, the composition $t(T(\cdot))$ yields a finite φ-tree. To show that $t(T(\cdot))$ is a $\langle\langle A\rangle\rangle\psi\,\mathcal{U}\,\vartheta$-witness tree rooted at $t(q)$ in $\mathsf{types}(Q)$, it remains to show that T satisfies properties 1 to 7 in the definition of a $\langle\langle A\rangle\rangle\psi\,\mathcal{U}\,\vartheta$-witness tree: properties 1, 3, 6, and 7 are immediate by definition of T; property 2 is a consequence of (i), property (4) a consequence of (a), and property 5 a consequence of (c).

3. This case is similar to the previous one and left to the reader.

From $\mathcal{S}, q_\varphi \models \varphi$ it follows that $\varphi \in t(q_\varphi)$. Then $\mathsf{types}(Q)$ is a set of types that are each realizable in $\mathsf{types}(Q)$ and $t(q_\varphi)$ is a type in $\mathsf{types}(Q)$ such that $\varphi \in t(q_\varphi)$. Let Δ be the set of types for φ computed by the type elimination algorithm. It is easy to see that $\mathsf{types}(Q) \subseteq \Delta$. Hence, the algorithm returns "Yes, the input formula φ is satisfiable". QED

Lemma 4.9 *The described elimination procedure runs in exponential time.*

Proof. Suppose φ is given and let $n = |\varphi|$. Recall that the size of the extended closure $\mathsf{ecl}(\varphi)$ is linear in the length of φ, i.e., $|\mathsf{ecl}(\varphi)| = c \cdot n$ for some constant $c \geq 1$. The algorithm computes a sequence $\Delta_0, \ldots, \Delta_m$ of sets of types such that

$\Delta_0 \supsetneq \Delta_1 \supsetneq \cdots \supsetneq \Delta_m$. Since $\Delta_0 = \Gamma_\varphi \subseteq 2^{\mathsf{ecl}(\varphi)}$, this sequence is finite with $m \leq 2^{c \cdot n}$. For each i with $0 \leq i < m$, it holds that $|\Delta_i| < |\Delta_0| \leq 2^{c \cdot n}$. Thus, to compute the set Δ_{i+1} at most $2^{c \cdot n}$ types in Δ_i need to be checked whether they are realizable in Δ_i. In the following, it is shown that for a type $\Psi \in \Delta_i$ at most $2^{\mathcal{O}(n^2)}$ steps are needed to check for Ψ's realizability in Δ_i. Consider the 3 points in Definition 4.6:

1. For at most for n^n vectors (as $k \leq n$), inclusion tests for maximal $2^{c \cdot n}$ types in Δ_i have to be performed. Hence, this takes not more than $n^n \cdot 2^{c \cdot n} = 2^{\mathcal{O}(n^2)}$ steps.

2.,3. By Lemma 4.7, to decide the existence of witness trees takes not more than $2^{\mathcal{O}(n^2)}$ steps. This has to be done for at most n formulas of the form $\langle\langle A \rangle\rangle \psi \,\mathcal{U}\, \vartheta$ or $\neg\langle\langle A \rangle\rangle \Box \psi$ in Ψ. Thus, altogether maximal $2^{\mathcal{O}(n^2)}$ steps are needed.

Note that checking whether there is a type in Δ_m that contains φ takes not more than $2^{\mathcal{O}(n)}$ steps. We conclude that our decision procedure runs in time exponential in the size of the input. QED

Let us now consider the lower bound. By Lemma 3.2, it is sufficient to prove ExpTime-hardness for variant (b) of the ATL satisfiability problem. For this variant, ExpTime-hardness does *not* trivially follow from ExpTime-hardness of CTL: the translation from CTL to ATL described at the end of Section 2 cannot be used since, when concerned with satisfiability over arbitrary sets of agents, there is no obvious ATL-equivalent of CTL-formulas of the form $\mathsf{E}\varphi \,\mathcal{U}\, \psi$. Note in particular that we cannot use $\langle\langle \Sigma \rangle\rangle \varphi \,\mathcal{U}\, \psi$ since the coalition Σ of all agents is not available for a formula. Moreover, the $\langle\langle \Sigma \rangle\rangle \varphi \,\mathcal{U}\, \psi$ equivalent formula $\neg\langle\langle \emptyset \rangle\rangle \neg(\varphi \,\mathcal{U}\, \psi)$ is not according to the syntax of ATL.

To show the lower bound for ATL satisfiability over arbitrary sets of agents, we reduce the ExpTime-hard global consequence problem in the modal logic K. We refer to, e.g., [13] for the syntax and semantics of K. To avoid confusion with the operators of ATL, we denote the diamond and box of K with $\blacklozenge$ and $\blacksquare$. Recall that the global consequence problem is to decide, given two K-formulas φ and ψ, whether it is the case that for every Kripke structure $\mathcal{M}$, if φ is true in every state of $\mathcal{M}$ (in symbols, $\mathcal{M} \models \varphi$), then ψ is true in every state of $\mathcal{M}$.

For the reduction, we use the following facts: (i) a Kripke structure $\mathcal{M}$ is equivalent to an alternating transition system with a single agent [3, 7]; (ii) $\langle\langle \emptyset \rangle\rangle \bigcirc \varphi^\sharp$ expresses that φ is true at all successors, and (iii) $\langle\langle \emptyset \rangle\rangle \Box \varphi$ expresses that φ holds in the (reachable part of the) whole model. Now let φ and ψ be K-formulas. Translate φ and ψ into formulas of ATL using the following translation $(\cdot)^\sharp$:

$$\begin{array}{rcl}
p^\sharp & = & p \\
(\varphi \wedge \psi)^\sharp & = & \varphi^\sharp \wedge \psi^\sharp \\
(\neg\varphi)^\sharp & = & \neg\varphi^\sharp \\
(\blacklozenge\varphi)^\sharp & = & \neg\langle\langle \emptyset \rangle\rangle \bigcirc \neg\varphi^\sharp \\
(\blacksquare\varphi)^\sharp & = & \langle\langle \emptyset \rangle\rangle \bigcirc \varphi^\sharp.
\end{array}$$

Now it suffices to show the following:

Lemma 4.10 *ψ follows globally from φ iff $\langle\langle\emptyset\rangle\rangle\Box\varphi^\sharp \wedge \neg\psi^\sharp$ is unsatisfiable in ATL with an arbitrary set of agents.*

Proof. It is straightforward to prove by structural induction that, for all Kripke structures $\mathcal{M}$, all states q of $\mathcal{M}$, and all K-formulas ϑ, we have $\mathcal{M}, q \models \vartheta$ iff $\mathcal{M}, q \models \vartheta^\sharp$, where in the latter case $\mathcal{M}$ is viewed as a single-agent ATS. We leave details to the reader and continue with a proof of the lemma.

"if". We show the contrapositive. Thus suppose ψ does not globally follow from φ. Then there is a Kripke structure $\mathcal{M}$ such that $\mathcal{M} \models \varphi$ and $\mathcal{M}, q \not\models \psi$ for some state q of $\mathcal{M}$. Then, $\mathcal{M}, q \models \langle\langle\emptyset\rangle\rangle\Box\varphi^\sharp$ and $\mathcal{M}, q \not\models \psi^\sharp$. Hence, $\mathcal{M} \models \langle\langle\emptyset\rangle\rangle\Box\varphi^\sharp \wedge \neg\psi^\sharp$, and this formula is satisfiable in a single-agent ATS.

"only if".We again show the contrapositive. Suppose $\langle\langle\emptyset\rangle\rangle\Box\varphi^\sharp \wedge \neg\psi^\sharp$ is satisfiable and take an ATS $\mathcal{M}$ and a state q of $\mathcal{M}$ such that $\mathcal{M}, q \models \langle\langle\emptyset\rangle\rangle\Box\varphi^\sharp \wedge \neg\psi^\sharp$. By Lemma 3.2 and since $\langle\langle\emptyset\rangle\rangle\Box\varphi^\sharp \wedge \neg\psi^\sharp$ does not refer to any agents, we may assume that $\mathcal{M}$ is a single-agent ATS and thus a Kripke structure. We have $\mathcal{M}, q \models \langle\langle\emptyset\rangle\rangle\Box\varphi^\sharp$ and $\mathcal{M}, q \not\models \psi^\sharp$. By the former, $\varphi^\sharp$ holds at all points reachable from q. Denote by $\mathcal{N}$ the model induced by $\mathcal{M}$ on those points. Then $\mathcal{N} \models \varphi$ but $\mathcal{N}, q \not\models \psi$. Hence, ψ does not follow globally from φ. QED

5 Conclusions

We have revisited the satisfiability problem for ATL, the Alternating-time Temporal Logic of Alur, Henzinger, and Kupferman, which was settled at ExpTime-complete in a result of van Drimmelen for cases where the set of agents was fixed externally in advance. We pointed out that if the set of agents is not fixed externally, then van Drimmelen's construction yielded only a 2ExpTime upper bound. This motivated the statement of three variations of the ATL satisfiability problem, where the set of agents was not fixed externally in advance. Our main result was to prove that each of these variations is ExpTime-complete.

Also of interest is the complexity of satisfiability for epistemic and other extensions of ATL, such as ATEL [10]. In fact, [15] uses a combination of the technique introduced in this paper with the technique of [9] to show that the simplest variety of ATEL from [10] with operators for common and distributed knowledge is ExpTime-complete as well.

For future work, it would be interesting to consider systems in which richer interactions between knowledge and ability occur. Another significant issue to address is the complexity of ATL* satisfiability. Given that ATL* subsumes CTL*, it follows ATL* satisfiability has a 2ExpTime lower bound. Is ATL* 2ExpTime complete?

Acknowledgments: We are indepted to Wiebe van der Hoek for fruitful discussions about ATL, and to the anonymous reviewers for their comments that helped to improve the presentation of our results. We gratefully acknowledge the support of the EPSRC under research grant GR/S62727/01 ("Virtual Organisations for e-Science"). The work reported here was partly carried out while the second author was visiting the University of Liverpool on a DAAD grant.

References

[1] R. Alur, T. A. Henzinger, and O. Kupferman. Alternating-time temporal logic. In *Proceedings of the 38th IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 100–109, Florida, October 1997.

[2] R. Alur, T. A. Henzinger, and O. Kupferman. Alternating-time temporal logic. *Lecture Notes in Computer Science*, 1536:23–60, 1998.

[3] R. Alur, T. A. Henzinger, and O. Kupferman. Alternating-time temporal logic. *J. ACM*, 49(5):672–713, 2002.

[4] R. Alur, T. A. Henzinger, F. Y. C. Mang, S. Qadeer, S. K. Rajamani, and S. Taşiran. Mocha: Modularity in model checking. In *CAV 1998: Tenth International Conference on Computer-aided Verification, (LNCS Volume 1427)*, pages 521–525. Springer Verlag, 1998.

[5] E. A. Emerson. Temporal and modal logic. In *Handbook of theoretical computer science (vol. B): formal models and semantics*, pages 995–1072. MIT Press, 1990.

[6] E. A. Emerson and J. Y. Halpern. Decision procedures and expressiveness in the temporal logic of branching time. In *STOC '82: Proceedings of the fourteenth annual ACM symposium on Theory of computing*, pages 169–180. ACM Press, 1982.

[7] V. Goranko and W. Jamroga. Comparing semantics of logics for multi-agent systems. *Synthese*, 139(2):241–280, 2004.

[8] V. Goranko and G. van Drimmelen. Decidability and complete axiomatization of the alternating-time temporal logic. Theoretical Computer Science, to appear, 2005.

[9] J. Y. Halpern and Y. Moses. A guide to completeness and complexity for modal logics of knowledge and belief. *Artificial Intelligence*, 54(3):319–380, 1992.

[10] W. Hoek and M. Wooldridge. Time, knowledge, and cooperation: Alternating-time temporal epistemic logic and its applications. *Studia Logica*, 75(1):125–157, 2003.

[11] M. Pauly. *Logic for Social Software.* PhD thesis, University of Amsterdam, 2001. ILLC Dissertation Series 2001-10.

[12] M. Pauly and M. Wooldridge. Logic for mechanism design — a manifesto. In *Proceedings of the 2003 Workshop on Game Theory and Decision Theory in Agent Systems (GTDT-2003)*, Melbourne, Australia, 2003.

[13] E. Spaan. *Complexity of Modal Logics.* PhD thesis, Department of Mathematics and Computer Science, University of Amsterdam, 1993.

[14] G. van Drimmelen. Satisfiability in alternating-time temporal logic. In *18th IEEE Symposium on Logic in Computer Science (LICS 2003), 22-25 June 2003, Ottawa, Canada, Proceedings.* IEEE Computer Society, 2003.

[15] D. Walther. Satisfiability of ATEL with distributed knowledge is ExpTime-complete. Technical report, University of Liverpool, 2005.