Towards the Verification of Strategic Properties in Multi-Agent Systems with Imperfect Information

Angelo Ferrando
University of Genoa
Genoa, Italy
angelo.ferrando@unige.it

Vadim Malvone
Telecom Paris
Palaiseau, France
vadim.malvone@telecom-paris.fr

ABSTRACT

In logics for strategic reasoning the main challenge is represented by their verification in contexts of imperfect information and perfect recall strategies. In this work, we show a technique to approximate the verification of Alternating-time Temporal Logic (ATL*) under imperfect information and perfect recall, which is known to be undecidable. Given a model M and a formula φ, we propose a verification procedure that generates sub-models of M in which each sub-model M' satisfies a sub-formula φ' of φ and the verification of φ' in M' is decidable. Then, we use CTL* model checking to provide a verification result of φ on M. We prove that our procedure is sound and in the same complexity class of ATL* model checking under perfect information and perfect recall. Moreover, we present a tool that uses our procedure and provide experimental results.

KEYWORDS

Verification of Multi-Agent Systems; Alternating-time Temporal Logic; Imperfect Information; Model Checking

ACM Reference Format:
Angelo Ferrando and Vadim Malvone. 2023. Towards the Verification of Strategic Properties in Multi-Agent Systems with Imperfect Information. In *Proc. of the 22nd International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2023), London, United Kingdom, May 29 – June 2, 2023*, IFAAMAS, 9 pages.

1 INTRODUCTION

A well-known formalism for reasoning about strategic behaviours in Multi-Agent Systems (MAS) is Alternating-time Temporal Logic (ATL) [1]. An important quality of ATL is its model checking complexity, which is PTIME-complete under perfect information. However, MAS in general have imperfect information and the model checking for ATL specifications under imperfect information and perfect recall is undecidable [12]. Given the relevance of the imperfect information setting, even partial solutions to the problem can be useful. In this contribution the main idea is to modify the topological structure of the models and use CTL verification. With more detail, given a model M and a specification φ, our procedure generates a set of sub-models of M in which there is perfect information and each of these sub-models satisfies a sub-formula of φ. Two classes of sub-models are defined: negative and positive. In the verification of φ, the former class under-approximates M while the latter over-approximates M. That is, if a sub-formula is (resp., is not) satisfied in a negative (resp., positive) sub-model, where there are more (resp., less) limitations than M, then the sub-formula is (resp., is not) satisfied in M. Finally, the procedure substitutes the remaining not satisfied strategic operators with path operators of CTL* and through its model checking, it checks whether: (i) the universal remaining part of φ is satisfied, (ii) the existential remaining part of φ is not satisfied. In both cases, we provide a preservation result to ATL*. Since the original problem is undecidable, we can not guarantee that the truth or falsity of the property can be always established. In other words, we cannot provide the completeness of our solution. However, the procedure provides a constructive method to evaluate an ATL* formula under imperfect information and perfect recall. We also show how such a procedure is in the same complexity class of ATL* model checking under perfect information and perfect recall.

Structure of the work. In Section 2, we present the syntax of ATL* as well as its semantics.In Section 3, we present our running example, a variant of the Curiosity rover scenario. Subsequently, in Section 4, we show our verification procedure by presenting how to capture sub-models with perfect information that satisfy a sub-formula of the original formula and then how this procedure can be used together with CTL* model checking to have results. We also provide evidence of the correctness and complexity of our procedure. Furthermore, the latter is used to implement an extension of MCMAS (Model Checker for Multi-Agent Systems) that we discuss in Section 5. We conclude by recapping our results and pointing to future works. Due to the limited space, an extended version of this work can be found in [15].

Related work. Various approaches for the verification of specifications in ATL under imperfect information have been recently put forward. In one line, restrictions are made on how information is shared amongst the agents, so as to retain decidability [11]. In a related line, interactions amongst agents are limited to public actions only [7, 8]. These approaches are markedly different from ours as they seek to identify classes for which verification is decidable. Instead, we consider the whole class of models and define a general verification procedure. In this sense, our approach is related to [4, 6] where an abstraction method over the strategies is defined and to [3, 5, 9, 17] where an approximation to the information is presented. However, while in these works perfect recall or imperfect information are approximated, here we study the topological structure of the models and use CTL* verification to provide a result. While the orthogonality between our approach and [4, 6] is clear since they approximate over the memory while we approximate over the information, it could be useful for the reader to add further words about the relation with [3, 5, 9]. With more detail, in [3, 5, 9] an abstract model abstracts the imperfect information by using a state for each equivalence class, here we don't have an abstraction

Proc. of the 22nd International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2023), A. Ricci, W. Yeoh, N. Agmon, B. An (eds.), May 29 – June 2, 2023, London, United Kingdom.

on the states, that is each state remains as it is in the original model or it is removed. Additionally, while in [3, 5, 9] there are may/must transitions, here, for the sub-models, we have the same transitions of the original model but the ones related to equivalent states that are all connected with a special state. Thus, the two approaches are orthogonal. Finally, [14, 16] presents a tool that tries to give a result for ATL with imperfect information and perfect recall by using Runtime Verification, a well-know verification technique that is not exhaustive. So, the objectives of [14, 16] are orthogonal with ours, while in our work we want to be exhaustive by using CTL* model checking, they use the computational power of Runtime Verification to give some assumptions over formulas of interest.

2 PRELIMINARIES

In this section we recall some preliminary notions. Given a set U, $\overline{U}$ denotes its complement. We denote the length of a tuple v as $|v|$, its j-th element as v_j, and its last element $v_{|v|}$ as $last(v)$. For $j \leq |v|$, let $v_{\geq j}$ be the suffix $v_j, \ldots, v_{|v|}$ of v starting from v_j and $v_{\leq j}$ the prefix $v_1, \ldots, v_j$ of v.

2.1 Model

We start by showing a formal model for Multi-Agent Systems via concurrent game structures with imperfect information [1, 18].

Definition 2.1. A *concurrent game structure with imperfect information (iCGS)* is a tuple $M{=}\langle Ag, AP, S, s_I, \{Act_i\}_{i\in Ag}, \{\sim_i\}_{i\in Ag}, d, \delta, V\rangle$ such that:

- $Ag = \{1, \ldots, m\}$ is a nonempty finite set of agents.
- AP is a nonempty finite set of atomic propositions (atoms).
- $S \neq \emptyset$ is a finite set of *states*, with *initial state* $s_I \in S$.
- For every $i \in Ag$, Act_i is a nonempty finite set of *actions*. Let $Act = \bigcup_{i\in Ag} Act_i$ be the set of all actions, and $ACT = \prod_{i\in Ag} Act_i$ the set of all joint actions.
- For every $i \in Ag$, $\sim_i$ is a relation of *indistinguishability* between states. That is, given states $s, s' \in S$, $s \sim_i s'$ iff s and s' are indistinguishable for agent i.
- The *protocol function* $d : Ag \times S \to (2^{Act} \setminus \{\emptyset\})$ defines the availability of actions so that for every $i \in Ag$, $s \in S$, (i) $d(i, s) \subseteq Act_i$ and (ii) $s \sim_i s'$ implies $d(i, s) = d(i, s')$.
- The *transition function* $\delta : S \times ACT \to S$ assigns a successor state $s' = \delta(s, \vec{a})$ to each $s \in S$, for every joint action $\vec{a} \in ACT$ such that $a_i \in d(i, s)$ for every $i \in Ag$.
- $V : S \to 2^{AP}$ is the *labelling function*.

By Def. 2.1 an iCGS describes the interactions of a group Ag of agents, starting from the initial state $s_I \in S$, according to the transition function δ. The latter is constrained by the availability of actions to agents, as specified by the protocol function d. Furthermore, we assume that agents can have imperfect information over the game; so in any state s, agent i considers epistemically possible all states s' that are i-indistinguishable from s [13]. When every $\sim_i$ is the identity relation, *i.e.*, $s \sim_i s'$ iff $s = s'$, we obtain a standard CGS with perfect information [1]. A history $h \in S^+$ is a finite (non-empty) sequence of states. The indistinguishability relations are extended to histories in a synchronous, point-wise way, *i.e.*, histories $h, h' \in S^+$ are *indistinguishable* for agent $i \in Ag$, or $h \sim_i h'$, iff (i) $|h| = |h'|$ and (ii) for all $j \leq |h|$, $h_j \sim_i h'_j$.

2.2 Syntax

We use ATL* [1] to reason about the strategic abilities of agents.

Definition 2.2. State (φ) and path (ψ) formulas in ATL* are defined as follows:

$$\begin{array}{lcl} \varphi & ::= & q \mid \neg\varphi \mid \varphi \wedge \varphi \mid \langle\langle\Gamma\rangle\rangle\psi \\ \psi & ::= & \varphi \mid \neg\psi \mid \psi \wedge \psi \mid X\psi \mid (\psi U \psi) \end{array}$$

where $q \in AP$ and $\Gamma \subseteq Ag$.

Formulas in ATL* are all and only the state formulas.

As usual, a formula $\langle\langle\Gamma\rangle\rangle\Phi$ is read as "the agents in coalition Γ have a strategy to achieve Φ". The meaning of temporal operators *next* X and *until* U is standard [2]. Operators $[[\Gamma]]$, *release* R, *eventually* F, and *globally* G can be introduced as usual.

Formulas in the ATL fragment are obtained from Def. 2.2 by restricting path formulas as follows:

$$\psi \quad ::= \quad X\varphi \mid (\varphi U \varphi) \mid (\varphi R \varphi)$$

In the rest of the paper, we will also consider the syntax of ATL* in negation normal form (NNF):

$$\begin{array}{lcl} \varphi & ::= & q \mid \neg q \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \langle\langle\Gamma\rangle\rangle\psi \mid [[\Gamma]]\psi \\ \psi & ::= & \varphi \mid \psi \wedge \psi \mid \psi \vee \psi \mid X\psi \mid (\psi U \psi) \mid (\psi R \psi) \end{array}$$

where $q \in AP$ and $\Gamma \subseteq Ag$.

2.3 Semantics

We assume that agents employ *uniform strategies* [18], *i.e.*, they perform the same action whenever they have the same information.

Definition 2.3. A *uniform perfect recall strategy* for agent $i \in Ag$ is a function $\sigma_i : S^+ \to Act_i$ such that for all histories $h, h' \in S^+$, (i) $\sigma_i(h) \in d(i, last(h))$ and (ii) $h \sim_i h'$ implies $\sigma_i(h) = \sigma_i(h')$.

By Def. 2.3 any strategy for agent i has to return actions that are enabled for i. Also, whenever two histories are indistinguishable for i, then the same action is returned. Notice that, for the case of perfect information, condition (ii) is satisfied by any strategy σ. Furthermore, we obtain memoryless (or imperfect recall) strategies by considering the domain of σ_i in S, *i.e.* $\sigma_i : S \to Act_i$. Given an iCGS M, a *path* π is an infinite sequence of states. We denote with S^ω the set of paths over S. Given a joint strategy Σ_Γ, comprising of one strategy for each agent in coalition Γ, a path π is Σ_Γ-*compatible* iff for every $j \geq 1$, $\pi_{j+1} = \delta(\pi_j, \vec{a})$ for some joint action $\vec{a}$ such that for every $i \in \Gamma$, $a_i = \sigma_i(\pi_{\leq j})$, and for every $i \in \overline{\Gamma}$, $a_i \in d(i, \pi_j)$. We denote with $out(s, \Sigma_\Gamma)$ the set of all Σ_Γ-compatible paths from s.

Now, we have all the ingredients to give the semantics of ATL*.

Definition 2.4. The satisfaction relation $\models$ for an iCGS M, state $s \in S$, path $\pi \in S^\omega$, atom $q \in AP$, and ATL* formula ϕ is defined as (clauses for Boolean connectives are immediate and thus omitted):

$$\begin{array}{lll} (M, s) \models q & \text{iff} & q \in V(s) \\ (M, s) \models \langle\langle\Gamma\rangle\rangle\psi & \text{iff} & \text{for some joint strategy } \Sigma_\Gamma, \\ & & \text{for all } \pi \in out(s, \Sigma_\Gamma),\ (M, \pi) \models \psi \\ (M, \pi) \models \varphi & \text{iff} & (M, \pi_1) \models \varphi \\ (M, \pi) \models X\psi & \text{iff} & (M, \pi_{\geq 2}) \models \psi \\ (M, \pi) \models \psi U \psi' & \text{iff} & \text{for some } k \geq 1,\ (M, \pi_{\geq k}) \models \psi', \text{ and} \\ & & \text{for all } 1 \leq j < k,\ (M, \pi_{\geq j}) \models \psi \end{array}$$

We say that formula φ is *true* in an iCGS M, or $M \models \varphi$, iff $(M, s_I) \models \varphi$. Now, we state the model checking problem.

Definition 2.5. Given an iCGS M and a formula φ, the model checking problem concerns determining whether $M \models \varphi$.

Since the semantics provided in Def. 2.4 is the standard interpretation of ATL* [1], it is well known that model checking ATL, *a fortiori* ATL*, against iCGS with imperfect information and perfect recall is undecidable [12]. In the rest of the paper we also consider CTL*, see [2] for details on this logic.

3 CASE STUDY

The Curiosity rover is one of the most complex systems successfully deployed in a planetary exploration mission to date. Its main objectives include recording image data and collecting soil/rock data. Differently from the original [20], in this example the rover is equipped with decision making capabilities, which make it autonomous. We simulate an inspection mission, where the Curiosity patrols a topological map of the surface of Mars.

In Figure 1, we model an example of a mission for the rover. The rover starts in state s_I, and it has to perform a setup action, consisting of checking the three main rover's components: arm (ca), mast (cm), or the wheels (cw). To save time and energy, the mechanic (the entity capable of performing the setup checks and making any corrections) performs only one setup operation per mission. Since the exact setup operation is not known by the rover, from its point of view, states s_1, s_2, s_3 are equivalent, *i.e.* it cannot distinguish between the three setup operations. However, the rover has to validate the setup operation. That is, if in the initial state the mechanic selects the operation to check the arm, *i.e.* ca, then from the next state, *i.e.* s_1, the rover needs to perform the ca action to be able to move from s_1. The same reasoning can be applied for the other two setup actions. After the selection, the mechanic can choose to check and eventually correct the component (action ok) or to decline the operation (action nok). In the former case the rover can continue the mission while in the latter the mission terminates with an error. In case the mechanic collaborates, the rover can start the mission. We consider with state s_4 the fact that the rover is in the base camp. So, its aim is to move from its position, to make a picture of a sample rock of Mars and then to return to the initial position. More in detail, from the state s_4, it can decide to move left (L) moving to state s_6, or right (R) moving to state s_7. In these two states the rover can make a photo of a sample rock (action mp). After this step, the rover has to conclude the mission by returning to the base camp. To accomplish the latter, it needs to do the complement moving action to go back to the previously visited state (R from s_6 and L from s_7).

So, given the model M we can define several specifications. For example, the specification that describes the rover mission is $\varphi_1 = \langle\langle rover \rangle\rangle F((oc \wedge rm) \wedge F((pl \vee pr) \wedge F(oc \wedge rm)))$. In words the latter formula means that there exists a strategy for the rover that sooner or later it can be ready to start the mission, can make a picture of a sample rock, and can return to the base camp. In the latter formula, we assume that the mechanic can decide not to cooperate. In this case, it is impossible for the rover to achieve the end of the mission even using perfect recall strategies. If we assume the cooperation of the mechanic (*i.e.* the mechanic checks a component and eventually corrects it via action ok), we can rewrite the specification as $\varphi_2 = \langle\langle rover, mechanic \rangle\rangle F((oc \wedge rm) \wedge \langle\langle rover \rangle\rangle F((pl \vee pr) \wedge F(oc \wedge rm)))$. In this case, by using memoryless strategies the formula φ_2 still remains false since the rover can select only one action on states s_1, s_2, and s_3 and in states s_6 and s_7, so it is impossible for it to validate any setup action given by the mechanic and to first make the picture and then go to the end of the mission. While, by using memoryfull strategies, the rover has a strategy to make the formula true by considering the cooperation of the mechanic. In fact, a simple strategy σ can be generated by: $\sigma(s_I) = chk$, $\sigma(s_I s_j) = ca$, $\sigma(s_I s_j s_k) = cm$, $\sigma(s_I s_j s_k s_r) = cw$, $\sigma(s_I s_j s_4) = L$, $\sigma(s_I s_j s_4 s_6) = mp$, $\sigma(s_I s_j s_4 s_6 s_5) = i$, and $\sigma(s_I s_j s_4 s_6 s_5 s_6) = R$, where $j, k, r \in \{1, 2, 3\}$. With the above observations, we can conclude that, to verify the specification φ_2 in the model M, we need ATL* model checking in the context of imperfect information and perfect recall, a problem in general undecidable. Other interesting specifications can involve specific rover's status. For example, we could be interested to verify if there exists a strategy for the rover in coalition with the mechanic such that it can be ready to make a picture but it is not in position to do that and, from that point, if it has the ability to make a picture and return in the starting mission, *i.e.* we want to verify the formula $\varphi_3 = \langle\langle rover, mechanic \rangle\rangle F(rp \wedge \neg ip \wedge \langle\langle rover \rangle\rangle F((pl \vee pr) \wedge F(oc \wedge rm)))$. It is easy to see that φ_3 is false in the model M because there is not a state in which $rp \wedge \neg ip$. Also, to check φ_3 in the model M, we need ATL* model checking in the context of imperfect information and perfect recall, a problem in general undecidable.

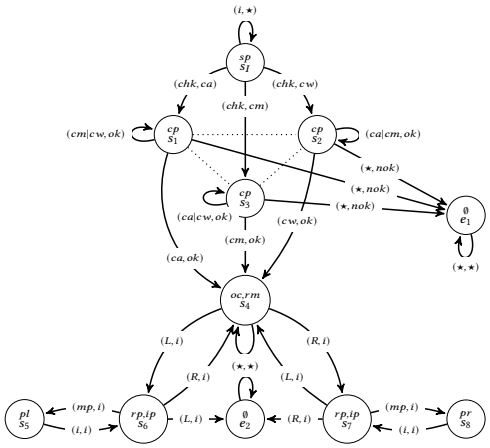

Figure 1: The rover's mission where i stands for idle, $\star$ for any action, and the rover's equivalence relation is denoted with dotted lines. The atoms have the following acronyms: sp as starting position, cp as check phase, oc as ok check phase, rm as ready to mission, rp as ready to make a picture, ip as in position, pl as picture left, and pr as picture right.

4 OUR PROCEDURE

Our procedure aims at giving a partial answer to a well-known undecidable problem. It first starts processing the input formula φ

and the model M to explicitly denote all atoms, positively. In fact, in general, given the labeling function V and a state s, it is assumed that the atoms not present in $V(s)$ are false. In our procedure, this is not enough due to the verification on sub-models in which the preservation results hold for the negation normal form of ATL*. For this reason, we need to make explicit what is true and what is false by duplicating each atom q in φ with a new atom nq in such a way that, if q is true in s, then $q \in V(s)$, otherwise q is false in s and consequently $nq \in V(s)$. This approach makes truth and falsity of atoms explicit in each state. With these new versions of M and φ, our procedure verifies all the sub-formulas of φ against a set of sub-models of the model M with perfect information. Such sub-models are extracted in order to break all indistinguishability relations (*i.e.*, for each sub-model each state is only indistinguishable from itself). Two classes of sub-models are defined: negative and positive. In the verification of φ, the former class under-approximates M while the latter over-approximates M. In more detail, negative (resp., positive) sub-models limit (resp., expand) the number of sub-formulas of φ that can be satisfied. Indeed, if a sub-formula is satisfied in a negative sub-model, where there are more limitations than M, then the sub-formula is also satisfied in M. While, if a sub-formula is not satisfied in a positive sub-model, where there are less limitations than M, then the sub-formula is also not satisfied in M. Then, the procedure splits φ into its sub-formulas, and verifies them over the sub-models of M with perfect information previously extracted. Every time a sub-formula is satisfied in a state s, the procedure keeps track of it by creating a new atom, by adding the new atom on s, and by updating φ with the new created atom. Finally, the procedure substitutes the remaining not satisfied strategic operators with path operators of CTL* and through its model checking (whose soundness is proved via Lemma 4.12 and Theorem 4.13) it checks whether M always reaches the atoms verified in the negative sub-model (universally), or whether M reaches at least once the atoms verified in the positive sub-model (existentially). These last two aspects are linked to Lemma 4.11, where we prove how the verification result over sub-models can be propagated to the original model.

Algorithm 1 *Preprocessing* (M, φ)

```
1: φ = NNF(φ);
2: neg-atoms = extract negated atoms from φ;
3: for ¬q ∈ neg-atoms do
4:     generate atom nq;
5:     AP = AP ∪ nq;
6:     replace ¬q in φ with nq;
7:     for s ∈ S do
8:         if q ∉ V(s) then
9:             V(s) = V(s) ∪ nq;
```

4.1 Phase 0: preprocessing

Given an iCGS M and an ATL* formula φ, Algorithm 1 rewrites φ in its equivalent Negation Normal Form (NNF), replaces the negated atoms by additional atoms, and updates the model M accordingly. With more detail, first, the NNF of φ is generated and all the negated atoms are extracted (lines 1-2). Then, for each negated atom $\neg q$, the algorithm generates a new atomic proposition nq (line 4), add nq to the set of atomic propositions AP (line 5), updates the formula (line 6), and updates the labeling function of M. For the latter, for each state s of M, the algorithm checks if q is false in s; if this is the case, it adds nq to the set of atomic propositions true on s (lines 7-9).

Now, we give the complexity of the described procedure.

Theorem 4.1. *Algorithm 1 terminates in polynomial time w.r.t. the size of φ and M.*

4.2 Phase 1: find sub-models

We present how to find the sub-models with perfect information inside the original model. First, we give the intuition behind such sub-models, and why we need them. Given a model M, each negative (resp., positive) sub-model denotes an under-approximation (resp., over-approximation) of M. Thus, negative (resp., positive) sub-models can be used to prove the satisfaction (resp., violation) of properties. So, if we (resp., don't) find a strategy to satisfy a property in the negative (resp., positive) sub-model, then the same strategy can be used (resp., there is definitely no strategy) in the original model. The formal definitions of these sub-models follow.

Definition 4.2 (Negative sub-models). Given an iCGS $M = \langle Ag, AP, S, s_I, \{Act_i\}_{i\in Ag}, \{\sim_i\}_{i\in Ag}, d, \delta, V\rangle$, we denote with $M^n = \langle Ag, AP, S^n, s_I^n, \{Act_i\}_{i\in Ag}, \{\sim_i^n\}_{i\in Ag}, d^n, \delta^n, V^n\rangle$ a negative sub-model of M such that:

- the set of states is defined as $S^n = S^\star \cup \{s_\perp\}$, where $S^\star \subseteq S$, and $s_I^n \in S^\star$ is the initial state.
- $\sim_i^n$ is defined as the corresponding $\sim_i$ restricted to $S^\star$.
- The protocol function is defined as $d^n : Ag \times S^n \to (2^{Act} \setminus \{\emptyset\})$, where $d^n(i,s) = d(i,s)$, for every $s \in S^\star$ and $d^n(i, s_\perp) = Act_i$, for all $i \in Ag$.
- The transition function is defined as $\delta^n : S^n \times ACT \to S^n$, where given a transition $\delta(s,\vec{a}) = s'$, if $s, s' \in S^\star$ then $\delta^n(s,\vec{a}) = \delta(s,\vec{a}) = s'$ else if $s' \in S \setminus S^\star$ and $s \in S^n$ then $\delta^n(s,\vec{a}) = s_\perp$.
- for all $s \in S^\star$, $V^n(s) = V(s)$ and $V^n(s_\perp) = \emptyset$.

Definition 4.3 (Positive sub-models). Given an iCGS $M = \langle Ag, AP, S, s_I, \{Act_i\}_{i\in Ag}, \{\sim_i\}_{i\in Ag}, d, \delta, V\rangle$, we denote with $M^p = \langle Ag, AP, S^p, s_I^p, \{Act_i\}_{i\in Ag}, \{\sim_i^p\}_{i\in Ag}, d^p, \delta^p, V^p\rangle$ a positive sub-model of M such that:

- the set of states is defined as $S^p = S^\star \cup \{s_\top\}$, where $S^\star \subseteq S$, and $s_I^p \in S^\star$ is the initial state.
- $\sim_i^p$ is defined as the corresponding $\sim_i$ restricted to $S^\star$.
- The protocol function is defined as $d^p : Ag \times S^p \to (2^{Act} \setminus \{\emptyset\})$, where $d^p(i,s) = d(i,s)$, for every $s \in S^\star$ and $d^p(i, s_\top) = Act_i$, for all $i \in Ag$.
- The transition function is defined as $\delta^p : S^p \times ACT \to S^p$, where given a transition $\delta(s,\vec{a}) = s'$, if $s, s' \in S^\star$ then $\delta^p(s,\vec{a}) = \delta(s,\vec{a}) = s'$ else if $s' \in S \setminus S^\star$ and $s \in S^p$ then $\delta^p(s,\vec{a}) = s_\top$.
- for all $s \in S^\star$, $V^p(s) = V(s)$ and $V^p(s_\top) = AP$.

Informally, $s_\top$ and $s_\perp$ are sink states that replace the sub-states removed from the original model (*i.e.*, $S \setminus S^\star$). The above definitions are more general (they can have imperfect information), however in our procedure only sub-models with perfect information are used.

The procedure to generate the set of negative and positive sub-models of M with perfect information for the agents involved in the

Algorithm 2 *FindSub-models* (M, φ)

1: extract $\sim$ from M;
2: extract S from M;
3: $candidates = \emptyset$;
4: $substates = \{S\}$;
5: **while** $substates \neq \emptyset$ **do**
6: extract S' from $substates$;
7: **if** there exists $s \sim_i s'$ with $s, s' \in S'$, $s \neq s'$ and $i \in Ag(\varphi)$ **then**
8: $S_1 = S' \setminus \{s\}$;
9: $substates = S_1 \cup substates$;
10: $S_2 = S' \setminus \{s'\}$;
11: $substates = S_2 \cup substates$;
12: **else**
13: $M^n = GenerateNegative(M, S')$;
14: $M^p = GeneratePositive(M, S')$;
15: $candidates = \langle M^n, M^p \rangle \cup candidates$;
16: **return** $candidates$;

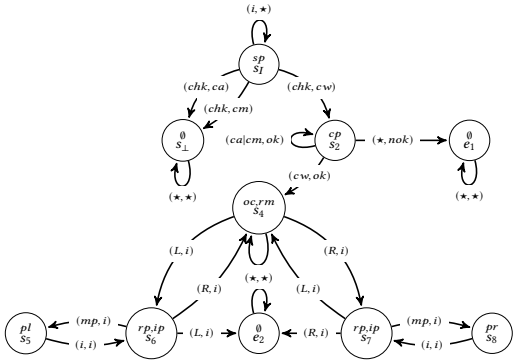

Figure 2: A negative sub-model generated by Algorithm 2 from model M depicted in Figure 1.

coalitions of φ is presented in Algorithm 2. Informally, the algorithm takes an iCGS M and an ATL* formula φ. It works with two sets: *substates* contains the sets of states of M that have to be evaluated and *candidates* contains couples of sub-models of M with perfect information. Each couple represents the positive and negative sub-models for a given subset of states of M. In the first part of the algorithm (lines 1-4) the set of states and the indistinguishability relation are extracted from the model M and the sets *substates* and *candidates* are initialized to S and $\emptyset$, respectively. The main loop (lines 5-15) works until the set *substates* is empty. In each iteration an element S' is extracted from *substates* and if there is an equivalence relation over the states in S' for some agent $i \in Ag(\varphi)$[1] then two new subsets of states are generated to remove this relation (lines 8-11), otherwise S' is a set of states with perfect information for the agents in $Ag(\varphi)$. So, from S' two models are generated, a negative sub-model as described in Definition 4.2 and a positive sub-model as described in Definition 4.3, and the couple is added to the set *candidates*. The latter is returned at the end of Algorithm 2.

Example 4.4. In Figure 2, we show a candidate negative sub-model M^n extracted from the rover mission (see Figure 1) using Algorithm 2. Note that, we can generate the positive sub-model counterpart by replacing the state $s_\perp$ with $s_\top$ and following the labeling rule of Definition 4.3.

[1] $Ag(\varphi)$ is the set of agents involved in the coalitions of φ.

Algorithm 3 *CheckSub-formulas* $(\langle M^n, M^p \rangle, \varphi)$

1: $result = \emptyset$;
2: $subformulas = SubFormulas(\varphi)$;
3: **while** $subformulas \neq \emptyset$ **do**
4: extract ψ from $subformulas$;
5: **for** $s \in S^n \cap S^p$ **do**
6: **if** $M^n, s \models_{IR} \psi$ **then**
7: $M^n = UpdateModel(M^n, s, atom_\psi)$;
8: $result = \langle s, \psi, natom_\psi \rangle \cup result$;
9: **if** $M^p, s \models_{IR} \psi$ **then**
10: $M^p = UpdateModel(M^p, s, atom_\psi)$;
11: $result = \langle s, \psi, patom_\psi \rangle \cup result$;
12: $M^p = UpdateModel(M^p, s_\top, atom_\psi)$;
13: **return** $result$;

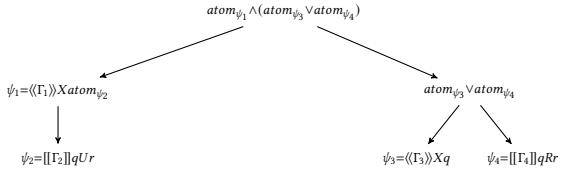

Figure 3: The tree for the formula $\varphi = \langle\langle \Gamma_1 \rangle\rangle X[[\Gamma_2]] qUr \wedge (\langle\langle \Gamma_3 \rangle\rangle Xq \vee [[\Gamma_4]] qRr)$.

Finally, we give the complexity of the procedure.

THEOREM 4.5. *Algorithm 2 terminates in exponential time w.r.t the size of M.*

4.3 Phase 2: check sub-formulas

Given a candidate, we present here how to check the sub-formulas on it. The procedure is presented in Algorithm 3. Informally, the algorithm takes a candidate and an ATL* formula φ. It works with the set *result* that contains tuples $\langle s, \psi, vatom_\psi \rangle$, where s is a state in a sub-model, ψ is a sub-formula of φ, and $vatom_\psi$ is the atomic proposition generated from ψ, where $v \in \{p, n\}$ is the type of the sub-model (*i.e.*, positive or negative). In more detail, the set *result* contains tuples $\langle s, \psi, vatom_\psi \rangle$, where s satisfies ψ and $vatom_\psi$ represents the atom that will take the place of ψ in the original formula φ (v is used to remember if ψ is satisfied in a negative ($v = n$) or positive sub-model ($v = p$)). In line 1 the set *result* is initialized to $\emptyset$. In line 2 the set *subformulas* is initialized via the subroutine $SubFormulas()$. By $SubFormulas(\varphi)$ we take all the sub-formulas of φ with only one strategic operator. With more detail, we can see our formula φ as a tree, where the root is φ, the other nodes are the sub-formulas of φ, and the leaves of the tree are the sub-formulas of φ having only one strategic operator. Given this tree, the algorithm generates for each node ψ an atomic proposition $atom_\psi$ and updates the tree by replacing every occurrence of ψ in its ancestors with $atom_\psi$. Then, the nodes of the tree are stored in the set *subformulas* by using a depth first search. For instance, let us consider the formula $\varphi = \langle\langle \Gamma_1 \rangle\rangle X[[\Gamma_2]] qUr \wedge (\langle\langle \Gamma_3 \rangle\rangle Xq \vee [[\Gamma_4]] qRr)$; when we apply $SubFormulas(\varphi)$ the tree in Figure 3 is generated and we obtain the ordered set $subformulas = [\psi_2, \psi_1, \psi_3, \psi_4]$. The extraction of the set of sub-formulas from φ is necessary to verify whether there are sub-models satisfying/unsatisfying at least a part of φ (not exclusively the whole φ). If that is the case, such

sub-models can later on be used to verify φ over the entire model M through CTL* model checking. The main loop (lines 3-12) works until all the sub-formulas of φ are treated. By starting from the first formula of the set, the loop in lines 5-11 proceeds for each state, and checks the current sub-formula against the currently selected sub-models M^n and M^p. Note that, in lines 6 and 9, we have *IR* as verification mode[2]. Thus, the models are checked under the assumptions of perfect information and perfect recall. If the sub-formula is satisfied over the model M^n (line 6), then by calling function $UpdateModel(M^n, s, atom_\psi)$, the procedure updates the model M^n by updating the set of atomic propositions as $AP = AP \cup \{atom_\psi\}$ and the labelling function of s as $V(s) = V(s) \cup \{atom_\psi\}$ (line 7). Furthermore, in line 8, the set *result* is updated by adding a new tuple. The reasoning applied to the model M^n is also applied to M^p (lines 9-11). At the end of the procedure, the state $s_\top$ is updated (line 12) since, by Definition 4.3, it needs to contain all the possible atoms that are in M.

Example 4.6. Given the negative sub-model M^n as in Figure 2 and its positive counterpart M^p as input of Algorithm 3 we can analyze formulas $\varphi_1 = \langle\langle rover\rangle\rangle F((oc \wedge rm) \wedge F((pl \vee pr) \wedge F(oc \wedge rm)))$, $\varphi_2 = \langle\langle rover, mechanic\rangle\rangle F((oc \wedge rm) \wedge \langle\langle rover\rangle\rangle F((pl \vee pr) \wedge F(oc \wedge rm)))$, and $\varphi_3' = \langle\langle rover, mechanic\rangle\rangle F(rp \wedge nip \wedge \langle\langle rover\rangle\rangle F((pl \vee pr) \wedge F(oc \wedge rm)))$[3]. We start with φ_1 where the set of sub-formulas is initialized to $\{\varphi_1\}$ since there is only one strategic operator in φ_1. By line 4, the algorithm extracts φ_1 and by the loop in lines 5-11 it verifies that the formula is true in the negative and positive sub-models in states s_4, s_5, s_6, s_7, and s_8. By line 7 (resp., 10) it updates the model M^n (resp., M^p) by adding $atom_{\psi_1}$ in AP and by updating the labeling function as $V'(s_i) = V(s_i) \cup \{atom_{\psi_1}\}$, for all $i \in \{4, 5, 6, 7, 8\}$. Since there is only one strategic operator, the procedure is concluded. For φ_2, the set *subformulas* is initialized to the set $\{\psi_1 = \langle\langle rover\rangle\rangle F((pl \vee pr) \wedge F(oc \wedge rm)), \psi_2 = \langle\langle rover, mechanic\rangle\rangle F((oc \wedge rm) \wedge atom_{\psi_1})\}$. By line 4, the algorithm extracts ψ_1 and by the loop in lines 5-11 it verifies that the formula is true in the negative and positive sub-models in states s_4, s_5, s_6, s_7, and s_8. By line 7 (resp., 10) it updates the model M^n (resp., M^p) by adding $atom_{\psi_1}$ in AP and by updating the labeling function as $V'(s_i) = V(s_i) \cup \{atom_{\psi_1}\}$, for all $i \in \{4, 5, 6, 7, 8\}$. For the second iteration of the main loop (lines 3-11) the formula ψ_2 is analyzed. In this case, ψ_2 is true in the negative and positive sub-models in states s_1, s_2, s_4, s_5, s_6, s_7, and s_8. Again, the models M^n and M^p are updated by adding $atom_{\psi_2}$ in AP and by updating the labeling function as $V'(s_i) = V(s_i) \cup \{atom_{\psi_2}\}$, for all $i \in \{1, 2, 4, 5, 6, 7, 8\}$. For φ_3', the set of sub-formulas is initialized to the set $\{\psi_1 = \langle\langle rover\rangle\rangle F((pl \vee pr) \wedge F(oc \wedge rm)), \psi_2 = \langle\langle rover, mechanic\rangle\rangle F(rp \wedge nip \wedge atom_{\psi_1})\}$. Since ψ_1 is the same as before for φ_2 we have the same behavior of the algorithm. For the second iteration, ψ_2 is false in all the states in $S^p \cap S^n$. So, no update is done in the set *result*.

To conclude this part, we provide the complexity result.

Theorem 4.7. *Algorithm 3 terminates in double exponential time w.r.t. the size of φ.*

[2] As usual, *I* denotes perfect information and *R* denotes perfect recall.

[3] φ_3' is the updated version of φ_3 given in Section 3, where *nip* replaces $\neg ip$ as described in the algorithm of Section 4.1.

Algorithm 4 $Verification(M, \varphi, result)$

```
1:  k = ?;
2:  for s ∈ S do
3:      take set atoms from result(s);
4:      UpdateModel(M, s, atoms);
5:  φ_n = φ, φ_p = φ;
6:  while result is not empty do
7:      extract ⟨s, ψ, vatom_ψ⟩ from result;
8:      if v = n then
9:          φ_n = UpdateFormula(ψ_n, ψ, natom_ψ);
10:     else
11:         φ_p = UpdateFormula(ψ_p, ψ, patom_ψ);
12: φ_A = FromATLtoCTL(φ_n, n);
13: φ_E = FromATLtoCTL(φ_p, p);
14: if M ⊨ φ_A then
15:     k = ⊤;
16: if M ⊭ φ_E then
17:     k = ⊥;
18: return k;
```

4.4 Phase 3: CTL* verification

In the previous sections, we showed how to extract sub-models of a model M satisfying/unsatisfying at least one sub-formula φ' of φ. Here, we present in Algorithm 4 how to apply CTL* model checking to try to conclude the satisfaction/violation of φ in M by using such sub-models. The algorithm takes as input an iCGS M, an ATL* formula φ, and a set *result*. The algorithm starts by updating the model M with the atoms generated from Algorithm 3 and stored in *result* (lines 2-4). In lines 5-11, the procedure generates formulas φ_p and φ_n in accordance with the tuples in *result*. In this part of the procedure, the algorithm uses the subroutine $UpdateFormula(\varphi, \psi, atom_\psi)$. The latter modifies the formula φ by replacing each occurrence of ψ with $atom_\psi$, where $atom_\psi$ is a unique atom identifier for formula ψ. As an example, let us consider $\varphi = \langle\langle\Gamma\rangle\rangle X\langle\langle\Gamma\rangle\rangle rUq$, with $\psi = \langle\langle\Gamma\rangle\rangle rUq$. When we apply $UpdateFormula(\varphi, \psi, atom_\psi)$, we obtain an updated version of φ, where all occurrences of ψ have been replaced by $atom_\psi$, *i.e.* φ becomes $\langle\langle\Gamma\rangle\rangle X atom_\psi$. The formulas φ_n and φ_p represent the ATL* formula which remains to verify in M. To achieve this, we transform φ_n and φ_p into their CTL* counterparts φ_A and φ_E, respectively (lines 12-13). Given a formula φ in ATL*, we denote with φ_A (resp., φ_E) the universal formula (resp., existential) of φ, *i.e.* we substitute each occurrence of a strategic operator in φ with the universal (resp., existential) operator of CTL*. Given φ_A and φ_E we can perform standard CTL* model checking. First, by verifying if φ_A is satisfied by M (line 14). If this is the case we can derive that $M \models \varphi$ and set $k = \top$. Otherwise, the algorithm continues verifying if φ_E is not satisfied by M (line 16). If this is the case we can derive that $M \not\models \varphi$ and set $k = \bot$. If neither of the two previous conditions are satisfied then the inconclusive result is returned.

Example 4.8. As for the previous example, we continue to analyze formulas φ_1, φ_2, and φ_3'. For φ_1 the set *result* is composed by the tuples: $\langle s_4, \psi_1, vatom_{\psi_1}\rangle$, $\langle s_5, \psi_1, vatom_{\psi_1}\rangle$, $\langle s_6, \psi_1, vatom_{\psi_1}\rangle$, $\langle s_7, \psi_1, vatom_{\psi_1}\rangle$, *and* $\langle s_8, \psi_1, vatom_{\psi_1}\rangle$ *where* $v \in \{n, p\}$. In lines 2-4, the model M depicted in Figure 1 is updated with the atoms $natom_{\psi_1}$ and $patom_{\psi_1}$ in states $s_4, s_5, s_6, s_7,$ *and* s_8. Then, in lines 6-11, φ_n and φ_p are generated as $natom_{\psi_1}$ and $patom_{\psi_1}$, respectively. Since there are no strategic operators in the latter formulas then $\varphi_A = \varphi_n$ *and* $\varphi_E = \varphi_p$. The condition in line 14 is not satisfied but it

is in line 16. So, the output of the procedure is $\bot$, *i.e.* the formula is false in the model M. For φ_2 the set *result* is composed by the tuples: $\langle s_4, \psi_1, vatom_{\psi_1}\rangle, \langle s_5, \psi_1, vatom_{\psi_1}\rangle, \langle s_6, \psi_1, vatom_{\psi_1}\rangle, \langle s_7, \psi_1, vatom_{\psi_1}\rangle, \langle s_8, \psi_1, vatom_{\psi_1}\rangle, \langle s_I, \psi_2, vatom_{\psi_2}\rangle, \langle s_2, \psi_2, vatom_{\psi_2}\rangle, \langle s_4, \psi_2, vatom_{\psi_2}\rangle, \langle s_5, \psi_2, vatom_{\psi_2}\rangle, \langle s_6, \psi_2, vatom_{\psi_2}\rangle, \langle s_7, \psi_2, vatom_{\psi_2}\rangle, \langle s_8, \psi_2, vatom_{\psi_2}\rangle$, where $v \in \{n, p\}$. In lines 2-4, M is updated with the atoms $natom_{\psi_1}$ and $patom_{\psi_1}$ in states s_4, s_5, s_6, s_7, and s_8; and atoms $natom_{\psi_2}$ and $patom_{\psi_2}$ in states s_I, s_2, s_4, s_5, s_6, s_7, and s_8. Then, in lines 6-11 φ_n and φ_p are generated as $natom_{\psi_2}$ and $patom_{\psi_2}$, respectively. As before, $\varphi_A = \varphi_n$ and $\varphi_E = \varphi_p$. The condition in line 16 is not satisfied but it is in line 14. So, the output of the procedure is $\top$, *i.e.* the formula is true in the model M. For φ_3' the set result is composed by the tuples: $\langle s_4, \psi_1, vatom_{\psi_1}\rangle$, $\langle s_5, \psi_1, vatom_{\psi_1}\rangle$, $\langle s_6, \psi_1, vatom_{\psi_1}\rangle$, $\langle s_7, \psi_1, vatom_{\psi_1}\rangle$, and $\langle s_8, \psi_1, vatom_{\psi_1}\rangle$ where $v \in \{n, p\}$. In lines 2-4, M is updated with the atoms $natom_{\psi_1}$ and $patom_{\psi_1}$ in states s_4, s_5, s_6, s_7, and s_8. Then, in lines 6-11 φ_n and φ_p are generated as $\langle\langle rover, mechanic\rangle\rangle F(rp \wedge nip \wedge natom_{\psi_1})$ and $\langle\langle rover, mechanic\rangle\rangle F(rp \wedge nip \wedge patom_{\psi_1})$, respectively. Since there are strategic operators in the latter formulas then the algorithm produces formulas $\varphi_A = AF(rp \wedge nip \wedge natom_{\psi_1})$ and $\varphi_E = EF(rp \wedge nip \wedge patom_{\psi_1})$. The condition in line 14 is not satisfied but it is in line 16. So, the output of the procedure is $\bot$, *i.e.* the formula is false in the model M.

Note that, in these examples, with our algorithm we can find solution to some ATL model checking problems that are in the class of undecidable problems (naturally, not for all ATL model checking problems in such class, otherwise it would be decidable). Furthermore, we preserve the truth values, *i.e.* they are the same as described in Section 3.

To conclude this part, we provide the complexity result.

Theorem 4.9. *Algorithm 4 terminates in exponential time w.r.t. the size of φ and polynomial time w.r.t. to the size of M.*

4.5 The overall model checking procedure

Given Algorithms 1, 2, 3, and 4, we can provide the overall procedure as described in Algorithm 5.

Algorithm 5 *ModelCheckingProcedure* (M, φ)

```
1: Preprocessing(M, φ);
2: candidates = FindSub-models(M, φ);
3: if |candidates| = 1 then
4:     return M ⊨_IR φ
5: while candidates is not empty do
6:     extract ⟨M_n, M_p⟩ from candidates;
7:     result = CheckSub-formulas(⟨M_n, M_p⟩, φ);
8:     k = Verification(M, φ, result);
9:     if k ≠ ? then
10:        return k;
11: return ?;
```

The *ModelCheckingProcedure*() takes in input a model M and a formula φ, and calls the function *Preprocessing*() to generate the NNF of φ and to replace all negated atoms with new positive atoms inside M and φ. After that, it calls the function *FindSub-models*() to generate all the positive and negative sub-models that represent all the possible sub-models with perfect information. If the number of candidates is equal to one (line 3), *i.e.*, the model M given in input is with perfect information, then we can directly call the ATL* model checking procedure in case of perfect information and perfect recall. Then, there is a while loop (lines 5-10) that for each candidate checks the sub-formulas true on the sub-models via *CheckSub-formulas*() and the truth value of the whole formula via *Verification*(). If the output of the latter procedure is different from ? then the result is directly returned (line 10).

Theorem 4.10. *Algorithm 5 terminates in double exponential time w.r.t. the size of φ and exponential time w.r.t. the size of M.*

4.6 Soundness

Before showing the soundness of our algorithm, we need to provide some auxiliary lemmas. Since in our procedure we use ATL* formulas in NNF with duplication of atoms to avoid the negations, in the rest of the section we will consider this type of formulas.

We start with two preservation results from sub-models to the original model.

Lemma 4.11. *Given a model M, a negative (resp., positive) sub-model with perfect information M^n (resp., M^p) of M, and a formula φ of the form $\varphi = \langle\langle\Gamma\rangle\rangle\psi$ for some $\Gamma \subseteq Ag$. For any $s \in S^n \setminus \{s_\bot\}$ (resp., $S^p \setminus \{s_\top\}$), we have that:*

$$M^n, s \models \varphi \Rightarrow M, s \models \varphi$$
$$M^p, s \not\models \varphi \Rightarrow M, s \not\models \varphi$$

The above result also holds for the universal strategic quantifier.

Now, we give two preservation results from CTL* formulas to ATL* formulas that derives from [1].

Lemma 4.12. *Given a model M, a formula φ in ATL* written in NNF, and the CTL* universal (resp., existential) version φ_A (resp., φ_E) of φ. For any $s \in S$, we have that:*

$$M, s \models \varphi_A \Rightarrow M, s \models \varphi$$
$$M, s \not\models \varphi_E \Rightarrow M, s \not\models \varphi$$

Finally, we show the soundness of our procedure.

Theorem 4.13. *Algorithm 5 is sound: if the value returned is not ?, then $M \models \varphi$ iff $k = \top$.*

Proof. Suppose that the value returned is different from ?. In particular, either $k = \top$ or $k = \bot$. If $M \models \varphi$ and $k = \bot$, then by Algorithm 4 and 5, we have that $M' \not\models \varphi_E$. Now, there are two cases: (1) M and M' are the same models or (2) M differs from M' for some atomic propositions added to M' in lines 2-4 of Algorithm 4. For (1), we know that M and M' are labeled with the same atomic propositions and thus $M' \not\models \varphi_E$ implies $M \not\models \varphi_E$ and, by Lemma 4.12, $M \not\models \varphi$, a contradiction. Hence, $k = \top$ as required. For (2), suppose that M' has only one additional atomic proposition $atom_\psi$, *i.e.* $AP' = AP \cup \{atom_\psi\}$. The latter means that Algorithm 3 found a positive sub-model M^p in which $M^p, s \models \psi$, for some $s \in S^p$. By Lemma 4.11, for all $s \in S^p \setminus \{s_\top\}$, we know that if $M^p, s \not\models \psi$ then $M, s \not\models \psi$. So, M' over-approximates M, *i.e.* there could be some states that in M' are labeled with $atom_\psi$ but they don't satisfy ψ in M. Thus, if $M' \not\models \varphi_E$ then $M \not\models \varphi_E$ and, by Lemma 4.12, $M \not\models \varphi$, a contradiction. Hence, $k = \top$ as required. Obviously, we can generalize the above reasoning in case M and M' differ for multiple atomic propositions. The case $k = \top$ can be solved by a similar reasoning. □

5 OUR TOOL

The algorithms presented were implemented in Java[4] (∼7k lines of code in total). The tool expects a model in input formatted as a Json file. This file is then parsed, and an internal representation of the model and formula are generated. After the preprocessing phase, Algorithm 2 is called to extract the sub-models. The verification of a sub-model against a sub-formula is achieved by translating the sub-model into its equivalent ISPL (Interpreted Systems Programming Language) program, which is then verified by using the model checker MCMAS[5]. This corresponds to the verification steps for Algorithm 3 at lines 6 and 9, and Algorithm 4 at lines 14 and 16. The entire manipulation, from parsing the model formatted in Json, to translating the latter to its equivalent ISPL program, was performed by extending an existent Java library [10]; the rest of the tool derives directly from the algorithms presented in this paper.

5.1 Experiments

We tested our tool over the rover's mission, on a machine with the following specifications: Intel(R) Core(TM) i7-7700HQ CPU @ 2.80GHz, 4 cores 8 threads, 16 GB RAM DDR4. By considering the model of Figure 1 and the three formulas φ_1, φ_2 and φ_3 presented in the same section, Algorithm 2 finds 3 sub-models with perfect information (precisely, 3 negative and 3 positive sub-models). Such sub-models are then evaluated by applying first Algorithm 3, and then Algorithm 4 (as shown in Algorithm 5). The resulting implementation takes 0.2 [sec] to conclude the satisfaction of φ_2 and the violation of φ_1 and φ_3 on the model (empirically confirming our expectations). The example of Figure 1 is expressly small, since it is used as running example to help the reader to understand the contribution. However, we further tested our tool on several extensions of the rover example that contained hundreds (∼ 150) of states and dozens of agents (multiple rovers and mechanics). Such more complex scenarios served us as a stress test to analyse the performance of our tool for more realistic MAS [6]. Since Algorithm 5 is $2EXPTIME$, when the model grows, the performance is highly affected. Furthermore, we tested our tool on a large set of automatically and randomly generated iCGSs and ATL formulas. The size of each iCGS was between 5 and 30 states while the formulas where between 2 and 10 strategic/temporal operators. The objective of these experiments was to show how many times our algorithm returned a conclusive verdict. For each model, we ran our procedure and counted the number of times a solution was returned. Note that, our approach terminates in any case, but since the general problem is undecidable, the result might be inconclusive (*i.e.* ?). The execution time to complete all experiments took ∼ 15 hours in total. In Figure 4, we report our results by varying the percentage of imperfect information (x axis) inside the iCGSs, from 0% (perfect information, *i.e.* all states are distinguishable for all agents), to 100% (no information, *i.e.* no state is distinguishable for any agent). For each percentage selected, we generated 10k random iCGSs and counted the number of times our algorithm returned with a conclusive result (*i.e.* $\top$ or $\bot$). As shown in Figure 4,

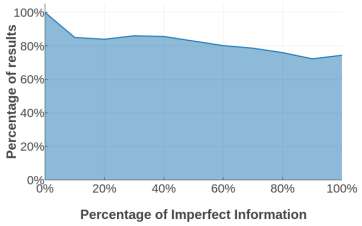

Figure 4: Experimental results.

our algorithm returned a conclusive verdict for 80% of the models analysed (y axis). It is important to notice how this result was not influenced (empirically) by the percentage of imperfect information added inside the iCGSs. In fact, the accuracy of the algorithm is not determined by the number of indistinguishable states, but by the topology of the iCGS and the structure of the formula under exam. In order to have pseudo-realistic results, the automatically generated iCGSs varied over the number of states, the complexity of the formula to analyse, and the number of transitions among states. More specifically, not all iCGSs were generated completely randomly, but a subset of them was generated considering more interesting topologies; that is, iCGSs obtained by modifying the iCGS presented in the rover's example by adding and/or removing new states and/or transitions. This contributed to have more realistic iCGSs, and consequently, more realistic results. The results obtained by our experiments using our procedure are encouraging. Unfortunately, no benchmark of existing iCGSs – to test our tool on – exists, thus these results may vary on more realistic scenarios. Nonetheless, considering the large set of iCGSs we experimented on, we do not expect substantial differences.

6 CONCLUSIONS AND FUTURE WORK

In this paper, we proposed a procedure to overcome the issue in employing logics for strategic reasoning in the context of MAS under perfect recall and imperfect information, a problem in general undecidable. Specifically, we showed how to generate the sub-models in which the verification of strategic objectives is decidable and then used CTL* model checking to provide a verification result. We proved that the entire procedure is in the same complexity class of ATL^*_{IR} model checking. We also implemented our procedure by using MCMAS and provided encouraging experimental results.

In future work, we intend to extend our procedure to increase the types of games and specifications that we can cover. In fact, we recall that our procedure is sound but not complete. It is not possible to find a complete method since, in general, ATL model checking with imperfect information and perfect recall strategies is undecidable. In this work, we considered how to remove the imperfect information from the models to find decidability, but it would be interesting to consider the other direction that produces undecidability, *i.e.* the perfect recall strategies. In particular, we could remove the perfect recall strategies to generate games with imperfect information and imperfect recall strategies, a decidable problem, and then use CTL model checking to provide a verification result. Additionally, we plan to extend our techniques to more expressive languages for strategic reasoning like Strategy Logic [19].

[4] https://github.com/AngeloFerrando/StrategyCTL
[5] https://vas.doc.ic.ac.uk/software/mcmas/
[6] In such experiments, the maximum execution time observed reaches even 3 hours (naturally, this does not only depend on the size of the model, but on the kind of formula verified as well).

REFERENCES

[1] Rajeev Alur, Thomas A. Henzinger, and Orna Kupferman. 2002. Alternating-time temporal logic. *J. ACM* 49, 5 (2002), 672–713. https://doi.org/10.1145/585265.585270

[2] Christel Baier and Joost-Pieter Katoen. 2008. *Principles of Model Checking (Representation and Mind Series).*

[3] Francesco Belardinelli, Angelo Ferrando, and Vadim Malvone. 2023. An abstraction-refinement framework for verifying strategic properties in multi-agent systems with imperfect information. *Artif. Intell.* 316 (2023). https://doi.org/10.1016/j.artint.2022.103847

[4] Francesco Belardinelli, Alessio Lomuscio, and Vadim Malvone. 2018. Approximating Perfect Recall When Model Checking Strategic Abilities. In *Principles of Knowledge Representation and Reasoning: Proceedings of the Sixteenth International Conference, KR 2018, Tempe, Arizona, 30 October - 2 November 2018*, Michael Thielscher, Francesca Toni, and Frank Wolter (Eds.). AAAI Press, 435–444. https://aaai.org/ocs/index.php/KR/KR18/paper/view/18010

[5] Francesco Belardinelli, Alessio Lomuscio, and Vadim Malvone. 2019. An Abstraction-Based Method for Verifying Strategic Properties in Multi-Agent Systems with Imperfect Information. In *The Thirty-Third AAAI Conference on Artificial Intelligence, AAAI 2019, The Thirty-First Innovative Applications of Artificial Intelligence Conference, IAAI 2019, The Ninth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2019, Honolulu, Hawaii, USA, January 27 - February 1, 2019*. AAAI Press, 6030–6037. https://doi.org/10.1609/aaai.v33i01.33016030

[6] Francesco Belardinelli, Alessio Lomuscio, Vadim Malvone, and Emily Yu. 2022. Approximating Perfect Recall when Model Checking Strategic Abilities: Theory and Applications. *J. Artif. Intell. Res.* 73 (2022), 897–932. https://doi.org/10.1613/jair.1.12539

[7] Francesco Belardinelli, Alessio Lomuscio, Aniello Murano, and Sasha Rubin. 2017. Verification of Broadcasting Multi-Agent Systems against an Epistemic Strategy Logic. In *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI 2017, Melbourne, Australia, August 19-25, 2017*, Carles Sierra (Ed.). ijcai.org, 91–97. https://doi.org/10.24963/ijcai.2017/14

[8] Francesco Belardinelli, Alessio Lomuscio, Aniello Murano, and Sasha Rubin. 2017. Verification of Multi-agent Systems with Imperfect Information and Public Actions. In *Proceedings of the 16th Conference on Autonomous Agents and MultiAgent Systems, AAMAS 2017, São Paulo, Brazil, May 8-12, 2017*, Kate Larson, Michael Winikoff, Sanmay Das, and Edmund H. Durfee (Eds.). ACM, 1268–1276. http://dl.acm.org/citation.cfm?id=3091301

[9] Francesco Belardinelli and Vadim Malvone. 2020. A Three-valued Approach to Strategic Abilities under Imperfect Information. In *Proceedings of the 17th International Conference on Principles of Knowledge Representation and Reasoning, KR 2020, Rhodes, Greece, September 12-18, 2020*, Diego Calvanese, Esra Erdem, and Michael Thielscher (Eds.). 89–98. https://doi.org/10.24963/kr.2020/10

[10] Francesco Belardinelli, Vadim Malvone, and Abbas Slimani. 2020. *A Tool for Verifying Strategic Properties in MAS with Imperfect Information.* https://github.com/VadimMalvone/A-Tool-for-Verifying-Strategic-Properties-in-MAS-with-Imperfect-Information

[11] Raphaël Berthon, Bastien Maubert, Aniello Murano, Sasha Rubin, and Moshe Y. Vardi. 2021. Strategy Logic with Imperfect Information. *ACM Trans. Comput. Log.* 22, 1 (2021), 5:1–5:51. https://doi.org/10.1145/3427955

[12] Catalin Dima and Ferucio Laurentiu Tiplea. 2011. Model-checking ATL under Imperfect Information and Perfect Recall Semantics is Undecidable. *CoRR* abs/1102.4225 (2011). arXiv:1102.4225 http://arxiv.org/abs/1102.4225

[13] Ronald Fagin, Joseph Y. Halpern, Yoram Moses, and Moshe Vardi. 1995. *Reasoning about Knowledge.* MIT.

[14] Angelo Ferrando and Vadim Malvone. 2021. Strategy RV: A Tool to Approximate ATL Model Checking under Imperfect Information and Perfect Recall. In *AAMAS '21: 20th International Conference on Autonomous Agents and Multiagent Systems, Virtual Event, United Kingdom, May 3-7, 2021*, Frank Dignum, Alessio Lomuscio, Ulle Endriss, and Ann Nowé (Eds.). ACM, 1764–1766. https://doi.org/10.5555/3463952.3464230

[15] Angelo Ferrando and Vadim Malvone. 2021. Towards the Verification of Strategic Properties in Multi-Agent Systems with Imperfect Information. *CoRR* abs/2112.13621 (2021). arXiv:2112.13621 https://arxiv.org/abs/2112.13621

[16] Angelo Ferrando and Vadim Malvone. 2022. Towards the Combination of Model Checking and Runtime Verification on Multi-agent Systems. In *Advances in Practical Applications of Agents, Multi-Agent Systems, and Complex Systems Simulation. The PAAMS Collection - 20th International Conference, PAAMS 2022, L'Aquila, Italy, July 13-15, 2022, Proceedings (Lecture Notes in Computer Science, Vol. 13616)*, Frank Dignum, Philippe Mathieu, Juan Manuel Corchado, and Fernando de la Prieta (Eds.). Springer, 140–152. https://doi.org/10.1007/978-3-031-18192-4_12

[17] Patrick Gardy and Yuxin Deng. 2019. Simulations for Multi-Agent Systems with Imperfect Information. In *Formal Methods and Software Engineering - 21st International Conference on Formal Engineering Methods, ICFEM 2019 (Lecture Notes in Computer Science, Vol. 11852)*, Yamine Aït Ameur and Shengchao Qin (Eds.). Springer, 138–153.

[18] Wojciech Jamroga and Wiebe van der Hoek. 2004. Agents that Know How to Play. *Fundam. Informaticae* 63, 2-3 (2004), 185–219. http://content.iospress.com/articles/fundamenta-informaticae/fi63-2-3-05

[19] Fabio Mogavero, Aniello Murano, Giuseppe Perelli, and Moshe Y. Vardi. 2014. Reasoning About Strategies: On the Model-Checking Problem. *ACM Trans. Comput. Log.* 15, 4 (2014), 34:1–34:47. https://doi.org/10.1145/2631917

[20] NASA. 2012. *Mars Curiosity Rover.* https://mars.nasa.gov/msl/home/