Journal of Artificial Intelligence Research 82 (2025) 777-817 Submitted 09/2024; published 02/2025

Reasoning about Decidability of Strategic Logics with Imperfect Information and Perfect Recall Strategies

Davide Catta CATTA@LIPN.UNIV-PARIS13.FR
Université Sorbonne Paris Nord, CNRS,
Laboratoire d'Informatique de Paris Nord, LIPN, F-93430
Villetaneuse, France

Angelo Ferrando ANGELO.FERRANDO@UNIMORE.IT
University of Modena and Reggio Emilia
Modena, Italy

Vadim Malvone VADIM.MALVONE@TELECOM-PARIS.FR
Télécom Paris, Institut Polytechnique de Paris
Palaiseau, France

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 the combination of two techniques 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. In case the previous step does not give a final result, we exploit a runtime verification mechanism to provide some intermediate result. 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.

1. Introduction

Multi-Agent Systems (MAS) can be seen as a set of rational entities capable of proactively decide how to act to fulfill their own goals. These entities, called generally agents, are autonomous, in the sense that they do not expect input from a user to act, and social: they can communicate amongst each other to achieve common goals.

Systems are not easy to trust in general. Because of this, we need verification techniques to confirm that such systems behave as expected. More specifically, in the case of MAS, it is relevant to know whether the agents are capable of achieving their own goals, by themselves or by collaborating with other agents by forming a coalition. This is usually referred to as the process of finding a strategy for the agent(s).

A well-known formalism for reasoning about strategic behaviors in MAS is ATL* (Alternating-time Temporal Logic) (Alur, Henzinger, & Kupferman, 2002). In fact, this logic permits to express complex statements about what coalitions of agents can achieve their objectives via strategic behaviors. Given a mathematical model representing the structure of the MAS (usually a directed labeled graph), one can first use the logic ATL* to specify properties of interest, then exploit model-checking techniques to verify whether (the mathematical representation of) the MAS satisfies such properties.

However, one should be careful. Before verifying ATL* specifications, two questions need to be answered:

i) *Given two possible evolutions of the system, is it possible for each agent to distinguish them?*

ii) *Given an objective φ, do agents need to keep in mind the past evolution of the system to determine the current action in order to realize φ?*

If (i) does not admit a positive answer, then we say that the agents have *imperfect information* about the system. The agents have *perfect information* otherwise. To capture the imperfect information setting, the mathematical model of the system will be equipped with indistinguishability relations (one for each agent) over the set of vertex of the model. Moreover, it will be assumed that agents behave in the same way in indistinguishable states of the model.

If (ii) admits a positive answer, then we say that agents need to use *perfect recall* strategies. If agents do not need to remember the past evolution of the system to determine their course of action, we say that agents use *imperfect recall* strategies.

Note that, it is often the case that agents have both imperfect information about the system and they need perfect recall strategies in order to achieve their goal. Unfortunately, given a mathematical representation $\mathfrak{M}$ of the system and an ATL* formula φ expressing the existence of a strategy for the agents to verify their goal, we cannot, in general, decide whether φ holds in $\mathfrak{M}$. In fact, it has been proved that the model-checking problem for ATL* specifications, under the assumption that agents have perfect recall and imperfect information about the model, is undecidable (Dima & Tiplea, 2011). Nonetheless, decidable fragments exist. Indeed, model checking ATL* under perfect information is 2EXPTIME-complete (Alur et al., 2002), while under imperfect information and imperfect recall is PSPACE-complete (Schobbens, 2003).

Given the relevance of the imperfect information setting, even partial solutions to the problem are useful. In this contribution, we precisely detail a methodology that can be used to give a partial solution to the problem. The main idea behind our methodology is to restrict the attention to particular sub-models of the initial given model $\mathfrak{M}$ in which agents have perfect information, and then use Computation Tree Logic (CTL* for short) and Runtime Verification (RV for short) to verify (an abstraction of) the ATL* property expressing the agents' goal.

With more detail, given a model $\mathfrak{M}$ and an ATL* specification φ, our procedure generates a set of sub-models of $\mathfrak{M}$ in which there is perfect information and each of these sub-models satisfies a sub-formula ψ of φ. Then, we use CTL* model checking to check whether: (i) the universal remaining part of φ is satisfied, (ii) the existential remaining part of φ is not satisfied. If this is the case, then we can conclude based on our preservation results. Naturally, since this problem is undecidable, it may happen that neither (i) nor (ii) hold (as we will thoroughly explain in the rest of the paper). In such a case, since the static verification step does not yield a verdict, our procedure may continue with the runtime verification step where runtime monitors are used to check if the remaining part of φ can be satisfied at execution time. If this is the case, we can terminate at runtime the satisfaction of φ for the corresponding system execution. This is determined by the fact that the system has been observed behaving as expected, since it has verified at design time the sub-formula ψ of φ, and at runtime the remaining temporal part of φ (which consists in the part left to verify in φ, not covered by ψ). Note that, the RV step does not imply that the system satisfies φ, indeed future executions may violate φ. The formal result over φ only concerns the current system execution, and how it has behaved in it. However, we will present preservation results on the initial model checking problem

of φ on the model of the system $\mathfrak{M}$, as well. This will be obtained by linking the result obtained at runtime, with its static counterpart. Hence, we are going to show how the satisfaction (resp., violation) of φ at runtime in our approach can be propagated to the verification question over φ on model $\mathfrak{M}$.

Since the original problem is undecidable, we cannot guarantee that the truth or falsity of the property can be established. In other word, 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 still bounded to the class of complexity of ATL* model checking under perfect information and perfect recall.

Our contribution The present work aggregates, harmonizes, and expands on works that have already appeared. In particular, the (partial) solution to the problem here presented is obtained by harmoniously combining and expanding those presented in (Ferrando & Malvone, 2023) and (Ferrando & Malvone, 2022) while in (Ferrando & Malvone, 2021), a demonstration paper presenting a part of the tool deriving by (Ferrando & Malvone, 2022) may be found. This partial solution is based on the idea that, we can approximate results about satisfaction of formulas in a given model with imperfect information, by focusing on sub-models of the given model in which the information is perfect. By doing so, we obtain a more satisfactory solution to the approximation problem of model checking ATL* specifications under imperfect information and perfect recall. With more details: the CTL* verification procedure presented in (Ferrando & Malvone, 2023) and the RV procedure presented in (Ferrando & Malvone, 2022) became here two distinct phases of the same procedure that are executed one after the other. Most of the propositions and theorems present in the contribution already appear in the two aforementioned works. However, proofs for such theoretical results are there not presented. We fill this gap by providing detailed proofs for each of these theorems and propositions. Specifically, in Section 2 we define our logic in Negation Normal Form (NNF) and provide new preliminary definitions (see Definition 4-5) and Proposition 1. Then, in Section 3, we extend the use case by providing additional details and the formal modeling. Section 4 is mostly new, in the previous published versions only the definitions of negative and positive sub-models and the statements of Lemma 2 and 3 were given. In Section 5, the Subsections 5.1 and 5.2 are completely new. Furthermore, Algorithm 7 and the proof of Lemma 4 are new contributions. In Subsection 5.7, we provide the proof of Lemma 5 and the example as new contributions. In Subsection 5.8, we provide the algorithm that integrates the two techniques. In addition, notice that all the proofs of the theorems for the complexities results are new. Finally, for the experimental part, we harmonize the results given in the previous works and provide a new benchmarks section to compare our approach with the existing tools provided in (Belardinelli, Lomuscio, Malvone, & Yu, 2022; Belardinelli, Ferrando, & Malvone, 2023).

2. Preliminaries

In this section, we introduce the main formal tools that we use in the rest of the paper. In particular, we introduce Concurrent Game Structures with imperfect information, the syntax, and semantics of the logic ATL*, and Runtime Verification. First, let us introduce some notation that will be used along the paper.

Notation If X is a set, and $Y \subseteq X$ we denote by $\overline{Y}$ its complement $X \setminus Y$. If X is a set, then X^+ denotes the set of non-empty finite sequences over X, and X^ω denotes the set of infinite sequences

over X. If ρ is a (infinite or finite) sequence over X, we denote by $|\rho|$ its length (which is ω when ρ is infinite). Given a natural number $1 \leq i \leq |\rho|$, we let ρ_i denote the i-th element of the sequence, $\rho_{\leq_i}$ denote the prefix $\rho_1, \ldots \rho_i$ of ρ and $\rho_{\geq i}$ denote the suffix of ρ starting at ρ_i. If $\rho = \rho_1, \ldots, \rho_n$ is a finite sequence $last(\rho)$ denotes the last element ρ_n of ρ. Furthermore, given two sequences ρ and ρ', with $\rho \cdot \rho'$ we denote the concatenation of the two. We write $\rho \sqsubseteq \pi$ when ρ is a prefix of π and $\rho \sqsubset \pi$ when ρ is a strict prefix of π.

2.1 Concurrent Game Structures with Imperfect Information

We can now formally define our models for MAS.

Definition 1. A concurrent game structure with imperfect information (iCGS for short) is a tuple $\mathfrak{M} = \langle \mathtt{Ag}, \mathtt{Ap}, \mathtt{S}, \mathtt{s}_I, \{\mathtt{Act}_i\}_{i \in \mathtt{Ag}}, \{\sim_i\}_{i \in \mathtt{Ag}}, d, \delta, \mathtt{V} \rangle$ where:

- $\mathtt{Ag} = \{1, \ldots, m\}$ is a finite, non-empty set of agents;
- $\mathtt{Ap}$ is a finite, non-empty set of atomic propositions (or atoms);
- $\mathtt{S}$ is a finite, non-empty set of states, with initial state $\mathtt{s}_I \in \mathtt{S}$;
- $\mathtt{Act}_i$ is a finite non-empty set of actions for all $i \in \mathtt{Ag}$; we denote by $\mathtt{Act} = \cup_{i \in \mathtt{Ag}} \mathtt{Act}_i$ and by $\mathtt{ACT} = \Pi_{i \in \mathtt{Ag}} \mathtt{Act}_i$. Elements of this latter set will be called *joint actions*;
- $\sim_i \subset \mathtt{S} \times \mathtt{S}$ is an equivalence relation between states, for all $i \in \mathtt{Ag}$;
- $d : \mathtt{Ag} \times \mathtt{S} \to 2^{act_i}$ is the protocol function mapping an agent i and a state to a non-empty subset of $\mathtt{Act}_i$. Moreover, for any two states s, s' such that $s \sim_i s$, $d(i, s) = d(i, s')$;
- $\delta : \mathtt{S} \times \mathtt{ACT} \to \mathtt{S}$ is the transition function, assigning to any state s and joint-action $\vec{a}$, such that $a_i \in d(i, s)$, a successor state s';
- $\mathtt{V} : \mathtt{S} \to 2^{\mathtt{Ap}}$ is the labeling function, assigning to any state s the set of atomic propositions holding in s.

By Definition 1 an iCGS describes the interactions of a group $\mathtt{Ag}$ of agents, starting from the initial state $\mathtt{s}_I \in \mathtt{S}$, 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 of the exact state of the game; so in any state s, the agent i considers epistemically possible all states s' that are i-indistinguishable from s (Fagin, Halpern, Moses, & Vardi, 1995). 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 (Alur et al., 2002).

Given an iCGS $\mathfrak{M}$, a *path* $\rho \in \mathtt{S}^\omega$ is an infinite sequence of states $\rho = s_1, s_2, \ldots$ such that for all $i \geq 1$, $\delta(s_i, \vec{a}) = s_{i+1}$ for some joint action $\vec{a}$. A *history* h is a finite sequence of states that is a prefix of some path ρ. We use H to denote the set of all histories. Given two histories h, h' and an agent i, we say that h is indistinguishable from h' for agent i (denote by $h \sim_i h$) iff $|h| = |h'| = n \in \mathbb{N}$ and for all $i \leq n$, $h_i \sim_i h'_i$.

We assume that agents employ *uniform strategies* (Jamroga & van der Hoek, 2004), *i.e.*, they perform the same action whenever they have the same information.

Definition 2. A *uniform strategy* for agent $i \in \mathtt{Ag}$ is a function $\sigma_i : \mathtt{S}^+ \to \mathtt{Act}_i$ such that for all histories h and h':

i) $\sigma_i(h) \in d(i, last(h))$ and,

ii) $h \sim_i h'$ implies $\sigma_i(h) = \sigma_i(h')$.

A strategy σ_i for i is said to be an imperfect recall (or memoryless) strategy whenever for every pair of histories h and h' we have that $last(h) = last(h')$ implies $\sigma_i(h) = \sigma_i(h')$. As it is usual, one can see a memoryless strategy σ_i as a function from $\mathtt{S}$ to $\mathtt{Act}_i$.

By Definition 2 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.

Given a joint strategy Σ_Γ, comprising of one strategy for each agent in $\Gamma \in \mathtt{Ag}$, a path ρ is Σ_Γ-*compatible* iff for every $j \geq 1$, $\rho_{j+1} = \delta(\rho_j, \vec{a})$ for some joint action $\vec{a}$ such that for every $i \in \Gamma$, $a_i = \sigma_i(\rho_{\leq j})$, and for every $i \in \overline{\Gamma}$, $a_i \in d(i, p_j)$. We denote with $out(s, \Sigma_\Gamma)$ the set of all Σ_Γ-compatible paths from s.

2.2 Alternating-time Temporal Logic

In this subsection, we expose the logic ATL* by recalling its syntax and semantics. We present the syntax of ATL* in negation normal form *i.e.*, the negation operator will be only applied to atomic propositions. This choice will greatly simplify the formal machinery deployed in the rest of the paper.

Definition 3. Given a non-empty countable set $\mathtt{Ap}$ of atomic propositions (or atoms) and a non-empty finite set $\mathtt{Ag}$ of agents, state (φ) and path (ψ) formulas in ATL* are defined by mutual induction according to the following grammar:

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

where $q \in \mathtt{Ap}$ and $\Gamma \subseteq \mathtt{Ag}$.

In what follows, we will use the letters p, q, and r (possibly indexed) to denote arbitrary atoms, the letters φ and ψ (possibly indexed) to denote arbitrary state and path formulas, and the letters θ and λ (possibly indexed) to denote arbitrary formulas. We will also use the term *coalitions* to refer to subsets of the set of agents. Negation can be extended to all formulas using the following De Morgan's equations:

$$\begin{array}{lll} \neg\neg q = q & \neg(\theta \wedge \lambda) = (\neg\theta \vee \neg\lambda) & \neg(\theta \vee \lambda) = (\neg\theta \wedge \neg\lambda) \\ \neg(\langle\!\langle \Gamma \rangle\!\rangle \psi) = ([\![\Gamma]\!] \neg\psi) & \neg([\![\Gamma]\!] \psi) = (\langle\!\langle \Gamma \rangle\!\rangle \neg\psi) & \\ \neg(\mathsf{X}\psi) = \mathsf{X}(\neg\psi) & \neg(\psi_1 \,\mathsf{U}\, \psi_2) = (\neg\psi_1 \,\mathsf{R}\, \neg\psi_2) & \neg(\psi_1 \,\mathsf{R}\, \psi_2) = (\neg\psi_1 \,\mathsf{U}\, \neg\psi_2) \end{array}$$

As usual, a formula $\langle\!\langle \Gamma \rangle\!\rangle \psi$ is read as "the agents in coalition Γ have a strategy to achieve ψ". The meaning of temporal connectives *next* X, *until* U, and *release* R is standard (Baier & Katoen, 2008). The connectives *eventually* F and *globally* G can be defined as usual.

Formulas in the ATL fragment of ATL* are obtained from Definition 3 by restricting path formulas as follows:

$$\psi \quad ::= \quad \mathsf{X}\varphi \mid (\varphi \,\mathsf{U}\, \varphi) \mid (\varphi \,\mathsf{R}\, \varphi)$$

Definition 4. The size $|\theta|$ of a formula θ, its rank $r(\theta)$, and its set of sub-formulas $sub(\theta)$ are recursively defined as follows:

- if θ is an atom or a negated atom, then $|\theta| = r(\theta) = 0$ and $sub(\theta) = \{\theta\}$;
- if θ is $\theta_1 \star \theta_2$ with $\star \in \{\wedge, \vee, \mathsf{U}, \mathsf{R}\}$, then $|\theta| = \mathtt{max}(|\theta_1|, |\theta_2|) + 1$, $r(\theta) = \mathtt{max}(r(\theta_1), r(\theta_2))$ and $sub(\theta) = sub(\theta_1) \cup sub(\theta_2) \cup \{\theta_1 \star \theta_2\}$;
- if θ is $\circ\theta_1$ with $\circ \in \{\mathsf{X}, \langle\!\langle\Gamma\rangle\!\rangle, [\![\Gamma]\!]\}$, then $|\theta| = |\theta_1| + 1$, $r(\theta) = r(\theta_1)$ if $\circ = \mathsf{X}$ and $r(\theta) = r(\theta_1) + 1$ otherwise, and $sub(\theta) = sub(\theta_1) \cup \{\theta\}$.

Remark that, we consider that a formula can be seen as a tree in which leaf are labeled by atoms or negated atoms and in which non-leaf nodes are labeled by connectives. Intuitively, the size of a formula is the height of this tree. This intuition will be formalized in the definition of formula tree that will follow. The rank of a formula is the maximum number of nestings of strategic operators of the formula.

Given a formula θ, we say that:

- θ is negation-free whenever none of its sub-formulas is a negated atom $\neg p$;
- θ is a CTL* formula whenever all its strategic subformulas are of the form $\langle\!\langle X\rangle\!\rangle\psi$, where $X \in \{\mathtt{Ag}, \emptyset\}$. For a CTL* formula we will write $\mathsf{E}\psi$ instead of $\langle\!\langle \mathtt{Ag}\rangle\!\rangle\psi$ and $\mathsf{A}\psi$ instead of $\langle\!\langle\emptyset\rangle\!\rangle\psi$;
- θ is an LTL formula whenever none of its sub-formulas is a strategic formula $\langle\!\langle\Gamma\rangle\!\rangle\psi$ or $[\![\Gamma]\!]\psi$ (*i.e.*, $r(\theta) = 0$).

Before specifying the semantics of ATL* formulas, let us define a concept that will be useful in the following. We recall, that a directed tree is a directed connected graph $\mathcal{T} = \langle V, \rightarrow\rangle$ in which there is a node r (the root) such that $\langle v', r\rangle \notin \rightarrow$ for every $v \in V$ and given any other node $v \in V$ there is exactly one node v' such that $\langle v', v\rangle \in \rightarrow$.

Definition 5. A *formula tree* is a finite directed tree $\mathcal{T}$ in which nodes are labeled with either atomic propositions, negated atomic propositions, or ATL* connectives and edges are labeled with elements from $\{0, 1, 2\}$, such that:

- each leaf of $\mathcal{T}$ is labeled with either an atomic proposition or a negated atomic proposition;
- each internal node is labeled with an ATL* connective;
- if v is a node of $\mathcal{T}$ and v is labeled with an ATL* binary connective, then v has exactly two children. One of v outgoings edges is labeled with 1, while the other is labeled with 2;
- if v is a node of $\mathcal{T}$ and v is labeled with an ATL* unary connective, then v has exactly one child and the edge going from v to this child is labeled with 0.

Given a formula θ, one can define the tree $\mathcal{T}_\theta$ by induction on $|\theta|$, as follows:

- if θ is an atomic proposition or a negated atomic proposition, then $\mathcal{T}_\theta$ is the tree composed of just one node labeled with θ itself;
- if $\theta = \theta_1 \star \theta_2$ with $\star$ a binary connective, then $\mathcal{T}_\theta$ is the directed tree in which the root r is labeled with $\star$, the set of nodes only contains r and all nodes of $\mathcal{T}_{\theta_1}$ and $\mathcal{T}_{\theta_2}$, and the set of edges only contains all edges of $\mathcal{T}_{\theta_1}$, all edges of $\mathcal{T}_{\theta_2}$, an edge from r to the root of $\mathcal{T}_{\theta_1}$ (labeled with 1) and an edge from r to the root of $\mathcal{T}_{\theta_2}$ (labeled with 2);
- if $\theta = \circ\theta_1$ with $\circ$ a unary connective, then $\mathcal{T}_\theta$ is the directed tree in which the root r is labeled with $\circ$, the set of nodes only contains r and all nodes of $\mathcal{T}_{\theta_1}$, and the set of edges only contains all edges of $\mathcal{T}_{\theta_1}$ and an outgoing edge from r to the root of $\mathcal{T}_{\theta_1}$ that is labeled with 0.

In Figure 1, we display an example of formula tree.

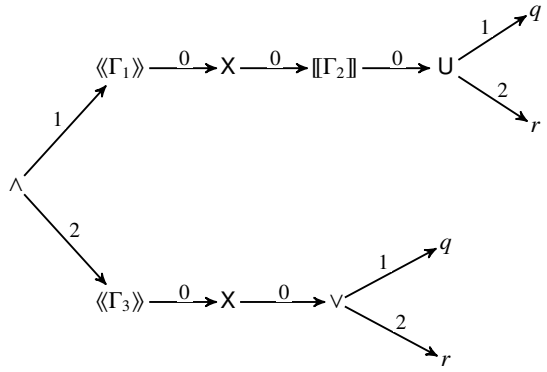

Figure 1: Formula tree for the formula $(\langle\langle\Gamma_1\rangle\rangle \mathsf{X}\, [\![\Gamma_2]\!]\, q \,\mathsf{U}\, r) \wedge (\langle\langle\Gamma_3\rangle\rangle \mathsf{X}\, (q \vee r))$.

It is straightforward to see that for each formula θ, $\mathcal{T}_\theta$ is a formula tree in the sense of Definition 5. Also, the following proposition can be easily proved by induction on the number of nodes of the considered tree.

Proposition 1. *If $\mathcal{T}$ is a formula tree, then there is a unique formula θ such that $\mathcal{T} = \mathcal{T}_\theta$.*

Consequently, in the following we will feel free to equivalently use a representation of formulas as trees or as strings of symbols according to our convenience.

Now, we have all the ingredients to assign a meaning to ATL* formulas on iCGS.

Definition 6. Let $\mathfrak{M} = \langle \mathtt{Ag}, \mathtt{Ap}, \mathtt{S}, \mathtt{s}_I, \{\mathtt{Act}_i\}_{i\in\mathtt{Ag}}, \{\sim_i\}_{i\in\mathtt{Ag}}, d, \delta, \mathtt{V}\rangle$ be an iCGS. The satisfaction relation $\models$ is defined by mutual induction on state and path formulas as follows. For state formulas at a state $s \in \mathtt{S}$:

- $(\mathfrak{M}, s) \models q$ iff $q \in \mathtt{V}(s)$
- $(\mathfrak{M}, s) \models \neg q$ iff $q \notin \mathtt{V}(s)$
- $(\mathfrak{M}, s) \models \varphi_1 \wedge \varphi_2$ iff $(\mathfrak{M}, s) \models \varphi_1$ and $(\mathfrak{M}, s) \models \varphi_2$

- $(\mathfrak{M}, s) \models \varphi_1 \vee \varphi_2$ iff $(\mathfrak{M}, s) \models \varphi_1$ or $(\mathfrak{M}, s) \models \varphi_2$
- $(\mathfrak{M}, s) \models \langle\!\langle \Gamma \rangle\!\rangle \psi$ iff there is a strategy Σ_Γ such that for all $\rho \in out(s, \Sigma_\Gamma)$ we have that $(\mathfrak{M}, \rho) \models \psi$
- $(\mathfrak{M}, s) \models [\![\Gamma]\!] \psi$ iff for every strategy Σ_Γ there is a path $\rho \in out(s, \Sigma_\Gamma)$ such that $(\mathfrak{M}, \rho) \models \psi$

For path formulas, given a path $\rho \in \mathsf{S}^\omega$:

- $(\mathfrak{M}, \rho) \models \varphi$ iff $(\mathfrak{M}, \rho_1) \models \varphi$
- $(\mathfrak{M}, \rho) \models \psi_1 \wedge \psi_2$ iff $(\mathfrak{M}, \rho) \models \psi_1$ and $(\mathfrak{M}, \rho) \models \psi_2$
- $(\mathfrak{M}, \rho) \models \psi_1 \vee \psi_2$ iff $(\mathfrak{M}, \rho) \models \psi_1$ or $(\mathfrak{M}, \rho) \models \psi_2$
- $(\mathfrak{M}, \rho) \models \mathsf{X}\psi$ iff $(\mathfrak{M}, \rho_{\geq 2}) \models \psi$
- $(\mathfrak{M}, \rho) \models \psi_1 \mathsf{U} \psi_2$ iff $(\mathfrak{M}, \rho_{\geq k}) \models \psi_2$, for some $k \geq 1$, and $(\mathfrak{M}, \rho_j) \models \psi_1$, for every $1 \leq j < k$
- $(\mathfrak{M}, \rho) \models \psi_1 \mathsf{R} \psi_2$ iff $(\mathfrak{M}, \rho_{\geq i}) \models \psi_2$, for all $i \geq 1$, or there is a $k \geq 1$ such that $(\mathfrak{M}, \rho_{\geq k}) \models \psi_1$ and, for all $1 \leq j \leq k$, $(\mathfrak{M}, \rho_{\geq j}) \models \psi_2$

We say that formula φ is *true* in an iCGS $\mathfrak{M}$, or $\mathfrak{M} \models \varphi$, iff $(\mathfrak{M}, \mathsf{s}_I) \models \varphi$. We can now state the model checking problem.

Definition 7 (Model Checking)**.** Given an iCGS $\mathfrak{M}$ and a formula φ, the model checking problem concerns determining whether $\mathfrak{M} \models \varphi$.

Remark 1. The imperfect recall (or memoryless) satisfaction relation $\models_{ir}$ is obtained by writing "imperfect recall strategy" instead of "strategy" in Definition 6 for the strategic operators $\langle\!\langle \Gamma \rangle\!\rangle$ and $[\![\Gamma]\!]$.

2.3 Runtime Verification

Runtime Verification (RV from now on) is a lightweight formal verification technique which focuses on the analysis of the runtime behavior of a system (Bartocci, Falcone, Francalanza, & Reger, 2018; Leucker & Schallhart, 2009). Differently from other verification techniques, RV is not exhaustive, since it focuses on the actual system execution. That is, a violation of the expected behavior is concluded only if such violation is observed. Nevertheless, RV does not check all possible system's behaviors, and so, it scales better than its static verification counterparts, which usually suffer from the state space explosion problem. The standard formalism to specify formal properties in RV is Linear Temporal Logic (LTL) (Pnueli, 1977). We now define what a monitor is, and how it works. Informally, a monitor is a function that, given a history in input, returns a verdict which denotes the satisfaction (resp., violation) of a formal property.

Definition 8 (Monitor)**.** Let $\mathfrak{M}$ be an iCGS and ψ be an LTL property. Then, a monitor for ψ is a function $Mon_\psi^{\mathfrak{M}} : S^+ \to \mathbb{B}_3$, where $\mathbb{B}_3 = \{\top, \bot, ?\}$:

$$Mon_\psi^{\mathfrak{M}}(h) = \begin{cases} \top & \forall \rho \in \mathsf{S}^\omega \ (\mathfrak{M}, h \cdot \rho) \models \psi \\ \bot & \forall \rho \in S^\omega \ (\mathfrak{M}, h \cdot \rho) \not\models \psi \\ ? & \textit{otherwise.} \end{cases}$$

where the path ρ is a valid continuation of the history h in $\mathfrak{M}$.

Intuitively, a monitor returns $\top$ if all continuations of h satisfy ψ; $\bot$ if all continuations of h violate ψ; ? otherwise. The first two outcomes are standard representations of satisfaction and violation, while the third is specific to RV. In more detail, it denotes when the monitor cannot conclude any verdict yet. This is closely related to the fact that RV is applied while the system is still running, and not all information about it are available. For instance, a property might be currently satisfied (resp., violated) by the system, but violated (resp., satisfied) in the (still unknown) future. The monitor can only safely conclude any of the two final verdicts ($\top$ or $\bot$) if it is sure such verdict will never change. The addition of the third outcome symbol ? helps the monitor to represent its uncertainty w.r.t. the current system execution.

3. Use Case Scenario

In this section, we present a variant of the Curiosity rover scenario under imperfect information and perfect recall. 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 (NASA, 2024), 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 2, we model an example of mission for the rover. The rover starts in state $\mathtt{s}_I$, and has to perform a setup action, consisting in checking the three main rover's components: arm, mast, and wheels. To save time and energy, the mechanic (the entity capable of performing the setup checks and making any corrections) can perform only one setup operation per mission. Since such operation is not known by the rover, from its point of view, states s_1, s_2, and s_3 are equivalent, *i.e.*, it cannot distinguish between the three setup operations. After the selection, the mechanic can choose to check and eventually correct the component (action *ok*) or 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 behind the ship. So, its aim is to move from its position, to take 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 take a photo of a sample rock (action tp). After this step, the rover has to conclude the mission by returning behind the ship. 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).

Formally, the model $\mathfrak{M} = \langle \mathtt{Ag}, \mathtt{Ap}, \mathtt{S}, \mathtt{s}_I, \{\mathtt{Act}_i\}_{i\in\mathtt{Ag}}, \{\sim_i\}_{i\in\mathtt{Ag}}, d, \delta, \mathtt{V}\rangle$ of the mission is defined as:

- $\mathtt{Ag} = \{rover, mechanic\}$;
- $\mathtt{Ap} = \{sp, cpa, cpm, cpw, oc, rm, rp, ip, pl, pr\}$;
- $\mathtt{S} = \{\mathtt{s}_I, s_1, s_2, s_3, s_4, s_5, s_6, s_7, s_8, e_1, e_2\}$;
- $\mathtt{Act}_{rover} = \{chk, L, R, tp, i\}$;
- $\mathtt{Act}_{mechanic} = \{ca, cm, cw, ok, nok, i\}$;
- there are the following non-trivial equivalence relations $s_1 \sim_{rover} s_2$, $s_1 \sim_{rover} s_3$, and $s_2 \sim_{rover} s_3$;

- the functions d and δ can be derived by Figure 2;
- the labelling function is defined as: $\mathtt{V}(\mathtt{s}_I) = \{sp\}$, $\mathtt{V}(s_1) = \{cpa\}$, $\mathtt{V}(s_2) = \{cpm\}$, $\mathtt{V}(s_3) = \{cpw\}$, $\mathtt{V}(s_4) = \{oc, rm\}$, $V(s_5) = \{pl\}$, $\mathtt{V}(s_6) = \mathtt{V}(s_7) = \{rp, ip\}$, $V(s_8) = \{pr\}$, and $V(e_1) = \mathtt{V}(e_2) = \emptyset$.

The atomic propositions have the following meaning: *sp* stays for starting position, *cpa* stays for check phase arm, *cpm* stays for check phase mast, *cpw* stays for check phase wheels, *oc* stays for ok check phase, *rm* stays for ready to mission, *rp* stays for ready to take a picture, *ip* stays for in position, *pl* stays for picture left, and *pr* stays for picture right.

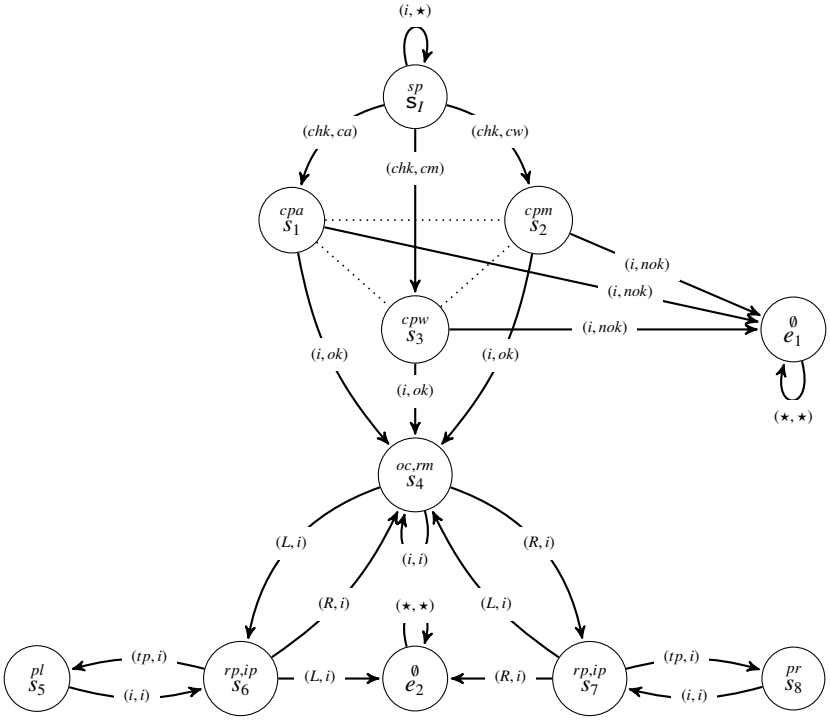

Figure 2: An example of the rover's mission where i stands for the idle action, $\star$ for any action, and the rover's equivalence relation is denoted with the dotted lines.

So, given the model $\mathfrak{M}$, we can define several specifications. For example, the specification that describes the rover mission is

$$\varphi_1 = \langle\langle rover \rangle\rangle \mathsf{F}((oc \wedge rm) \wedge \mathsf{F}((pl \vee pr) \wedge \mathsf{F}(oc \wedge rm)))$$

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 take a picture of a sample rock, and can return behind the ship. 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 memoryful 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

$$\varphi_2 = \langle\langle rover, mechanic\rangle\rangle \mathsf{F}((oc \wedge rm) \wedge \langle\langle rover\rangle\rangle \mathsf{F}((pl \vee pr) \wedge \mathsf{F}(oc \wedge rm)))$$

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_6 and s_7, so it is impossible for it to first make the picture and then go to the end of the mission. While, by using memoryful 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:

- $\sigma(\mathsf{s}_I) = chk$
- $\sigma(\mathsf{s}_I s_j) = i$, where $j \in \{1, 2, 3\}$
- $\sigma(\mathsf{s}_I s_j s_4) = L$, where $j \in \{1, 2, 3\}$
- $\sigma(\mathsf{s}_I s_j s_4 s_6) = tp$, where $j \in \{1, 2, 3\}$
- $\sigma(\mathsf{s}_I s_j s_4 s_6 s_5) = i$, where $j \in \{1, 2, 3\}$
- $\sigma(\mathsf{s}_I s_j s_4 s_6 s_5 s_6) = R$, where $j \in \{1, 2, 3\}$

With the above observations, we can conclude that, to verify the specification φ_2 in the model $\mathfrak{M}$, we need the 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 take a picture but it is not in position to do that and, from that point, if it has the ability to take a picture and return in the starting mission, *i.e.*, we want to verify the formula

$$\varphi_3 = \langle\langle rover, mechanic\rangle\rangle \mathsf{F}((rp \wedge \neg ip) \wedge \langle\langle rover\rangle\rangle \mathsf{F}((pl \vee pr) \wedge \mathsf{F}(oc \wedge rm)))$$

It is easy to see that φ_3 is false in the model $\mathfrak{M}$ because there is not a state in which $rp \wedge \neg ip$. However, to check φ_3 in the model $\mathfrak{M}$, we need the model checking of ATL* in the context of imperfect information and perfect recall, a problem in general undecidable.

In what follows, we present a sound but not complete procedure handling this class of problems, and use the curiosity rover scenario to help the reader during each step.

4. The Formal Background of Our Procedure

In this section, we give formal definitions of the objects that our procedure uses to give a partial solution to the undecidability problem.

4.1 Negative and Positive Models

Our procedure tries to circumvent the problem of imperfect information. To do this, some particular sub-models of the initial model are selected. In this subsection, we define such types of models and show preservation results between these sub-models and the initial model.

We start by presenting the sub-models that under-approximate the original model w.r.t. the satisfaction of ATL* formulas.

Definition 9 (Negative sub-model)**.** Given an iCGS $\mathfrak{M} = \langle \mathtt{Ag}, \mathtt{Ap}, \mathtt{S}, \mathtt{s}_I, \{\mathtt{Act}_i\}_{i\in\mathtt{Ag}}, \{\sim_i\}_{i\in\mathtt{Ag}}, d, \delta, \mathtt{V}\rangle$ a negative sub-model of $\mathfrak{M}$ is an iCGS $\mathfrak{M}^\bullet = \langle \mathtt{Ag}, \mathtt{Ap}, \mathtt{S}^\bullet, \mathtt{s}_I^\bullet, \{\mathtt{Act}_i\}_{i\in\mathtt{Ag}}, \{\sim_i^\bullet\}_{i\in\mathtt{Ag}}, d^\bullet, \delta^\bullet, \mathtt{V}^\bullet\rangle$ where:

- $\mathtt{S}^\bullet = S' \cup \{s_\bot\}$, S' is a subset of $\mathtt{S}$ and $s_\bot$ is a fresh state;
- for any $i \in \mathtt{Ag}$, $\sim_i^\bullet = (\sim_i \cap (S' \times S')) \cup \{(s_\bot, s_\bot)\}$;
- for every state $s \in \mathtt{S}^\bullet$ and every agent $i \in \mathtt{Ag}$, the function $d^\bullet(i, s)$ is equal to $d(i, s)$ if $s \in S'$ and it is equal to $\mathtt{Act}_i$ if $s = s_\bot$;
- for every state $s \in \mathtt{S}^\bullet \setminus \{s_\bot\}$, if $\delta(s, \vec{a}) \in S'$ then $\delta^\bullet(s, \vec{a}) = \delta(s, \vec{a})$, otherwise $\delta^\bullet(s, \vec{a}) = s_\bot$. Moreover, $\delta^\bullet(s_\bot, \vec{a}) = s_\bot$ for each joint action $\vec{a}$;
- for every state $s \in \mathtt{S}^\bullet$, if $s \in S'$ then $\mathtt{V}^\bullet(s) = \mathtt{V}(s)$, otherwise (*i.e.*, $s = s_\bot$) $\mathtt{V}^\bullet(s) = \emptyset$.

Now, we continue by presenting the sub-models that over-approximate the original model w.r.t. the satisfaction of ATL* formulas.

Definition 10 (Positive sub-model)**.** Given an iCGS $\mathfrak{M} = \langle \mathtt{Ag}, \mathtt{Ap}, \mathtt{S}, \mathtt{s}_I, \{\mathtt{Act}_i\}_{i\in\mathtt{Ag}}, \{\sim_i\}_{i\in\mathtt{Ag}}, d, \delta, \mathtt{V}\rangle$ a positive sub-model of $\mathfrak{M}$ is an iCGS $\mathfrak{M}^\circ = \langle \mathtt{Ag}, \mathtt{Ap}, \mathtt{S}^\circ, \mathtt{s}_I^\circ, \{\mathtt{Act}_i\}_{i\in\mathtt{Ag}}, \{\sim_i^\circ\}_{i\in\mathtt{Ag}}, d^\circ, \delta^\circ, \mathtt{V}^\circ\rangle$ where:

- $\mathtt{S}^\circ = S' \cup \{s_\top\}$, S' is a subset of $\mathtt{S}$ and $s_\top$ is a fresh state;
- for any $i \in \mathtt{Ag}$, $\sim_i^\circ = (\sim_i \cap (S' \times S')) \cup \{(s_\top, s_\top)\}$;
- for every state $s \in \mathtt{S}^\circ$ and every agent $i \in \mathtt{Ag}$, the function $d^\circ(i, s)$ is equal to $d(i, s)$ if $s \in S'$ and it is equal to $\mathtt{Act}_i$ if $s = s_\top$;
- for every state $s \in \mathtt{S}^\circ \setminus \{s_\top\}$, for every joint action $\vec{a}$, if $\delta(s, \vec{a}) \in S'$ then $\delta^\circ(s, \vec{a}) = \delta(s, \vec{a})$, otherwise $\delta^\circ(s, \vec{a}) = s_\top$. Moreover, $\delta^\circ(s_\top, \vec{a}) = s_\top$ for every joint action $\vec{a}$;
- for every state $s \in \mathtt{S}^\circ$, if $s \in S'$ then $\mathtt{V}^\circ(s) = \mathtt{V}(s)$, otherwise (*i.e.*, $s = s_\top$) $\mathtt{V}^\circ(s) = \mathtt{Ap}$.

Given an iCGS $\mathfrak{M}$ and an ATL* formula θ, the procedure that we have devised will select positive and negative sub-models of $\mathfrak{M}$ in which we have perfect information and evaluate the formula θ on those models. For the procedure to make sense, we must have preservation results between these two sub-model classes and the original model. Specifically, we would like that when a strategic formula φ is true in a negative sub-model then φ is true in the model $\mathfrak{M}$ as well and, conversely, that when φ is false in a positive sub-model it is also false in the original model. Unfortunately, this is not the case. Consider the model $\mathfrak{M}$ and its negative and positive sub-models $\mathfrak{M}^\bullet$ and $\mathfrak{M}^\circ$ displayed in Figure 3. Suppose that the set of agents of $\mathfrak{M}$ counts just one agent a_1, and that this agent has at its disposal only the action a. We have that, $(\mathfrak{M}^\bullet, s_0) \models \langle\langle a_1\rangle\rangle \mathsf{X} \neg p$ and $(\mathfrak{M}, s_0) \not\models \langle\langle a_1\rangle\rangle \mathsf{X} \neg p$ and that $(\mathfrak{M}^\circ s_0) \not\models \langle\langle a_1\rangle\rangle \mathsf{X} \neg q$ while $(\mathfrak{M}, s_0) \models \langle\langle a_1\rangle\rangle \mathsf{X} \neg q$.
As the reader may have guessed, the failure of the wanted preservation results depends upon the presence of negated atoms in the specification we have used. In fact, the preservation results hold for ATL* formulas containing no negated atoms as the following series of proposition and lemmas show.

Let us fix some terminology; given an iCGS $\mathfrak{M}$ and a path π of $\mathfrak{M}$, we say that:

1. π is $\bot$-*like* iff for any i we have that $\mathtt{V}(\pi_i) = \emptyset$;

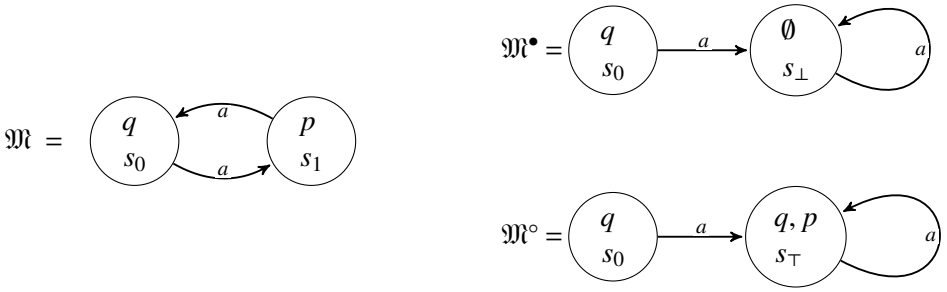

Figure 3: A model and two possible negative and positive sub-models.

2. π is $\top$-*like* iff for any i we have that $\mathtt{V}(\pi_i) = \mathtt{Ap}$.

If σ and π are two paths, we say that σ is a *k-twin* of π iff there is $k \in \mathbb{N}$ such that $\mathtt{V}(\pi_i) = \mathtt{V}(\sigma_i)$ for every $i \leq k$.

Proposition 2. *Let θ be a negation-free LTL formula and $\mathfrak{M}$ an iCGS.*

1. *For every $\perp$-like path π of $\mathfrak{M}$, $(\mathfrak{M}, \pi) \not\models \theta$.*
2. *For every $\top$-like path π of $\mathfrak{M}$, $(\mathfrak{M}, \pi) \models \theta$.*

Proof. We only give an argument for (1) as the one for (2) is completely alike. The proof is by induction on $|\theta|$. If $|\theta| = 0$ then θ is some atomic formula p. By definition of satisfaction $(\mathfrak{M}, \pi) \models p$ iff $(\mathfrak{M}, \pi_1) \models p$ iff $p \in \mathtt{V}(\pi_1)$. Since $\mathtt{V}(\pi_1) = \emptyset$, we conclude that $(\mathfrak{M}, \pi_1) \not\models p$. Suppose that the statement of the proposition holds for any formula whose size is at most n, and consider any formula whose size is $n + 1$. Remark that any suffix of $\perp$-like path is $\perp$-like path. From this observation, the proposition's statement easily follows by induction hypothesis. □

Lemma 1. *Let θ be negation-free LTL formula, $\mathfrak{M}$ an iCGS, and π a path of $\mathfrak{M}$:*

1. *Suppose that $\pi_{\geq j}$ is $\perp$-like for some $j \geq 1$. If $(\mathfrak{M}, \pi_{\geq k}) \models \theta$ then $k \leq j$ and for every iCGS $\mathfrak{N}$ and every path σ of $\mathfrak{N}$ such that σ is $j - 1$-twin of π we have that $(\mathfrak{N}, \sigma_{\geq k}) \models \theta$.*
2. *Suppose that $\pi_{\geq j}$ is $\top$-like for some $j \geq 1$. If $(\mathfrak{M}, \pi_{\geq k}) \not\models \theta$ then $k \leq j$ and for every iCGS $\mathfrak{N}$ and every path σ of $\mathfrak{N}$ such that σ is $j - 1$-twin of π we have that $(\mathfrak{N}, \sigma_{\geq k}) \not\models \theta$.*

Proof. We only give an argument for (1) as the one for (2) is completely alike. The proof is by induction on $|\theta|$. If $|\theta| = 0$ then $(\mathfrak{M}, \pi_{\geq k}) \models p$ iff $(\mathfrak{M}, \pi_k) \models p$ iff $p \in \mathtt{V}(\pi_k)$. Remark that this implies $k < j$. Consider any model $\mathfrak{N}$ and any path σ of $\mathfrak{N}$ such that that σ is $j - 1$ twin of π. Since $1 < j$, we have that $p \in \mathtt{V}(\sigma_k)$ and thus $(\mathfrak{N}, \sigma_{\geq k}) \models p$ as desired. Suppose that the lemma's statement holds for any formula whose size is at most n and let λ be a formula whose size is $n + 1$. Fix a path π of $\mathfrak{M}$, an iCGS $\mathfrak{N}$, and a path σ of $\mathfrak{N}$ that is a $j - 1$-twin of π. We detail some cases.

- If $\lambda = \lambda_1 \wedge \lambda_2$ then $(\mathfrak{M}, \pi_{\geq k}) \models \lambda_1 \wedge \lambda_2$ iff $(\mathfrak{M}, \pi_{\geq k}) \models \lambda_i$ for $i \in \{1, 2\}$. By Proposition 2, we deduce that $k < j$. We thus apply the induction hypothesis, and we obtain that $(\mathfrak{N}, \sigma_{\geq k}) \models \lambda_i$ for $i \in \{1, 2\}$, thus $(\mathfrak{N}, \sigma_{\geq k}) \models \lambda_1 \wedge \lambda_2$.

- if $\lambda = \lambda_1 \mathsf{U} \lambda_2$ then $(\mathfrak{M}, \pi_{\geq k}) \models \lambda_1 \mathsf{U} \lambda_2$ iff there is $r \geq 1$ such that $(\mathfrak{M}, \pi_{\geq k+r}) \models \lambda_2$ and $(\mathfrak{M}, \pi_{\geq k+i}) \models \lambda_1$ for every $0 \leq i < r$. Using again Proposition 2, we deduce that $k + r < j$, we apply the induction hypothesis and obtain that $(\mathfrak{N}, \sigma_{\geq k+r}) \models \lambda_2$ and $(\mathfrak{N}, \sigma_{\geq k+i}) \models \lambda_1$ for each $0 \leq i < r$. We thus obtain that $(\mathfrak{N}, \sigma_{\geq k}) \models \lambda_1 \mathsf{U} \lambda_2$.

□

By the Lemma and Proposition above, and by the fact that any for any path π we have that $\pi = \pi_{\geq 1}$, we obtain the following.

Corollary 1. *Let θ be negation-free LTL formula, $\mathfrak{M}$ an iCGS, and π a path of $\mathfrak{M}$:*

1. *if $\pi_{\geq j}$ is $\bot$-like for some $j \geq 1$ and $(\mathfrak{M}, \pi) \models \theta$ then for every iCGS $\mathfrak{N}$ and every path σ of $\mathfrak{N}$ such that σ is $j-1$-twin of π we have that $(\mathfrak{N}, \sigma) \models \theta$.*

2. *if $\pi_{\geq j}$ is $\top$-like for some $j \geq 1$ and $(\mathfrak{M}, \pi) \not\models \theta$, then for every iCGS $\mathfrak{N}$ and every path σ of $\mathfrak{N}$ such that σ is $j-1$-twin of π we have that $(\mathfrak{N}, \sigma) \not\models \theta$.*

Lemma 2. *Let $\mathfrak{M}$ be an iCGS and $\mathfrak{M}^\bullet$ a negative sub-model of $\mathfrak{M}$ with perfect information. For any subset Γ of agents, for any negation-free formula φ of the form $\langle\langle\Gamma\rangle\rangle \psi$ (resp., $[\![\Gamma]\!] \psi$), for any $s \in S^\bullet$, we have that $(\mathfrak{M}^\bullet, s) \models \varphi$ implies $s \neq s_\bot$ and $(\mathfrak{M}, s) \models \varphi$.*

Proof. We consider a formula φ whose rank is 1, i.e., ψ is an LTL formula. This suffices since we can extend the result to formulas of arbitrary rank using the classic bottom up approach.

Suppose that $(\mathfrak{M}^\bullet, s) \models \varphi$. By the definition of satisfaction, this means that there is a strategy Σ_Γ for the coalition Γ such that $(\mathfrak{M}^\bullet, \rho) \models \psi$ for any $\rho \in out(s, \Sigma_\Gamma)$. Observe that given any of these paths ρ, we have two cases:

1. either for every $i \geq 1$, $\rho_i \in \mathsf{S}$, or

2. ρ has the form $\rho^{\mathsf{S}} \cdot \rho^\bot$, where ρ^{S} is the prefix of ρ such that $\rho^{\mathsf{S}}_i \in \mathsf{S}$ for each $1 \leq i \leq |\rho^{\mathsf{S}}|$, and $\rho^\bot$ is the suffix of ρ where $\rho^\bot = s^\omega_\bot$. That is, $\rho^\bot$ is a $\bot$-like. Remark that for any path ρ of the form $\rho^S \cdot \rho^\bot$ the prefix ρ^S must be non-empty because of Proposition 2. We thus obtain that $s \neq s_\bot$.

In the first case ρ is also a path of $\mathfrak{M}$, thus we get that $(\mathfrak{M}, \rho) \models \varphi$. In the second case, since ρ^S is non-empty and each state s of ρ is also a state of $\mathfrak{M}$, there must be at least a path σ of $\mathfrak{M}$ such that ρ^S is a strict-prefix or σ. We deduce that ρ and σ are k-twins for $k < j$ and $\rho_{\geq j} = \rho^\bot$. We thus apply Corollary 1 and obtain that $(\mathfrak{M}, \sigma) \models \psi$. Now, consider any uniform strategy Σ'_Γ for the coalition Γ such that $\Sigma'_\Gamma(h) = \Sigma_\Gamma(h)$ for any history h such that $h \sqsubset \rho \in out(s, \Sigma_\Gamma)$. Clearly any path $\sigma \in out(s, \Sigma'_\Gamma)$ is either equal to some path in $out(s, \Sigma_\Gamma)$ or is a k-twin, for some k, of a path of the form $\pi \cdot \pi^\bot$ in $out(s, \Sigma_\Gamma)$. Thus, any path in $out(s, \Sigma'_\Gamma)$ satisfies ψ. By the above discussion, we can conclude that $(\mathfrak{M}, s) \models \varphi$.

□

The following Lemma states that any (negation-free) formula that is not satisfied in a positive submodel of a given model, will be not satisfied in the model itself.

Lemma 3. *Let $\mathfrak{M}$ be an iCGS and $\mathfrak{M}^\circ$ a positive sub-model of $\mathfrak{M}$ with perfect information. For any subset Γ of agents, for any negation-free formula φ of the form $\langle\!\langle\Gamma\rangle\!\rangle\psi$ (resp., $[\![\Gamma]\!]\psi$), for any $s \in S^\circ$, we have that $(\mathfrak{M}^\circ, s) \not\models \varphi$ implies $s \neq s_\top$ and $(\mathfrak{M}, s) \not\models \varphi$.*

Proof. The proof of this lemma mirrors the proof of the lemma above. One obtains that any path in $\mathfrak{M}^\circ$ must either be composed only of states of $\mathfrak{M}$ or must be of the form $\rho^{\mathsf{S}} \cdot \rho^\top$ where $\rho^\top = s_\top^\omega$. Given this fact, we apply exactly the same line of reasoning as above. □

We thus have that the wanted preservation result holds for formulas that are negation-free. At first glance, this may see a limitation of our approach. However, first impressions are not always correct: we can extend the above preservation results to all formulas by using a simple trick.

First, given an ATL* formula θ, we substitute each negated atom $\neg q$ of θ by a "fresh" atom nq, obtaining a formula $(\theta)^t$. Then, given an iCGS $\mathfrak{M}$ in which states are labeled by elements of $\mathtt{Ap}$, we define a new iCGS $\mathfrak{M}_{\mathtt{Ap}^-}$ that have the same structure as $\mathfrak{M}$ and in which states are labeled by elements of $\mathtt{Ap} \cup \mathtt{Ap}^-$ where $\mathtt{Ap}^-$ is a set of fresh atomic propositions containing an atom nq for each atom $q \in \mathtt{Ap}$. In this latter iCGS we will impose that an atom $nq \in \mathtt{Ap}^-$ will label a state s if and only if $(\mathfrak{M}, s) \models \neg q$. As a consequence of this transformation, we will obtain that $(\mathfrak{M}, s) \models \theta$ if and only if $(\mathfrak{M}_{\mathtt{Ap}^-}, s) \models (\theta)^t$. More precisely, let $\mathtt{Ap}$ be a finite, non-empty set of atomic propositions. We denote by $\mathtt{Ap}^-$ the set $\{np \mid p \in \mathtt{Ap}\}$. The formal translation follows.

Definition 11. Let $\mathcal{F}$ be a set of formulas generated by a finite set of atoms $\mathtt{Ap}$. Let $\mathcal{F}^-$ be a set of negation-free formulas generated by $\mathtt{Ap} \cup \mathtt{Ap}^-$. The function $(-)^t : \mathcal{F} \to \mathcal{F}^-$ is defined by structural induction on $\theta \in \mathcal{F}$ as follows:

$$\begin{aligned}
(p)^t &= p && \text{if } p \text{ is an atom}\\
(\neg p)^t &= np &&\\
(\circ\theta)^t &= \circ(\theta)^t && \circ \in \{\mathsf{X}, \langle\!\langle\Gamma\rangle\!\rangle, [\![\Gamma]\!]\}\\
(\theta \star \lambda)^t &= (\theta)^t \star (\lambda)^t && \star \in \{\wedge, \vee, \mathsf{U}, \mathsf{R}\}
\end{aligned}$$

Note that, the function $(-)^t$ is injective, translates state formulas to state formulas, path formulas to path formulas, and preserves the size of the formulas, *i.e.*, $|\theta| = |(\theta)^t|$ for any formula θ.

Now, we have all the elements to provide the formal definition of $\mathfrak{M}_{Ap^-}$.

Definition 12. Let $\mathfrak{M} = \langle \mathtt{Ag}, \mathtt{Ap}, \mathtt{S}, \mathtt{s}_I, \{\mathtt{Act}_i\}_{i\in\mathtt{Ag}}, \{\sim_i\}_{i\in\mathtt{Ag}}, d, \delta, \mathtt{V}\rangle$ be a model and $\mathtt{Ap}^- = \{np \mid p \in \mathtt{Ap}\}$. We write $\mathfrak{M}_{Ap^-}$ to denote the iCGS $\langle \mathtt{Ag}, \mathtt{Ap} \cup \mathtt{Ap}^-, \mathtt{S}, \mathtt{s}_i, \{\mathtt{Act}_i\}_{i\in\mathtt{Ag}}, \{\sim_i\}_{i\in\mathtt{Ag}}, d, \delta, \mathtt{V}^\star\rangle$ where $\mathtt{V}^\star(s) = \mathtt{V}(s) \cup \{np_1, \ldots, np_m\}$ iff $np_i \in \mathtt{Ap}^-$ and $p_i \notin \mathtt{V}(s)$.

We can now prove the result that we have anticipated just above Definition 11.

Proposition 3. *Let $\mathfrak{M} = \langle \mathtt{Ag}, \mathtt{Ap}, \mathtt{S}, \mathtt{s}_I, \{\mathtt{Act}_i\}_{i\in\mathtt{Ag}}, \{\sim_i\}_{i\in\mathtt{Ag}}, d, \delta, \mathtt{V}\rangle$ be an iCGS, $\mathfrak{M}_{\mathtt{Ap}^-}$ the corresponding iCGS introduced in Definition 12, and let r be either a path $\rho \in S^\omega$ or state $s \in S$. For any formula θ, we have that:*

$$(\mathfrak{M}, r) \models \theta \quad \textit{iff} \quad (\mathfrak{M}_{\mathtt{Ap}^-}, r) \models (\theta)^t$$

Proof. We prove that $(\mathfrak{M}, r) \models \theta$ implies $(\mathfrak{M}_{\mathtt{Ap}^-}, r) \models (\theta)^t$, and that $(\mathfrak{M}_{\mathtt{Ap}^-}, r) \models (\theta)^t$ implies $(\mathfrak{M}, r) \models \theta$ by induction on the size $|\theta|$ of θ.

($\Rightarrow$) If $|\theta| = 0$, then there are two cases:

i) θ is an atom $q \in \mathtt{Ap}$. Since $q \in \mathtt{V}(s)$ implies $q \in \mathtt{V}^\star(s)$ for all $s \in \mathtt{S}$ and $q \in \mathtt{Ap}$, we can conclude that $(\mathfrak{M}_{\mathtt{Ap}^-}, r) \models (\theta)^t$.

ii) $\theta = \neg q$ for some atom q, then $(\neg q)^t = nq$. Suppose that $(\mathfrak{M}, s) \models \neg q$. By the semantics $q \notin \mathtt{V}(s)$. By the definition of $\mathfrak{M}_{\mathtt{Ap}^-}$, $nq \in \mathtt{V}^\star(s)$ and thus $(\mathfrak{M}_{\mathtt{Ap}^-}, s) \models nq$.

Suppose that the statement of the proposition holds for any formula θ' such that $|\theta'| \leq n$ and let θ a formula with $|\theta| = n + 1$. We detail two cases:

– if $\theta = \mathsf{X}\psi$ then $(\mathfrak{M}, \rho) \models \mathsf{X}\psi$ iff $(\mathfrak{M}, \rho_{\geq 2}) \models \psi$. By the definition of $\mathfrak{M}_{\mathtt{Ap}^-}$ and by induction hypothesis, $(\mathfrak{M}_{\mathtt{Ap}^-}, \rho_{\geq 2}) \models (\psi)^t$, thus $(\mathfrak{M}_{\mathtt{Ap}^-}, \rho) \models \mathsf{X}(\psi)^t = (\theta)^t$.

– if $\theta = \psi_1 \mathsf{U} \psi_2$ then $(\mathfrak{M}, \rho) \models \theta$ iff there is $k \geq 1$ $(\mathfrak{M}, \rho_{\geq k}) \models \psi_2$ and for all $1 \leq j < k$ $(\mathfrak{M}, \rho_{\geq j}) \models \psi_1$. By induction hypothesis, $(\mathfrak{M}_{\mathtt{Ap}^-}, \rho_{\geq k}) \models (\psi_2)^t$ and for all $1 \leq j < k$, $(\mathfrak{M}_{\mathtt{Ap}^-}, \rho_{\geq j}) \models (\psi_1)^t$. Thus, by definition of $(-)^t$, $(\mathfrak{M}_{\mathtt{Ap}^-}, \rho) \models (\psi_1)^t \mathsf{U} (\psi_2)^t = (\theta)^t$.

($\Leftarrow$) If $|\theta| = 0$ there are two cases.

i) if θ is an atom q, since for all $q \in \mathtt{Ap}$, $q \in \mathtt{V}^\star(r)$ implies $q \in \mathtt{V}(r)$ whenever r is a state of $\mathtt{S}$. Since $(q)^t = q$ the result follows.

ii) if $\theta = \neg q$ for some atom q, we have that $(\neg q)^t = nq$ where $nq \in \mathtt{Ap}^-$. We have that $(\mathfrak{M}_{\mathtt{Ap}^-}, r) \models nq$ iff r is a state $s \in \mathtt{S}$ and $nq \in \mathtt{V}^\star(s)$. By Definition 12, $nq \in \mathtt{V}^\star(s)$ iff $q \notin \mathtt{V}(s)$ it follows that $(\mathfrak{M}, r) \models \neg q$.

Suppose that the statement of the proposition holds for any formula θ' such that $|\theta'| \leq n$ and let θ a formula with $|\theta| = n + 1$. All cases follow by a straightforward application of the induction hypothesis and by the definition of $\mathfrak{M}_{\mathtt{Ap}^-}$. We thus detail only two cases:

– if $\theta = \varphi_1 \wedge \varphi_2$ then $(\theta)^t = (\varphi_1)^t \wedge (\varphi_2)^t$. By definition $(\mathfrak{M}_{\mathtt{Ap}^-}, r) \models (\varphi_1)^t \wedge (\varphi_2)^t$ iff $(\mathfrak{M}_{\mathtt{Ap}^-}, r) \models (\varphi_1)^t$ and $(\mathfrak{M}_{\mathtt{Ap}^-}, r) \models (\varphi_2)^t$. We have that $|\varphi_i| < |\varphi_1 \wedge \varphi_2|$ for $i \in \{1, 2\}$. By induction hypothesis $(\mathfrak{M}, r) \models \varphi_1$ and $(\mathfrak{M}, r) \models \varphi_2$, thus $(\mathfrak{M}, r) \models \varphi_1 \wedge \varphi_2$.

– if $\theta = \langle\!\langle \Gamma \rangle\!\rangle \psi$ for some set of agents Γ and state-formula ψ then $(\theta)^t = \langle\!\langle \Gamma \rangle\!\rangle (\psi)^t$. $(\mathfrak{M}_{\mathtt{Ap}^-}, r) \models \langle\!\langle \Gamma \rangle\!\rangle (\psi)^t$ iff r is some state s and for some joint strategy Σ_Γ, for all $\rho \in out(s, \Sigma_\Gamma)$, $(\mathfrak{M}_{\mathtt{Ap}^-}, \rho) \models (\psi)^t$. Since $|\psi| < |\langle\!\langle \Gamma \rangle\!\rangle \psi|$, from $(\mathfrak{M}_{\mathtt{Ap}^-}, \rho) \models (\psi)^t$ we conclude by induction hypothesis that $(\mathfrak{M}, \rho) \models \psi$. By Definition 12, the function d and δ as well as the equivalence relation $\sim_i$ are the same on both $\mathfrak{M}$ and $\mathfrak{M}_{\mathtt{Ap}^-}$. We can conclude that the strategy Σ_Γ satisfies $\langle\!\langle \Gamma \rangle\!\rangle \psi$ at $s = r$ in $\mathfrak{M}$.

□

Thanks to the above proposition, we can extend the preservation results of Lemma 2 and 3 to *any* strategic formula $\langle\!\langle \Gamma \rangle\!\rangle \psi$. Given a model $\mathfrak{M}$, we simply evaluate $(\langle\!\langle \Gamma \rangle\!\rangle \psi)^t$ on negative (resp., positive) sub-models of $\mathfrak{M}_{\mathtt{Ap}^-}$.

5. The Procedure

In this section, we detail the procedure that we use to give an approximate solution to the model checking problem of ATL* with imperfect information and perfect recall. Our procedure is composed of different steps. In the following, we give a detailed description of each of these steps and provide complexity result for them. Finally, we put them together and prove the soundness of the so obtained algorithm.

Before diving into more details, we provide an high-level idea of our approach by means of Figure 4. As it can be noted, our procedure is divided into multiple phases. In the first step (ImperfectRecall), we check whether there are sub-formulas that can be checked with memoryless strategies and imperfect information over the model; a problem that is known to be decidable (Schobbens, 2003). Then, we clean the model and the formula w.r.t. the negated atoms (Preprocessing). After that, we generate all the positive and negative sub-models with perfect information (FindSub-Models). If only one sub-model is so generated, this means we have perfect information for the agents involved in the property; so, we can exploit model checking for perfect information and memoryful strategies (Model Checking IR). Otherwise, for each sub-model, we first check the satisfaction/violation of sub-formulas (CheckSub-Formulas). Then, to provide a verification result for the whole formula, we use CTL* model checking. If it is not possible to conclude a verdict, then the procedure continues its evolution by storing some information via Runtime Verification. The loop continues until there are no more sub-models to analyse, or a conclusive verdict is provided via CTL* model checking.

5.1 Subroutine Subformulas

The algorithms that we will present in the upcoming sections use different subroutines. The behavior of most of them can be straightforwardly grasped. Thus, we will introduce them directly into the algorithms so as not to further burden the reading of our work. One of these subroutines, however, requires a more detailed presentation. This subroutine, that we will present in the following paragraph, will be extensively used in Algotithm 5 to reduce the problem of checking whether a formula with multiple occurrence of strategic operators is true in a given model, to the problem of checking whether a formula with a single occurrence of a strategic operator is true in the given model. Said otherwise, the subroutine is needed to implement the classic bottom-up approach of strategic and temporal logic (Alur et al., 2002). We recall that the bottom-up approach can be formulated as follows. Given a formula ϕ containing multiple strategic operators and a model $\mathfrak{M}$, we first collect the set of states of the model that satisfies subformulas $\phi_1, ..., \phi_n$ of ϕ containing a single strategic operator. For any of these formulas we choose a fresh atomic formula $p_1, ..., p_n$. We add p_i to $\mathtt{V}(s)$ whenever s satisfies ϕ_i in $\mathfrak{M}$, obtaining a new model $\mathfrak{M}'$. We now consider the formula ϕ' obtained from ϕ by substituting each occurrence of one of the ϕ_i with atom p_i and evaluate ϕ' on $\mathfrak{M}'$ reusing the same procedure.

Let *Replace* be a function that takes as input a formula θ, a list of formulas $\langle \lambda_1, \ldots, \lambda_n \rangle$, and a list of atomic proposition $\langle p_1, \ldots, p_n \rangle$, and outputs the formula $\theta^\star$ obtained by replacing each occurrence of the formula λ_i in θ (if any) with p_i, for all $i \in \{1, \ldots, n\}$. If θ is a formula, we denote by $SubS(\theta)$ the set of strategic sub-formulas of θ and by $SubS^n(\theta)$ the set of strategic sub-formulas of θ whose rank is n. Algorithm 1 takes as input a formula θ and works as follows. First, we initialize a list L (line 1), then a while loop starts and continues until the rank $r(\theta)$ of θ is bigger or equal to 1 (lines 2-10). Inside the latter, the algorithm loops on the subset of strategic sub-formulas of θ

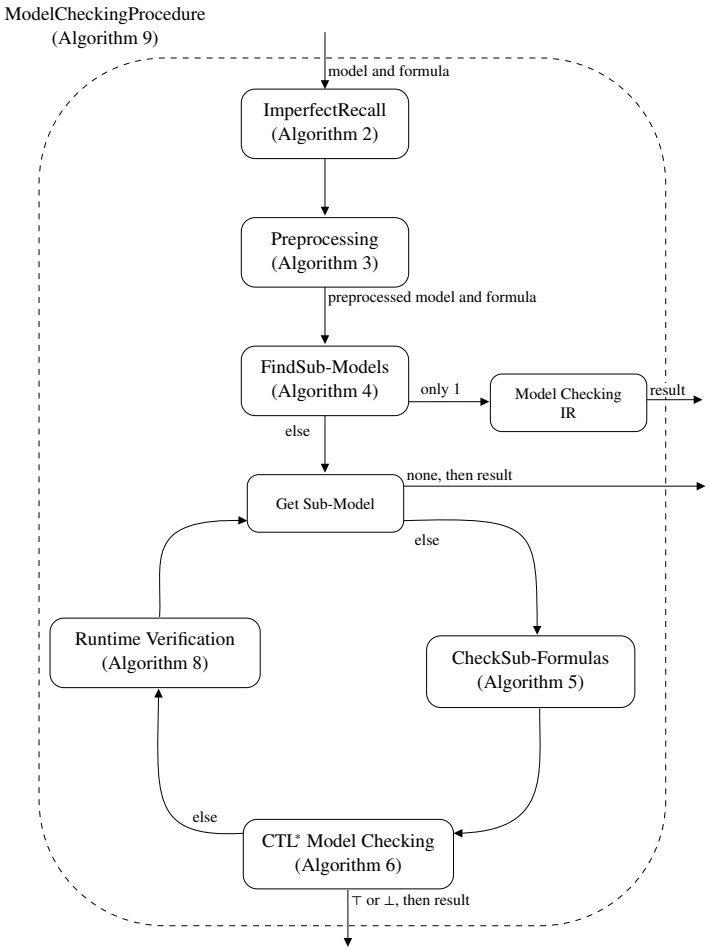

Figure 4: Procedure's overview.

whose rank is 1 (lines 5-8). For each of these sub-formulas φ, it adds φ to *Temp* (line 6), creates a new atomic proposition p_φ and adds it to *Satom* (lines 7-8). It then exits the for loop and replaces in θ each strategic formula of *Temp* with the corresponding atom in *Satom* (line 9) and concatenates *Temp* to L (line 10). By doing so, the rank of θ goes down at every cycle of the while loop. Finally, the algorithm outputs the lists L and *Satom*. Note that, this procedure works in polynomial time with respect to the number of sub-formulas of θ.

Example 1. Let us consider the formula:

$$\varphi = (\langle\langle \Gamma_1 \rangle\rangle \mathsf{X} [\![\Gamma_2]\!]\, q \,\mathsf{U}\, r) \wedge (\langle\langle \Gamma_3 \rangle\rangle \mathsf{X}\, q \vee [\![\Gamma_4]\!]\, q \,\mathsf{R}\, r)$$

this formula has rank 2, and its strategic sub-formulas of rank 1 are $\varphi_1 = [\![\Gamma_2]\!]\, q \,\mathsf{U}\, r$, $\varphi_2 = \langle\langle \Gamma_3 \rangle\rangle \mathsf{X}\, q$ and $\varphi_3 = [\![\Gamma_4]\!]\, q \,\mathsf{R}\, r$. After the first iteration of the while loop, we obtain that *Temp* is $\langle \varphi_1, \varphi_2, \varphi_3 \rangle$ and *Satom* is $\langle atom_{\varphi_1}, atom_{\varphi_2}, atom_{\varphi_3} \rangle$, then L is equal to *Temp* and φ becomes $(\langle\langle \Gamma_1 \rangle\rangle \mathsf{X}\, atom_{\varphi_1}) \wedge (atom_{\varphi_2} \vee atom_{\varphi_3})$. This latter formula has rank 1, and its only strategic sub-formula is $\varphi_4 = \langle\langle \Gamma_1 \rangle\rangle \mathsf{X}\, atom_{\varphi_1}$. After another iteration of the while loop, we obtain that φ has become $atom_{\varphi_4} \wedge$

Algorithm 1 *Subformulas*(θ)

```
1: L = ∅;
2: while r(θ) ≥ 1 do
3:     Temp = ∅;
4:     Satom = ∅;
5:     for φ ∈ SubS^1(θ) do
6:         Temp = Temp · ⟨φ⟩;
7:         generate atom atom_φ;
8:         Satom = Satom · ⟨atom_φ⟩;
9:     θ = Replace(θ, Temp, Satom);
10:    L = L · Temp;
11: return ⟨L, Satom⟩;
```

($atom_{\varphi_2} \vee atom_{\varphi_3}$) and that $L = \langle \varphi_1, \varphi_2, \varphi_3, \varphi_4 \rangle$ and $Satom = \langle atom_{\varphi_1}, atom_{\varphi_2}, atom_{\varphi_3}, atom_{\varphi_4} \rangle$. Since now φ has rank 0, we can exit the while loop and return the lists L and *Satom*.

5.2 Phase 1: Imperfect Recall Check

Now that we have detailed the behavior of the subroutine *Subformulas*, we can start exposing the main components of our procedure.

First, given an ATL* formula θ and an iCGS $\mathfrak{M}$, we check which strategic sub-formulas of θ can be satisfied by $\mathfrak{M}$ by using imperfect recall strategies. With more detail, we create a new iCGS $\mathfrak{M}'$ whose states are labeled by atomic propositions representing the strategic sub-formulas of θ that can be satisfied using imperfect recall strategies and we substitute in φ those strategic sub-formulas by the aforementioned atomic propositions (generated by the subroutine *Subformulas*).

Algorithm 2 is simple and works as follows: for each strategic sub-formula φ of θ and each state s of $\mathfrak{M}$, we check whether $(\mathfrak{M}, s) \models_{ir} \varphi$. If this is the case, we add $atom_\varphi$ to the set of atoms labeling s (line 6-7). Finally, we return the so obtained model and formula (line 8). Note that, the complexity of this procedure is given by the model checking for ATL* with imperfect information and imperfect recall strategies, that is PSPACE (Schobbens, 2003).

Algorithm 2 *ImperfectRecall* ($\mathfrak{M}$, θ)

```
1: ⟨L, Satom⟩ = Subformulas(θ);
2: for φ ∈ L do
3:     extract atom_φ from Satom;
4:     for s ∈ S do
5:         if (M, s) ⊨_ir φ then
6:             Ap = Ap ∪ {atom_φ};
7:             V(s) = V(s) ∪ {atom_φ};
8: return ⟨M, θ⟩;
```

5.3 Phase 2: Preprocessing

Here, we present the second phase of our procedure. Informally, given an iCGS $\mathfrak{M}$ and an ATL* formula θ, Algorithm 3 replaces the negated atoms by additional atoms, and updates the model $\mathfrak{M}$ accordingly. With more detail, first all the negated atoms are extracted (line 1). Then, for each negated atom $\neg p$, the algorithm generates a new atomic proposition np (line 3), adds np to the set

of atomic propositions $\mathtt{Ap}$ (line 4), updates the formula (line 5), and updates the labeling function of $\mathfrak{M}$. For the latter, for each state s of $\mathfrak{M}$, the algorithm checks if p is false in s; if this is the case, it adds np to the set of atomic propositions true on s (lines 6-8). Note that, this procedure works in polynomial time with respect to the number of negated atoms.

Algorithm 3 *Preprocessing* $(\mathfrak{M}, \theta)$

```
1: neg-atoms = NegAtoms(φ);
2: for ¬p ∈ neg-atoms do
3:     generate atom np;
4:     Ap = Ap ∪ {np};
5:     Replace(θ, ¬p, np);
6:     for s ∈ S do
7:         if p ∉ V(s) then
8:             V(s) = V(s) ∪ np;
```

5.4 Phase 3: Find Sub-models

Here, we present how to find the sub-models with perfect information inside the original model. The procedure to generate the set of positive and negative sub-models of $\mathfrak{M}$ with perfect information for the agents involved in the coalitions of θ is presented in Algorithm 4. The algorithm takes an iCGS $\mathfrak{M}$ and an ATL* formula θ and works with two sets. The set *substates* contains the sets of states of $\mathfrak{M}$ that have to be evaluated, and the set *candidates* contains pairs of sub-models of $\mathfrak{M}$ with perfect information. Each pair represents the positive and negative sub-models for a given subset of states of $\mathfrak{M}$. In the first part of the algorithm (lines 1-4) the set of states and the indistinguishability relation are extracted from the model $\mathfrak{M}$ and the lists *substates* and *candidates* are initialised to $\mathtt{S}$ and $\emptyset$, respectively. The main loop (lines 5-15) works until the list *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 $i \in \mathtt{Ag}(\theta)$[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 $Ag(\theta)$. So, from S' two models are generated, a positive sub-model as described in Definition 10 and a negative sub-model as described in Definition 9, and the pair is added to the list *candidates*. The latter is returned at the end of Algorithm 4 (line 16).

Example 2. In Figure 5, we show a candidate negative sub-model $\mathfrak{M}^\bullet$ extracted from the rover mission (see the model in Figure 2) using Algorithm 4. Note that, we can generate the positive sub-model counterpart by replacing the state $s_\perp$ with $s_\top$ and following the labelling rule described in Definition 10.

To conclude this part, we provide the complexity of the described procedure.

Theorem 1. *Algorithm 4 terminates in exponential time w.r.t. the size of the model* $\mathfrak{M}$.

Proof. The procedures *GenerateNegative*() and *GeneratePositive*() to produce models $\mathfrak{M}^\bullet$ and $\mathfrak{M}^\circ$ are polynomial in the size of the original model $\mathfrak{M}$. In the worst case, the loop to generate the list *candidates* (lines 5-15) terminates in an exponential number of steps. Note that, the worst case happens when there is full indistinguishability, *i.e.*, $\forall s, s' \in \mathtt{S}$: $s \sim_i s'$. For the above reasoning, the result follows. □

1. $\mathtt{Ag}(\theta)$ is the set of agents involved in the coalitions of the formula θ.

Algorithm 4 *FindSub-models* $(\mathfrak{M}, \theta)$

```
1: extract ∼ from 𝔐;
2: extract S from 𝔐;
3: candidates = ∅;
4: substates = {S};
5: while substates is not empty do
6:     extract S′ from substates;
7:     if there exists s ∼_i s′ with s ≠ s′ and i ∈ Ag(θ) then
8:         S_1 = S′ \ {s};
9:         substates = S_1 ∪ substates;
10:        S_2 = S′ \ {s′};
11:        substates = S_2 ∪ substates;
12:    else
13:        𝔐• = GenerateNegative(𝔐, S′);
14:        𝔐° = GeneratePositive(𝔐, S′);
15:        candidates = ⟨𝔐•, 𝔐°⟩ ∪ candidates;
16: return candidates;
```

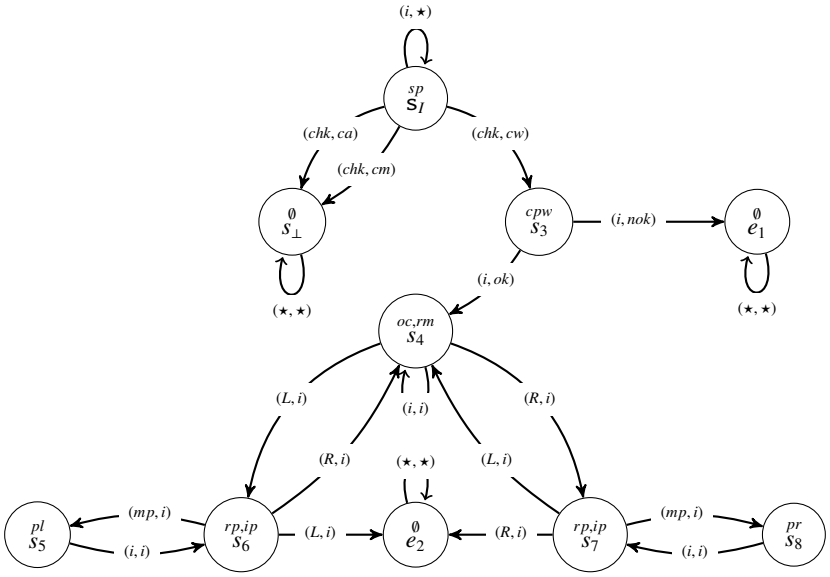

Figure 5: A negative sub-model generated by Algorithm 4 from model $\mathfrak{M}$ depicted in Figure 2.

5.5 Phase 4: Check Sub-formulas

Given a candidate, we present here how to check the sub-formulas on it. The procedure is presented in Algorithm 5. Informally, the algorithm takes a candidate and an ATL* formula θ. It works with the set *result* that contains tuples $\langle s, \varphi, vatom_\varphi \rangle$, where s is a state in a sub-model, φ is a (strategic) sub-formula of θ, and $vatom_\varphi$ is the atomic proposition generated from φ, where $v \in \{n, p\}$ is the type of the sub-model (*i.e.*, positive or negative). In more detail, s satisfies φ and v is used to remember if φ 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*(). The extraction of the set of sub-formulas from θ is necessary to verify whether there are sub-models satisfying/unsatisfying at least a part of θ (not exclusively the whole θ). If that is the case, such sub-

models can later on be used to verify θ over the entire model $\mathfrak{M}$ through CTL* model checking. The main loop (lines 3-12) works until all the sub-formulas of θ 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 $\mathfrak{M}^\bullet$ and $\mathfrak{M}^\circ$. 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 $\mathfrak{M}^\bullet$ (line 6), then the procedure updates the model $\mathfrak{M}^\bullet$ through the set of atomic propositions and the labelling function (line 7). Furthermore, in line 8, the set *result* is updated by adding a new tuple. The reasoning applied to the model $\mathfrak{M}^\bullet$ is also applied to $\mathfrak{M}^\circ$ (lines 9-11). At the end of the procedure, the state $s_\top$ is updated (line 12) since, by Definition 10, it needs to contain all the possible atoms that are in $\mathfrak{M}$.

Algorithm 5 *CheckSub-formulas* ($\langle \mathfrak{M}^\bullet, \mathfrak{M}^\circ \rangle$, θ)

1: $result = \emptyset$;
2: $\langle subformulas, atoms \rangle = Subformulas(\theta)$;
3: **while** $subformulas \neq \emptyset$ **do**
4: extract φ from *subformulas*; extract $atom_\varphi$ from *atoms*;
5: **for** $s \in \mathtt{S}^\bullet \cap \mathtt{S}^\circ$ **do**
6: **if** $\mathfrak{M}^\bullet, s \models_{IR} \varphi$ **then**
7: $\mathtt{Ap}^\bullet = \mathtt{Ap}^\bullet \cup \{atom_\varphi\}$; $\mathtt{V}^\bullet(s) = \mathtt{V}^\bullet(s) \cup \{atom_\varphi\}$;
8: $result = \langle s, \varphi, natom_\varphi \rangle \cup result$;
9: **if** $\mathfrak{M}^\circ, s \models_{IR} \varphi$ **then**
10: $\mathtt{Ap}^\circ = \mathtt{Ap}^\circ \cup \{atom_\varphi\}$; $\mathtt{V}^\circ(s) = \mathtt{V}^\circ(s) \cup \{atom_\varphi\}$;
11: $result = \langle s, \varphi, patom_\varphi \rangle \cup result$;
12: $\mathtt{Ap}^\circ = \mathtt{Ap}^\circ \cup \{atom_\varphi\}$; $\mathtt{V}^\circ(s_\top) = \mathtt{V}^\circ(s_\top) \cup \{atom_\varphi\}$;
13: **return** *result*;

Example 3. Given the negative sub-model $\mathfrak{M}^\bullet$ as in Figure 5 and its positive counterpart $\mathfrak{M}^\circ$ as input of Algorithm 5 we can analyze the formulas[3]:

$$\varphi_1 = \langle\!\langle rover \rangle\!\rangle \mathsf{F}((oc \wedge rm) \wedge \mathsf{F}((pl \vee pr) \wedge \mathsf{F}(oc \wedge rm)))$$

$$\varphi_2 = \langle\!\langle rover, mechanic \rangle\!\rangle \mathsf{F}((oc \wedge rm) \wedge \langle\!\langle rover \rangle\!\rangle \mathsf{F}((pl \vee pr) \wedge \mathsf{F}(oc \wedge rm)))$$

$$\varphi_3' = \langle\!\langle rover, mechanic \rangle\!\rangle \mathsf{F}((rp \wedge nip) \wedge \langle\!\langle rover \rangle\!\rangle \mathsf{F}((pl \vee pr) \wedge \mathsf{F}(oc \wedge rm)))$$

We start with φ_1 where the list *subformulas* is initialized to $\langle \varphi_1 \rangle$ 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 lines 7 and 10 it updates the model $\mathfrak{M}^\diamond$ by adding $atom_{\varphi_1}$ in $\mathtt{Ap}^\diamond$ and by updating the labeling function as $\mathtt{V}^\diamond(s_i) = \mathtt{V}^\diamond(s_i) \cup \{atom_{\varphi_1}\}$, for all $i \in \{4, 5, 6, 7, 8\}$ and with $\diamond \in \{\bullet, \circ\}$. Since there is only one strategic operator, the procedure is concluded. For φ_2, the list *subformulas* is initialized to $\langle \varphi_{21} = \langle\!\langle rover \rangle\!\rangle \mathsf{F}((pl \vee pr) \wedge \mathsf{F}(oc \wedge rm)), \varphi_{22} = \langle\!\langle rover, mechanic \rangle\!\rangle \mathsf{F}((oc \wedge rm) \wedge atom_{\varphi_{21}}) \rangle$. By line 4, the algorithm extracts φ_{21} 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 lines 7 and 10 it updates the model $\mathfrak{M}^\diamond$ by adding $atom_{\varphi_{21}}$ in $\mathtt{Ap}^\diamond$ and by updating the labeling function as $\mathtt{V}^\diamond(s_i) = \mathtt{V}^\diamond(s_i) \cup \{atom_{\varphi_{21}}\}$, for all $i \in \{4, 5, 6, 7, 8\}$ and with $\diamond \in \{\bullet, \circ\}$. For the second iteration of the main loop (lines 3-11) the

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 Algorithm 3.

formula φ_{22} is analyzed. In this case, φ_{22} is true in the negative and positive sub-models in states s_I, s_3, s_4, s_5, s_6, s_7, and s_8. Again, the models $\mathfrak{M}^{\bullet}$ and $\mathfrak{M}^{\circ}$ are updated by adding $atom_{\varphi_{22}}$ in $\mathtt{Ap}^{\diamond}$ and by updating the labeling function as $\mathtt{V}^{\diamond}(s_i) = \mathtt{V}^{\diamond}(s_i) \cup \{atom_{\varphi_{22}}\}$, for all $i \in \{I, 3, 4, 5, 6, 7, 8\}$ with $\diamond \in \{\bullet, \circ\}$. For φ'_3, the list *subformulas* is initialized to $\langle \varphi_{21} = \langle\!\langle rover \rangle\!\rangle \mathsf{F}((pl \vee pr) \wedge \mathsf{F}(oc \wedge rm)), \varphi'_{31} = \langle\!\langle rover, mechanic \rangle\!\rangle \mathsf{F}(rp \wedge nip \wedge atom_{\varphi_{21}}) \rangle$. Since φ_{21} is the same as before for φ_2, we have the same behavior of the algorithm. For the second iteration, φ'_{31} is false in all the states in $\mathtt{S}^{\circ} \cap \mathtt{S}^{\bullet}$. So, no update is done in the set *result*.

To conclude this part, we provide the complexity result.

Theorem 2. *Algorithm 5 terminates in double exponential time w.r.t. the size of θ.*

Proof. The procedure *Subformulas*() is polynomial in the size of the formula under exam. ATL* model checking, called in lines 6 and 9, is $2EXPTIME$ (Alur et al., 2002). Furthermore, the number of iterations of the inner part (lines 5-11) is $|\theta| \cdot |\mathtt{S}|$, *i.e.*, polynomial in the size of the model and of the formula. Then, the total complexity derives from ATL* model checking and is $2EXPTIME$. □

5.6 Phase 5: CTL* Verification

In the previous sections, we showed how to extract sub-models of a model $\mathfrak{M}$ satisfying at least one sub-formula φ of θ. Here, we present in Algorithm 6, how to apply CTL* model checking to try to conclude the satisfaction/violation of θ in $\mathfrak{M}$.

Algorithm 6 takes as input an iCGS $\mathfrak{M}$, an ATL* formula θ, and a set *result*. The latter contains tuples $\langle s, \varphi, vatom_{\varphi} \rangle$, where s is a state of $S^{\bullet} \cap S^{\circ}$, φ is a (strategic) sub-formula of θ, $vatom_{\varphi}$ is the atomic proposition generated from φ. Algorithm 7 denotes an auxiliary procedure, which is used in Algorithm 6 to update the model and the formula w.r.t. the results obtained in Algorithm 5. Thus, we start describing Algorithm 7. The algorithm starts by updating the model $\mathfrak{M}$ with the atoms generated from Algorithm 5 and stored in *result* (lines 1-3). In lines 4-10, the procedure generates formulas $\varphi_{\circ}$ and $\varphi_{\bullet}$ in accordance with the tuples in *result*. The formulas $\theta_{\bullet}$ and $\theta_{\circ}$ represent the ATL* formulas which remain to verify in $\mathfrak{M}$. These, along with the updated model $\mathfrak{M}'$, are the outcomes of Algorithm 7, and are the ones returned to Algorithm 6 (line 2). Continuing with Algorithm 6, the algorithm proceeds with the CTL* verification. To achieve this, we transform $\theta_{\bullet}$ and $\theta_{\circ}$ into their CTL* counterparts θ_{A} and θ_{E}, respectively (lines 3-4). Given a formula θ in ATL*, we denote with θ_{A} (resp., θ_{E}) the universal formula (resp., existential) of θ, *i.e.*, we substitute each occurrence of a strategic operator in θ with the universal (resp., existential) operator of CTL*. Given θ_{A} and θ_{E} we can perform standard CTL* model checking. First, by verifying if θ_{A} is satisfied by $\mathfrak{M}'$ (line 5). If this is the case, we can derive that $\mathfrak{M} \models \theta$ and set $k = \top$. Otherwise, the algorithm continues verifying if θ_{E} is not satisfied by $\mathfrak{M}'$ (line 7). If this is the case we can derive that $\mathfrak{M} \not\models \theta$ and set $k = \bot$. If neither of the two previous conditions are satisfied, then the inconclusive result is returned.

Before concluding, we show a preservation result from CTL* to ATL*. This is the key to guarantee that the truth value returned by Algorithm 6 also holds for the ATL* formula.

Lemma 4. *Given a model $\mathfrak{M}$, a formula φ in ATL*, and the CTL* universal (resp., existential) version φ_{A} (resp., φ_{E}) of φ. For any $s \in \mathsf{S}$, we have that:*

1. $(\mathfrak{M}, s) \models \varphi_{\mathsf{A}} \Rightarrow (\mathfrak{M}, s) \models \varphi$;

Algorithm 6 *Verification* ($\mathfrak{M}$, θ, *result*)

```
k = ?;
⟨𝔐′, ⟨θ•, θ∘⟩⟩ = UpdateModel&Formula(𝔐, θ, result);
θ_A = FromATLtoCTL(θ•, •);
θ_E = FromATLtoCTL(θ∘, ∘);
if 𝔐′ ⊨ θ_A then
    k = ⊤;
if 𝔐′ ⊭ θ_E then
    k = ⊥;
return k;
```

Algorithm 7 *UpdateModel&Formula* ($\mathfrak{M}$, θ, *result*)

```
for s ∈ S do
    take set atoms from result(s);
    Ap = Ap ∪ atoms; V(s) = V(s) ∪ atoms;
θ• = θ, θ∘ = θ;
while result is not empty do
    extract ⟨s, φ, atom_φ⟩ from result;
    if v = n then
        θ• = Replace(θ•, φ, natom_φ);
    else
        θ∘ = Replace(θ∘, φ, patom_ψ);
return ⟨𝔐, ⟨θ•, θ∘⟩⟩;
```

2. $(\mathfrak{M}, s) \not\models \varphi_{\mathsf{E}} \Rightarrow (\mathfrak{M}, s) \not\models \varphi$.

Proof. To prove (1), consider the formula $\varphi = \langle\!\langle \Gamma \rangle\!\rangle \psi$, in which $\Gamma \subseteq \mathtt{Ag}$ and ψ is a temporal formula without quantification. By consequence, $\varphi_{\mathsf{A}} = \mathsf{A}\psi$. By the semantics of CTL*, $(\mathfrak{M}, s) \models A\psi$ iff for all paths ρ with $\rho_1 = s$, $(\mathfrak{M}, \rho) \models \psi$. As presented in (Alur et al., 2002), the formula $\mathsf{A}\psi$ in CTL* is equivalent to $\langle\!\langle \emptyset \rangle\!\rangle \psi$ in ATL*. In fact, by Definition 6, the set of paths that need to verify ψ are in $out(s, \emptyset)$, *i.e.*, all the paths of $\mathfrak{M}$ from s. Then, we need to prove that: if $(\mathfrak{M}, s) \models \langle\!\langle \emptyset \rangle\!\rangle \psi$ then $(\mathfrak{M}, s) \models \langle\!\langle \Gamma \rangle\!\rangle \psi$. But, this follows directly by the semantics in Definition 6.

Now, consider the formula $\varphi = [\![\Gamma]\!] \psi$, in which $\Gamma \subseteq \mathtt{Ag}$ and ψ is a temporal formula without quantification. By consequence, $\varphi_{\mathsf{A}} = \mathsf{A}\psi$. By the semantics of CTL*, $(\mathfrak{M}, s) \models \mathsf{A}\psi$ iff for all paths ρ with $\rho_1 = s$, $(\mathfrak{M}, \rho) \models \psi$. As presented in (Alur et al., 2002), the formula $\mathsf{A}\psi$ in CTL* is equivalent to $[\![\mathtt{Ag}]\!] \psi$ in ATL*. In fact, by Definition 6, for each joint strategy $\Sigma_{\mathtt{Ag}}$ the resulted path needs to verify ψ. Since we consider all the possible joint strategies for the whole set of agents by consequence all the paths of $\mathfrak{M}$ from s need to satisfy ψ. Then, we need to prove that: if $(\mathfrak{M}, s) \models [\![\mathtt{Ag}]\!] \psi$ then $(\mathfrak{M}, s) \models [\![\Gamma]\!] \psi$. But, this follows directly by the semantics in Definition 6.

To prove (2), consider the formula $\varphi = \langle\!\langle \Gamma \rangle\!\rangle \psi$, in which $\Gamma \subseteq \mathtt{Ag}$ and ψ is a temporal formula without quantifications. By consequence, $\varphi_{\mathsf{E}} = \mathsf{E}\psi$. By the semantics of CTL*, $(\mathfrak{M}, s) \not\models \mathsf{E}\psi$ iff there is no path ρ with $\rho_1 = s$, such that $(\mathfrak{M}, \rho) \models \psi$. As presented in (Alur et al., 2002), the formula $\mathsf{E}\psi$ in CTL* is equivalent to $\langle\!\langle \mathtt{Ag} \rangle\!\rangle \psi$ in ATL*. In fact, by Definition 6, we take $out(s, \Sigma_{\mathtt{Ag}})$, *i.e.*, only one path of $\mathfrak{M}$. Then, we need to prove that: if $(\mathfrak{M}, s) \not\models \langle\!\langle \mathtt{Ag} \rangle\!\rangle \psi$ then $(\mathfrak{M}, s) \not\models \langle\!\langle \Gamma \rangle\!\rangle \psi$. But, this follows directly by the semantics in Definition 6.

Now, consider the formula $\varphi = [\![\Gamma]\!] \psi$, in which $\Gamma \subseteq \mathtt{Ag}$ and ψ is a temporal formula without quantifications. By consequence, $\varphi_{\mathsf{E}} = \mathsf{E}\psi$. By the semantics of CTL*, $(\mathfrak{M}, s) \not\models \mathsf{E}\psi$ iff there is not a path ρ with $\rho_1 = s$, such that $(\mathfrak{M}, \rho) \models \psi$. As presented in (Alur et al., 2002), the formula $\mathsf{E}\psi$ in

CTL* is equivalent to $[\![\emptyset]\!]\psi$ in ATL*. In fact, by Definition 6, we take $out(s,\emptyset)$, *i.e.*, all the paths of $\mathfrak{M}$ starting from s, and verify if there exists a path in $out(s,\emptyset)$ satisfying ψ. Then, we need to prove that: if $(\mathfrak{M},s) \not\models [\![\emptyset]\!]\psi$ then $(\mathfrak{M},s) \not\models [\![\Gamma]\!]\psi$. But, this follows directly by the semantics in Definition 6.

To conclude the proof, note that if we have a formula with more strategic operators, then we can use the classic bottom-up approach.

□

Example 4. 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: $\langle s_4, \varphi_1, vatom_{\varphi_1}\rangle, \langle s_5, \varphi_1, vatom_{\varphi_1}\rangle, \langle s_6, \varphi_1, vatom_{\varphi_1}\rangle$, $\langle s_7, \varphi_1, vatom_{\varphi_1}\rangle$, *and* $\langle s_8, \varphi_1, vatom_{\varphi_1}\rangle$ where $v \in \{p, n\}$. First, Algorithm 6 is launched and the model $\mathfrak{M}$ depicted in Figure 2 is updated with the atoms $natom_{\varphi_1}$ and $patom_{\varphi_1}$ in states s_4, s_5, s_6, s_7, *and* s_8. Then, $\theta_\bullet$ and $\theta_\circ$ are generated as $natom_{\varphi_1}$ and $patom_{\varphi_1}$, respectively. Since there are no strategic operators in the latter formulas, then $\theta_{\mathsf{A}} = \theta_\bullet$ *and* $\theta_{\mathsf{E}} = \theta_\circ$. In Algorithm 6, the condition in line 5 is not satisfied, but it is in line 7. So, the output of the procedure is $\bot$, *i.e.*, the formula is false in the model $\mathfrak{M}$. For φ_2 the set *result* is composed by the tuples:
$\langle s_4, \varphi_{21}, vatom_{\varphi_{21}}\rangle, \langle s_5, \varphi_{21}, vatom_{\varphi_{21}}\rangle, \langle s_6, \varphi_{21}, vatom_{\varphi_{21}}\rangle, \langle s_7, \varphi_{21}, vatom_{\varphi_{21}}\rangle, \langle s_8, \varphi_{21}, vatom_{\varphi_{21}}\rangle$,
$\langle s_0, \varphi_{22}, vatom_{\varphi_{22}}\rangle, \langle s_3, \varphi_{22}, vatom_{\varphi_{22}}\rangle, \langle s_4, \varphi_{22}, vatom_{\varphi_{22}}\rangle, \langle s_5, \varphi_{22}, vatom_{\varphi_{22}}\rangle, \langle s_6, \varphi_{22}, vatom_{\varphi_{22}}\rangle$,
$\langle s_7, \varphi_{22}, vatom_{\varphi_{22}}\rangle$, $\langle s_8, \varphi_{22}, vatom_{\varphi_{22}}\rangle$ where $v \in \{n, p\}$. Using again Algorithm 6, $\mathfrak{M}$ is updated with the atoms $natom_{\varphi_{21}}$ and $patom_{\varphi_{21}}$ in states s_4, s_5, s_6, s_7, and s_8; and atoms $natom_{\varphi_{22}}$ and $patom_{\varphi_{22}}$ in states s_I, s_3, s_4, s_5, s_6, s_7, and s_8. Then, $\theta_\bullet$ and $\theta_\circ$ are generated as $natom_{\varphi_{22}}$ and $patom_{\varphi_{22}}$, respectively. As before, $\theta_{\mathsf{A}} = \theta_\bullet$ and $\theta_{\mathsf{E}} = \theta_\circ$. The condition in line 5 of Algorithm 6 is satisfied, so the output of the procedure is $\top$, *i.e.*, the formula is true in the model $\mathfrak{M}$. For φ_3' the set result is composed by the tuples: $\langle s_4, \varphi_{21}, vatom_{\varphi_{21}}\rangle, \langle s_5, \varphi_{21}, vatom_{\varphi_{21}}\rangle, \langle s_6, \varphi_{21}, vatom_{\varphi_{21}}\rangle$, $\langle s_7, \varphi_{21}, vatom_{\varphi_{21}}\rangle$, and $\langle s_8, \varphi_{21}, vatom_{\varphi_{21}}\rangle$ where $v \in \{n, p\}$. Using Algorithm 6, $\mathfrak{M}$ is updated with the atoms $natom_{\varphi_{21}}$ and $patom_{\varphi_{21}}$ in states s_4, s_5, s_6, s_7, and s_8. Then $\theta_\bullet$ and $\theta_\circ$ are generated as $\langle\!\langle rover, mechanic\rangle\!\rangle \mathsf{F}(rp \wedge nip \wedge natom_{\varphi_{21}})$ and $\langle\!\langle rover, mechanic\rangle\!\rangle \mathsf{F}(rp \wedge nip \wedge patom_{\varphi_{21}})$, respectively. Since there are strategic operators in the latter formulas, then the algorithm produces formulas $\theta_{\mathsf{A}} = \mathsf{AF}(rp \wedge nip \wedge natom_{\varphi_{21}})$ and $\theta_{\mathsf{E}} = \mathsf{EF}(rp \wedge nip \wedge patom_{\varphi_{21}})$. In Algorithm 6, the condition in line 5 is not satisfied, but it is in line 7. So, the output of the procedure is $\bot$, *i.e.*, the formula is false in the model $\mathfrak{M}$.

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). Moreover, 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 3. *Algorithm 6 requires polynomial space w.r.t. the size of φ and of $\mathfrak{M}$.*

Proof. In Algorithm 7, the cost of the loop in lines 1-3 is polynomial in the size of the model. Since the complexity of *Replace*() is polynomial in the size of the formula, then the cost of the loop in lines 5-10 is polynomial in the size of the formula and in the size of the model. In Algorithm 6, the procedure $FromATLtoCTL()$ is polynomial in the size of the formula. The CTL* model checking problem is *PSPACE* (Vardi & Wolper, 1994) (lines 5 and 7). Then, the total complexity is *PSPACE*.

□

5.7 Phase 6: Runtime Verification

Now, we focus on the procedure *RuntimeVerification*(). In previous steps, the sub-models satisfying sub-properties φ of θ are generated, and listed into the set *result*. In Algorithm 8, we report the procedure performing runtime verification on the system. Such algorithm gets in input the model $\mathfrak{M}$, an ATL* property θ to verify, an history h of $\mathfrak{M}$, and the set *result* containing the sub-properties of θ that have been checked on sub-models of $\mathfrak{M}$. This procedure generates a tuple $\langle k, \theta_{mc}, \theta_{rv}, \theta_{unchk}\rangle$, where k is a truth value, θ_{mc} is the set of sub-formulas verified via model checking, θ_{rv} is the set of sub-formulas verified via runtime verification, and θ_{unchk} is the set of sub-formulas that cannot be verified in both static and runtime techniques. First, in line 2, $\mathfrak{M}$ and θ are updated according to the atoms listed in *result*. This step is used to identify in $\mathfrak{M}$ and θ which sub-formulas have already been verified through *CheckSub-formulas*(). Note that, *UpdateModel&Formula* produces two new ATL* formulas $(\theta_\bullet, \theta_\circ)$, which correspond to the updated version of θ for the negative and positive sub-models, respectively.

Once $\theta_\bullet$ and $\theta_\circ$ have been generated, they need to be converted into their corresponding LTL representations to be verified at runtime. This translation is obtained by removing the strategic operators, while leaving the temporal ones (and the atoms). The resulting two new LTL properties $\psi_\bullet$ and $\psi_\circ$ are so obtained (lines 7-8). Finally, by having these two LTL properties, the algorithm proceeds generating (using the standard LTL monitor generation algorithm (Bauer, Leucker, & Schallhart, 2011)) the corresponding monitors $Mon^{\mathfrak{M}'}_{\psi_\bullet}$ and $Mon^{\mathfrak{M}'}_{\psi_\circ}$. Such monitors are then used by Algorithm 8 to check $\psi_\bullet$ and $\psi_\circ$ over an execution trace h given in input. Analysing h the monitor can conclude the satisfaction (resp., violation) of the LTL property under analysis (w.r.t. the model $\mathfrak{M}'$). However, only certain results can actually be considered valid. Specifically, when $Mon^{\mathfrak{M}'}_{\psi_\bullet}(h) = \top$, or when $Mon^{\mathfrak{M}'}_{\psi_\circ}(h) = \bot$. The other cases, which may include the inconclusive verdict (?), are considered undefined, since nothing can be concluded at runtime. The reason why the conditions in lines 11-14 are enough to conclude $\top$ and $\bot$ directly follow from the preservation results presented in Lemma 5. The rest of the algorithm is only for storing how the sub-formulas have been verified, whether at runtime (*i.e.*, stored in θ_{rv}), at static time (*i.e.*, stored in θ_{mc}), or not at all (*i.e.*, stored in θ_{unchk}). Specifically, in lines 16-19, the algorithm iterates on all sub-formulas that have not been verified at static time. For each sub-formula, first, we obtain the corresponding LTL version (line 17), then we synthesise the corresponding monitor (line 18), and finally we use the latter to verify the sub-formula against the history h. If the monitor returns a conclusive verdict, the sub-formula remains in the set θ_{rv}, otherwise the latter is removed from θ_{rv} and added to θ_{unchk}. At the end of the loop, the tuple $\langle k, \theta_{mc}, \theta_{rv}, \theta_{unchk}\rangle$ is returned (line 20).

Before concluding, we show a preservation result from LTL to ATL*. This is the key to guarantee that the results returned in line 12 and 14 make sense in terms of ATL* formulas.

Lemma 5. *Given a model $\mathfrak{M}$ and an ATL* formula φ, for any history h of $\mathfrak{M}$ starting in s_I, we have that:*

1. $Mon_{\varphi_{LTL}}(h) = \top \;\Rightarrow\; (\mathfrak{M}, s_I) \models \varphi_{\mathtt{Ag}}$
2. $Mon_{\varphi_{LTL}}(h) = \bot \;\Rightarrow\; (\mathfrak{M}, s_I) \not\models \varphi_{\emptyset}$

where φ_{LTL} is the variant of φ where all strategic operators are removed, $\varphi_{\mathtt{Ag}}$ is the variant of φ where all strategic operators are converted into $\langle\langle \mathtt{Ag} \rangle\rangle$, $\varphi_\emptyset$ is the variant of φ where all strategic operators are converted into $\langle\langle \emptyset \rangle\rangle$.

Algorithm 8 *RuntimeVerification* ($\mathfrak{M}$, θ, h, $result$)

$k = ?$;
$\langle \mathfrak{M}', \langle \theta_\bullet, \theta_\circ \rangle \rangle = UpdateModel\&Formula(\mathfrak{M}, \theta, result)$;
$\theta_{mc} = \emptyset$;
for $\langle s, \varphi, atom \rangle \in result$ **do**
    $\theta_{mc} = \theta_{mc} \cup \varphi$;
$\theta_{rv} = SubFormulas(\theta) \setminus \theta_{mc}$;
$\psi_\bullet = FromATLtoLTL(\theta_\bullet)$;
$\psi_\circ = FromATLtoLTL(\theta_\circ)$;
$Mon^{\mathfrak{M}'}_{\psi_\bullet} = GenerateMonitor(\psi_\bullet)$;
$Mon^{\mathfrak{M}'}_{\psi_\circ} = GenerateMonitor(\psi_\circ)$;
if $Mon^{\mathfrak{M}'}_{\psi_\bullet}(h) = \top$ **then**
    return $\langle \top, \theta_{mc}, \theta_{rv}, \emptyset \rangle$;
if $Mon^{\mathfrak{M}'}_{\psi_\circ}(h) = \bot$ **then**
    return $\langle \bot, \theta_{mc}, \theta_{rv}, \emptyset \rangle$;
$\theta_{unchk} = \emptyset$;
for $\varphi' \in \theta_{rv}$ **do**
    $\varphi' = FromATLtoLTL(\varphi')$
    $Mon^{\mathfrak{M}'}_{\varphi'} = GenerateMonitor(\varphi')$;
    if $Mon^{\mathfrak{M}'}_{\varphi'}(h) = ?$ **then** $\theta_{rv} = \theta_{rv} \setminus \varphi'$; $\theta_{unchk} = \theta_{unchk} \cup \varphi'$;
return $\langle k, \theta_{mc}, \theta_{rv}, \theta_{unchk} \rangle$;

Proof. To prove (1), consider the formula $\varphi = \langle\!\langle \Gamma \rangle\!\rangle \psi$, in which $\Gamma \subseteq \mathtt{Ag}$ and ψ is a temporal formula without quantifications. So, $\varphi_{LTL} = \psi$ and $\varphi_{\mathtt{Ag}} = \langle\!\langle \mathtt{Ag} \rangle\!\rangle \psi$. By Definition 8 we know that $Mon_{\varphi_{LTL}}(h) = \top$ if and only if for all path ρ in S^ω we have that $h \cdot \rho$ is in $[\![\varphi_{LTL}]\!]$. Note that, the latter is the set of paths that satisfy ψ, *i.e.*, $[\![\varphi_{LTL}]\!] = \{\rho \mid (\mathfrak{M}, \rho) \models \psi\}$. By Definition 6 we know that $(\mathfrak{M}, \mathtt{s}_I) \models \varphi_{\mathtt{Ag}}$ if and only if there exist a strategy profile $\Sigma_{\mathtt{Ag}}$ such that for all paths ρ in $out(\mathtt{s}_I, \Sigma_{\mathtt{Ag}})$ we have that $(\mathfrak{M}, \rho) \models \psi$. Notice that, since the strategic operator involves the whole set of agents, $out(\mathtt{s}_I, \Sigma_{\mathtt{Ag}})$ is composed by a single path. Thus, to guarantee that $\varphi_{\mathtt{Ag}}$ holds in $\mathfrak{M}$, our objective is to construct from $\mathtt{s}_I$ the history h as prefix of the unique path in $out(\mathtt{s}_I, \Sigma_{\mathtt{Ag}})$. Since we have $\langle\!\langle \mathtt{Ag} \rangle\!\rangle$ as strategic operator, this means that there is a way for the set of agents to construct h starting from $\mathtt{s}_I$ and the set $out(\mathtt{s}_I, \Sigma_{\mathtt{Ag}})$ becomes equal to $\{\rho\}$, where $\rho = h \cdot \rho'$, for any $\rho' \in S^\omega$. From the above reasoning, the result follows. The proof for (2) is a variant of the previous case.

To conclude the proof, note that if we have a formula with more strategic operators then we can use the classic bottom-up approach. □

Remark 2. The above lemma shows a preservation result from RV to ATL* model checking for a given model $\mathfrak{M}$ and formula φ that needs to be discussed. If our monitor returns true, then a sub-formula φ' of φ is satisfied in a negative sub-model and at runtime the formula φ_{LTL} holds on the history h given in input. Thus, by Lemma 5 we have that $\varphi_{\mathtt{Ag}}$ is satisfied in $\mathfrak{M}$. That is, $\varphi_{\mathtt{Ag}}$ substitutes $\mathtt{Ag}$ as coalition for all the strategic operators of φ but the ones in φ'. So, our procedure approximates the truth value of φ by considering the case in which all the agents in the MAS collaborate to achieve the objectives not satisfied in the model checking phase. Notice that, this preservation result also holds in the imperfect information case. In fact, the agents can reproduce an history h by using a uniform strategy profile. Consequently, while in (Belardinelli, Lomuscio, & Malvone, 2019; Belardinelli & Malvone, 2020; Belardinelli et al., 2023) the approximation is given in terms of information and in (Belardinelli, Lomuscio, & Malvone, 2018; Belardinelli et al., 2022)

is given in terms of memory of strategies, via Algorithm 8, we give results by approximating the coalitions. Notice that, this approach is only used every time Algorithm 6 is not able to return a conclusive verdict. So, thanks to Algorithm 8, our procedure produces always results, even partial. This aspect is strongly relevant in concrete scenario in which there is the necessity to have some sort of verification results. For example, in the context of swarm robots (Kouvaros & Lomuscio, 2016), with our procedure we can verify macro properties such as "the system works properly" since we are able to guarantee fully collaboration between agents because this property is relevant and desirable for each agent in the system. The same reasoning described above, can be applied in a complementary way for the case of positive sub-models and the falsity.

Example 5. Given the model in Figure 2, the negative sub-model in Figure 5, and its positive counterpart, consider the following formula:

$$\theta = \langle\langle rover \rangle\rangle \mathsf{F}\, cpa$$

Such an ATL* formula specifies whether it is possible for the rover to have the robotic arm checked. Assume, the procedure is checking the sub-model in Figure 5. In this model, state s_1 is not present and Algorithm 6 cannot conclude a result. So, Algorithm 8 is called. Assume as finite trace $h = \mathsf{s}_I, s_1, s_4, s_6$, and $\theta_\bullet$ and $\theta_\circ$ the ATL* formulas updated w.r.t. sub-formulas verified in the negative and positive sub-models, respectively (line 2). Note that, in this example, we have $\theta_\bullet = \theta_\circ = \theta$, because Algorithm 5 cannot verify the unique strategic sub-formula of θ. At this point, Algorithm 8 generates the corresponding LTL formulas $\psi_\bullet$ and $\psi_\circ$ to be checked (lines 7-8), that is, $\psi_\bullet = \psi_\circ = \mathsf{F}\, cpa$. Then, the procedure synthesises the resulting monitors, and uses them to check the given history h (lines 9-10). The monitor for $\psi_\bullet$ concludes $\top$ when applied to h, because s_1 is present in h, and s_1 is labelled with cpa (line 11). So, the procedure terminates by returning the tuple $\langle \top, \emptyset, \theta, \emptyset \rangle$ (line 12).

To conclude this part, we provide the complexity result.

Theorem 4. *Algorithm 8 terminates in double exponential time w.r.t. the size of θ.*

Proof. The auxiliary functions *UpdateModel&Formula*(), *SubFormulas*(), and *FromATLtoLTL*() are polynomial in the size of the formula and model under exam. The procedure *GenerateMonitor*() is double exponential w.r.t. the size of θ (Bauer et al., 2011). The loop in lines 4-5 is linear w.r.t. the size of the list result. The loop in lines 16-19 calls a polynomial number of times a double exponential procedure (*i.e.*, *GenerateMonitor*()). Then, the total complexity follows. □

5.8 The Overall Model Checking Procedure

Given the algorithms presented in the previous subsections, we can now provide the overall procedure as described in Algorithm 9.

The *ModelCheckingProcedure*() takes in input a model $\mathfrak{M}$, a formula θ, and a history h of $\mathfrak{M}$. First, it calls Algorithm 2, and updates the model and the formula accordingly. Then, it calls the function *Preprocessing*() to replace all negated atoms with new positive atoms inside the model and the formula. After that, the algorithm 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 5), *i.e.*, the model $\mathfrak{M}$ given in input is with perfect information w.r.t. the agents involved in the formula θ, then we can directly call the ATL* model

Algorithm 9 *ModelCheckingProcedure* ($\mathfrak{M}$, θ, h)

1: $FinalResult = \emptyset$;
2: $\langle \mathfrak{R}, \lambda \rangle = ImperfectRecall(\mathfrak{M}, \theta)$
3: $Preprocessing(\mathfrak{R}, \lambda)$;
4: $candidates = FindSub\text{-}models(\mathfrak{R}, \lambda)$;
5: **if** $|candidates| = 1$ **then**
6: **return** $\langle \mathfrak{R} \models_{IR} \lambda, \emptyset \rangle$;
7: **while** *candidates* is not empty **do**
8: extract $\langle \mathfrak{R}^{\bullet}, \mathfrak{R}^{\circ} \rangle$ from *candidates*;
9: $result = CheckSub\text{-}formulas(\langle \mathfrak{R}^{\bullet}, \mathfrak{R}^{\circ} \rangle, \lambda)$;
10: $k = Verification(\mathfrak{R}, \lambda, result)$;
11: **if** $k \neq ?$ **then**
12: **return** $\langle k, \emptyset \rangle$;
13: **else** $FinalResult = RuntimeVerification(\mathfrak{R}, \lambda, h, result) \cup FinalResult$;
14: **return** $\langle ?, FinalResult \rangle$;

checking procedure in case of perfect information and perfect recall. After that, there is a while loop (lines 7-13) that for each candidate checks the sub-formulas true on the sub-models via $CheckSub\text{-}formulas()$ 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 12). Otherwise, the Runtime Verification phase is launched (line 13). In case the condition in line 11 is false for each candidate, then the procedure returns the partial runtime verification results in line 14.

Theorem 5. *Algorithm 9 terminates in double exponential time w.r.t. the size of φ and exponential time w.r.t. the size of $\mathfrak{M}$.*

Proof. In the worst case the while loop in lines 7-13 needs to check all the candidates (*i.e.*, the condition in line 11 is always false) and the size of the set of candidates is equal to the size of the set of states of $\mathfrak{M}$ (*i.e.*, polynomial in the size of $\mathfrak{M}$). So, the total complexity is still determined by the subroutines and follows by Theorems 1, 2, 3, and 4. □

We now prove that our procedure (*i.e.*, Algorithm 9) is sound, we do so in terms of the preservation results presented previously in this paper.

Theorem 6. *Algorithm 9 is sound: if the value k returned is not ?, then $\mathfrak{M} \models \varphi$ iff $k = \top$.*

Proof. Let $\mathfrak{R} = \mathfrak{M}_{\mathtt{Ap}^-}$, we prove that $\mathfrak{R} \models (\varphi)^t$ iff $k = \top$. This is sufficient because of Algorithm 3 and Proposition 3. Suppose that the value returned is different from ?. In particular, either $k = \top$ or $k = \bot$. Suppose, to reach a contradiction, that $\mathfrak{R} \models (\varphi)^t$ and $k = \bot$. By Algorithm 6 and 9, we have that $\mathfrak{R}' \not\models (\varphi)^t_{\mathsf{E}}$. Now, there are two cases: (1) $\mathfrak{R}$ and $\mathfrak{R}'$ are the same models or (2) $\mathfrak{R}$ differs from $\mathfrak{R}'$ for some atomic propositions added to $\mathfrak{R}'$ in line 2 of Algorithm 6. In case (1), we reach a contradiction by a direct application of Lemma 4. In case (2), we know that $\mathfrak{R}$ and $\mathfrak{R}'$ differs for some atomic propositions. Suppose w.l.o.g. that $\mathfrak{R}'$ has only one additional atomic proposition $atom_{\varphi'}$, *i.e.*, $\mathtt{Ap}' = \mathtt{Ap} \cup \{atom_{\varphi'}\}$. The latter means that Algorithm 5 found a positive sub-model $\mathfrak{R}^{\circ}$ such that $(\mathfrak{R}^{\circ}, s) \models \varphi'$, for some state s of $\mathfrak{R}$. Since $(\varphi)^t$ is negation-free, so it is φ', thus by Lemma 3, we obtain that $(\mathfrak{R}, s) \not\models \varphi'$. So, $\mathfrak{R}'$ over-approximates $\mathfrak{R}$, *i.e.*, there could be some states that in $\mathfrak{R}'$ are labeled with $atom'_{\varphi}$ but they do not satisfy φ' in $\mathfrak{R}$. Thus, if $\mathfrak{R}' \not\models (\varphi)^t_{\mathsf{E}}$ then $\mathfrak{R} \not\models (\varphi)^t_{\mathsf{E}}$ and, by Lemma 4, $\mathfrak{R} \not\models (\varphi)^t$, a contradiction. Hence, $k = \top$ as required.

On the other hand, if $k = \top$ then by Algorithm 6 and 9, we have that $\mathfrak{R}' \models (\varphi)^t_{\mathsf{A}}$. Again, there are two cases: (1) $\mathfrak{R}$ and $\mathfrak{R}'$ are the same models or (2) $\mathfrak{R}$ differs from $\mathfrak{R}'$ for some atomic propositions added to $\mathfrak{R}'$ in line 2 of Algorithm 6. In case (1), we conclude by Lemma 4. In case (2), we know that $\mathfrak{R}$ and $\mathfrak{R}'$ differs for some atomic propositions. Suppose w.l.o.g. that $\mathfrak{R}'$ has only one additional atomic proposition $atom_{\varphi'}$, *i.e.*, $\mathtt{Ap}' = \mathtt{Ap} \cup \{atom_{\varphi'}\}$. The latter means that Algorithm 5 found a negative sub-model $\mathfrak{R}^\bullet$ in which $\mathfrak{R}^\bullet, s \models \varphi'$, for some state s of $\mathfrak{R}$. Since φ' is negation-free, by Lemma 2, we know that if $(\mathfrak{R}^\bullet, s) \models \varphi'$ then $(\mathfrak{R}, s) \models \varphi'$. So, $\mathfrak{R}'$ under-approximates $\mathfrak{R}$, *i.e.*, there could be some states that in $\mathfrak{R}'$ are not labeled with $atom'_{\varphi}$ but they satisfy φ' in $\mathfrak{R}$. Thus, if $\mathfrak{R}' \models (\varphi)^t_{\mathsf{A}}$ then $\mathfrak{R} \models (\varphi)^t_{\mathsf{A}}$ and, by Lemma 4, $\mathfrak{R} \models (\varphi)^t$ as required.

□

6. Our Tool

In this section, we present the tool which represents an instantiation of the theory presented in this paper. Such a tool can be found as a publicly available GitHub repository[4].

An overview of the tool is reported in Figure 6. In more detail, the tool takes in input an ATL formula θ to verify, an iCGS on which we want to verify θ, and a trace h denoting an execution of the actual system (the one modelled by the iCGS). Once such inputs are passed to the tool, first the latter synthesises an internal representation of the iCGS, which is denoted as a graph object in Java. Such internal representation is used inside the tool to manipulate the iCGS and work on it. Specifically, since the actual static verification of the ATL formula θ is obtained by exploiting the existing MCMAS[5] model checker, our tool needs to address the translation of the model into the format supported by MCMAS. Indeed, the latter does not support iCGSs by default, but it expects models denoted as Interpreted Systems specified using the ISPL format. Thus, in order to use MCMAS, our tool translates the iCGS, on which we want to perform the verification, into its equivalent Interpreted System. Once this step is performed, the static verification can be achieved (this corresponds to lines 6 and 9 of Algorithm 5). After this step, which corresponds to Algorithm 5, the tool has access to a list of sub-models for which a subset of properties of θ is satisfied. Such a list, as shown in Algorithm 7 is used to update the internal model representation $\mathfrak{M}$ and the formula θ, accordingly. The resulting model and formula are then used, as shown in Algorithm 6, to perform the CTL verification (again exploiting MCMAS). Depending on the result of such verification step (as shown in Algorithm 9), the tool concludes, with a final verdict, or continues with the Runtime Verification step. In case the CTL verification step failed to conclude either the satisfaction or violation of θ, the tool moves on with the implementation of Algorithm 8; where the Runtime Verification step is performed. The latter is obtained by verifying the LTL version of the remaining part of θ. We remind the reader that some sub-properties of θ may have been verified in Algorithm 5, but some may have been left to verify. This step is obtained through monitoring the LTL properties, and it has been implemented by exploiting the LamaConv library[6]. Through such a library, the LTL Runtime Verification is performed; while its result is used to populate the corresponding list of outcomes (as shown in Algorithm 8). Note that, the Runtime Verification step terminates the procedure only in case no more sub-models have to be checked, and the CTL verification step is not capable of concluding.

4. https://github.com/AngeloFerrando/strategyCTLRV
5. https://vas.doc.ic.ac.uk/software/mcmas/
6. https://www.isp.uni-luebeck.de/lamaconv

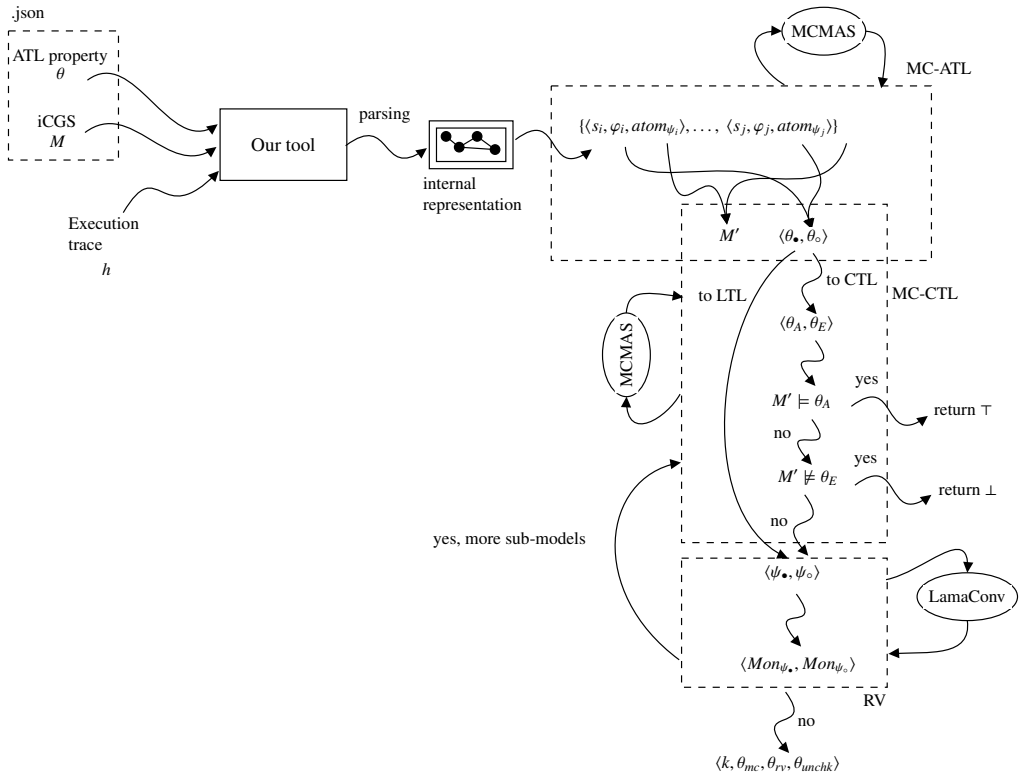

Figure 6: High-level overview of the tool.

In the next sections, we are going to report the results we obtained by experimenting our tool. We show both results w.r.t. the CTL and Runtime Verification instantiations.

6.1 Experiments

We have 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 2 and the three formulas φ_1, φ_2, and φ_3 presented in the same section, Algorithm 4 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 5, and then Algorithm 6 (as shown in Algorithm 9). The resulting implementation takes less than 1 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 2 is expressly small, since it is used as running example to help the reader to understand the contribution. However, the actual rover example we tested on our tool contains hundreds of states and dozens of agents (we have multiple rovers and mechanics in the real example). Such more complex scenario served us as a stressed test to analyse the performance of our tool for more realistic MAS. Since Algorithm 9 is exponential, when the model grows, the performance is highly affected. Nonetheless, even when the model reaches 300 states, the tool concludes in ~10 min.

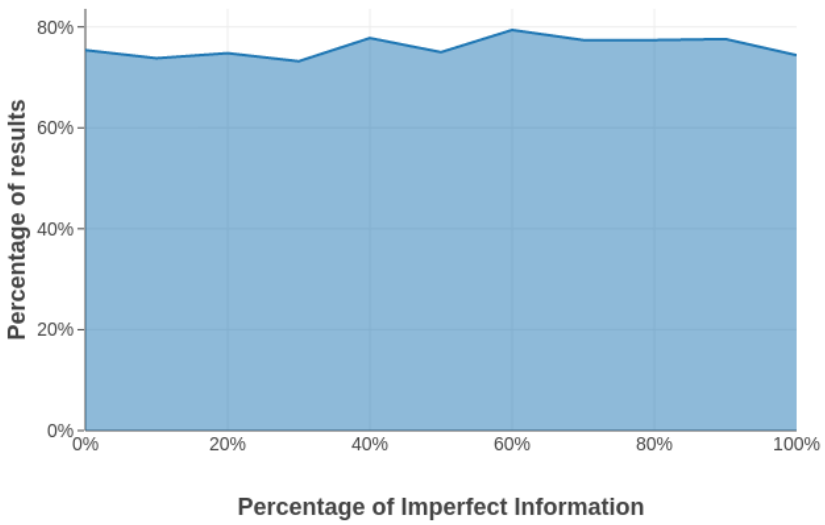

Figure 7: Success rate.

Other than the rover's mission, our tool has been further experimented on a large set of automatically and randomly generated[7] iCGSs. In what follows, we divide the experiments w.r.t. the CTL and RV modules. This is done to analyse more in detail their performances in random scenarios.

Experiments on the CTL Module The objective of these experiments have been to show how many times Algorithm 6 has returned a conclusive verdict. For each model, we have run our procedure and counted the number of times a solution has been returned. Note that, our approach concludes in any case, but since the general problem is undecidable, the result might be inconclusive (*i.e.*, ?). In Figure 7, 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 have generated $10,000$ random iCGSs and counted the number of times our algorithm has returned a conclusive result (*i.e.*, $\top$ or $\bot$). As it can be seen in Figure 7, our algorithm has returned a conclusive verdict for almost the 80% of the models analysed (y axis). It is important to notice how this result has not been 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 have 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 have been generated completely randomly, but a subset of them has been generated considering more interesting topologies (similar to the rover's mission). This has contributed to have more realistic iCGSs, and consequently, more realistic results.

Experiments on the RV Module Other than testing our tool w.r.t. the success rate over a random set of iCGSs, we evaluated the execution time as well. Specifically, we were much interested in

7. Such iCGSs ranged in size from 10 to 50 states, with the number of agents varying between 2 and 8.

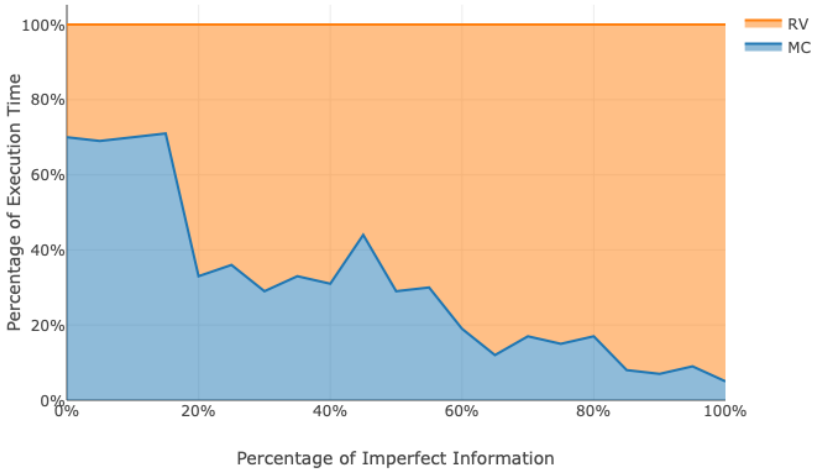

Figure 8: Execution time (static vs dynamic).

analysing how such execution time is divided between *CheckSub-formulas*() and Algorithm 8. *i.e.*, how much time is spent on verifying the models statically (through model checking), and how much is spent on verifying the temporal properties (through runtime verification). Figure 8 reports the results we obtained on the same set of randomly generated iCGSs used in Figure 7. The results we obtained are intriguing, indeed we can note a variation in the percentage of time spent on the two phases (y-axis) moving from low percentages to high percentages of imperfect information in the iCGSs (x-axis). When the iCGS is close to have perfect information (low percentages on x-axis), we may observe that most of the execution time is spent on performing static verification (~70%), which corresponds to *CheckSub-formulas*(). On the other hand, when imperfect information grows inside the iCGS (high percentage on x-axis), we may observe that most of the execution time is spent on performing runtime verification (~90% in occurrence of absence of information). This behaviour is determined by the number of candidates extracted by the *FindSub-models*() function. When the iCGS has perfect information, such function only extracts a single candidate (*i.e.*, the entire model), since *FindSub-models*() generates only one tuple. Such single candidate can be of non-negligible size, and the resulting static verification, time consuming; while the subsequent runtime verification is only performed once on the remaining temporal parts of the property to verify. On the other hand, when the iCGS has imperfect information, *FindSub-models*() returns a set of candidates that can grow exponentially w.r.t. the number of states of the iCGS. Nonetheless, such candidates are small in size, since *FindSub-models*() splits the iCGS into multiple smaller iCGSs with perfect information. Thus, the static verification step is applied on small iCGSs and requires less execution time and the runtime verification step is called for each candidate (so an exponential number of times as in the model checking counterpart). In conclusion, it is important to emphasise that, even though the monitor synthesis is computationally hard (*i.e.*, $2EXPTIME$), the resulting runtime verification process is polynomial in the size of the history analysed. Naturally, the actual running complexity of a monitor depends on the formalism used to describe the formal property. In this work, monitors are synthesised from LTL properties. Since LTL properties are

translated into Moore machines (Bauer et al., 2011), the time complexity w.r.t. the length of the analysed trace is linear. This can be understood intuitively by noticing that the Moore machine so generated has finite size, and it does not change at runtime.

6.2 Benchmarks

In addition to the experiments presented above, we carried out benchmarks considering a variant of the simple voting scenario presented in (Jamroga, Knapik, Kurpiewski, & Mikulski, 2019), and used in (Belardinelli et al., 2022, 2023) to evaluate the partial model checking procedure proposed therein. Furthermore, we carried out experiments on the rover case study with (Belardinelli et al., 2022, 2023) as well.

The voting scenario comprises of ℓ voters, k candidates, and a single coercer. Every voter $i \leq \ell$ votes for one candidate $j \leq k$ (action $vote_{ij}$), and after casting her ballot, voter i can either give a proof of vote to the coercer (action $give_{ij}$), or refrain from doing so (action n_give_i), assuming the proof is trustworthy. The coercer receives the proof (action rec_i), and decides whether to punish voter i or not (actions p_i and n_p_i). The decision is made t timestamps after the proof is submitted by the voter (the coercer delays decision by performing the *wait* action). For the sake of clarity, the iCGS is given in Figure 9 in the case with a single voter, two candidates, and one waiting step. In that case, we have the following indistinguishability relations between different states: $s_6 \sim_{Coercer} s_7$, $s_6 \sim_{Coercer} s_8$, and $s_7 \sim_{Coercer} s_8$. Note that, since we have a single voter, in Figure 9 we omit the index i for the voter's actions.

This scenario and the corresponding iCGS is used in (Belardinelli et al., 2022) to analyse the expressive power of bounded-recall strategies, and in (Belardinelli et al., 2023) to experiment an abstraction refinement mechanism. In particular, the property "there exists a strategy for the coercer such that voter i is not punished if she votes for candidate 1 and provides the proof, otherwise she is punished" (for any $i \leq l$) can be represented in ATL as follows:

$$\theta_1 \quad = \quad \langle\langle Coercer \rangle\rangle F((vote_{i1} \wedge n\_p_i) \vee (\bigvee_{j=2}^{k} vote_{ij} \wedge p_i) \vee (n\_give_i \wedge p_i))$$

We observe that θ_1 is false w.r.t. memoryless strategies, since for this property to hold, the coercer is supposed to perform two different actions in indistinguishable states (the states connected with dotted lines in Figure 9 are indistinguishable for the coercer). However, the coercer has a $(t+1)$-bounded recall strategy (in the sense of (Belardinelli et al., 2022)) to win the game, where t is the number of waiting steps. More formally, we have that $(M, s_0) \models_n \theta_1$ holds iff $n > t$.

In (Belardinelli et al., 2022, 2023) the authors evaluate the ATL specification θ_2 stating that the voters collectively have a strategy to avoid being punished:

$$\theta_2 = \langle\langle all\_voters \rangle\rangle G\neg(\bigvee_{i=1}^{n_v}(n\_give_i \wedge n\_p_i) \vee (\neg n\_give_i \wedge p_i))$$

This voting scenario is a good candidate to compare our verification approach with the ones proposed in (Belardinelli et al., 2022, 2023).

In those works, the authors consider the same problem as we do here, *i.e.*, ATL* model checking under imperfect information and perfect recall. However, (Belardinelli et al., 2022) approximates perfect recall by considering bounded-recall strategies with an increasingly greater bound, while (Belardinelli et al., 2023) approximates imperfect information (by an abstraction refinement

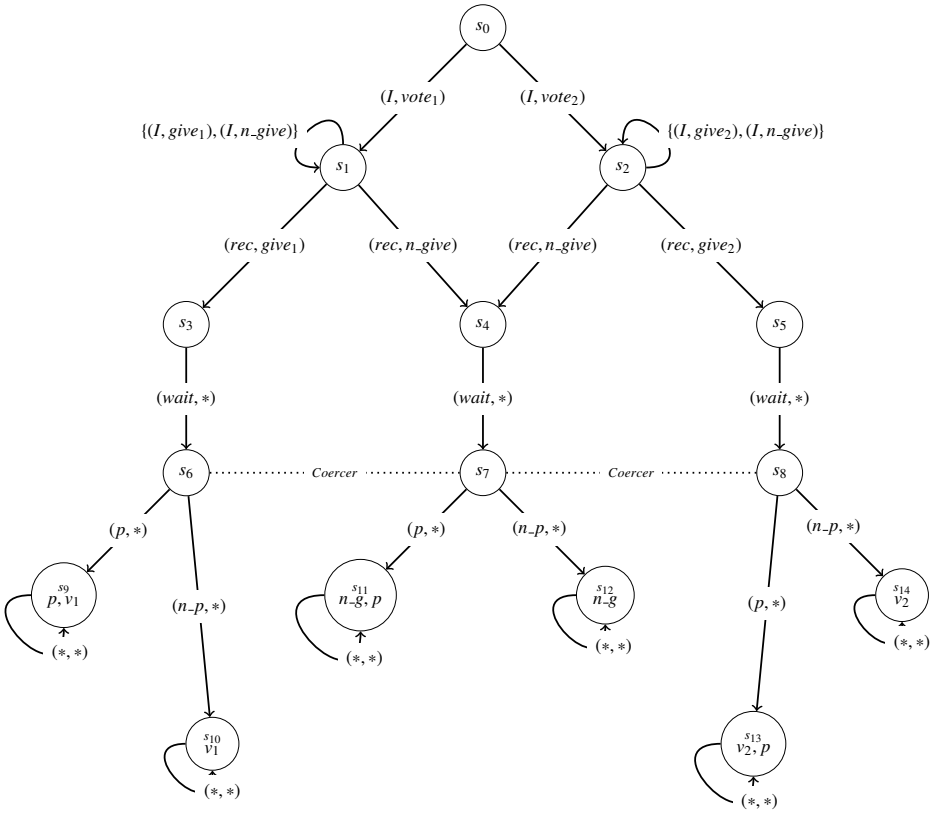

Figure 9: The iCGS $\mathfrak{M}$ for the simple voting scenario. We consider the setting with one waiting step before the Coercer makes a decision for punishment. Here, action $*$ represents any action available for the agent and action I represents the idle action. The states $s_9, \ldots, s_{14}$ are labeled with atomic propositions, where p stands for punish, v_i stands for voter voted for candidate i, and n_g the voter did not give a proof to the coercer. This figure is taken from (Belardinelli et al., 2023).

method). Instead, in here, we propose to study sub-models of iCGSs and present a richer verification mechanism of the latter based upon the combination of CTL* model checking and runtime verification.

Table 1 reports our experimental results w.r.t. the voting scenario previously introduced. We experimented our approach and the ones from (Belardinelli et al., 2022, 2023). Specifically, w.r.t. Figure 9, we tested a more complex scenario involving 2 voters and 1 coercer. The results are divided into three groups (each of 4 rows in Table 1). The first group considers the scenario where we have 6 layers of waiting states. That is, looking at Figure 9, we may see in there only one layer of waiting states (*i.e.*, s_6, s_7, and s_8). In our experiments, we stressed this aspect (as in (Belardinelli et al., 2022, 2023)), and we considered 6, 8, and 10 layers of waiting states. For each of these scenarios, we experimented our approach, and the ones from (Belardinelli et al., 2022, 2023). Note

Voters	Waiting	Bound	Verdict (Belardinelli et al., 2022)	Verdict (Belardinelli et al., 2023)	Verdict (Ours)	(Belardinelli et al., 2022)	(Belardinelli et al., 2023)	Ours
2	6	7	$\bot$	$\bot$	$\bot$	2.8	135.3	3294.3
2	6	9	$\bot$	$\bot$	$\bot$	5.8	135.3	3294.3
2	6	11	$\bot$	$\bot$	$\bot$	11.1	135.3	3294.3
2	6	13	$\bot$	$\bot$	$\bot$	18.1	135.3	3294.3
2	8	7	*uu*	$\bot$	$\bot$	3.2	139.1	3851.6
2	8	9	$\bot$	$\bot$	$\bot$	6.8	139.1	3851.6
2	8	11	$\bot$	$\bot$	$\bot$	12.4	139.1	3851.6
2	8	13	$\bot$	$\bot$	$\bot$	18.5	139.1	3851.6
2	10	7	–	$\bot$	$\bot$	T/O	143.6	4263.9
2	10	9	–	$\bot$	$\bot$	T/O	143.6	4263.9
2	10	11	–	$\bot$	$\bot$	T/O	143.6	4263.9
2	10	13	–	$\bot$	$\bot$	T/O	143.6	4263.9

Table 1: Benchmarks results for the simple voting scenario for property θ_2 (all times are in seconds), where T/O is reported when time superior at 2 hours (timeout).

that, an additional parameter has to be considered, that is the bound used in (Belardinelli et al., 2022); this denotes its maximum bound for the strategies.

The results reported in Table 1 show that our approach is capable of concluding the verification, along with (Belardinelli et al., 2023). While (Belardinelli et al., 2022) suffers from scalability and stops when the layers of waiting states become 10. Note that, even though both conclude with the correct outcome (*i.e.*, the violation of formula θ_2), (Belardinelli et al., 2023) terminates in less time than us.

As we see in Table 2, our approach instead takes less time w.r.t. (Belardinelli et al., 2022, 2023) when applied to the rover case study. This empirically demonstrate that the approaches could be alternated to tackle different kinds of iCGSs. That is, depending on the iCGS structure and topology, one approach could be better suited to perform the verification. This aspect is relevant from a more technological perspective, and it is left as a future work.

Formulas	Bound	Verdict (Belardinelli et al., 2022)	Verdict (Belardinelli et al., 2023)	Verdict (Ours)	(Belardinelli et al., 2022)	(Belardinelli et al., 2023)	Ours
φ_1	1	*uu*	$\bot$	$\bot$	0.54	2.01	1.03
φ_1	2	–	$\bot$	$\bot$	T/O	2.01	1.03
φ_2	1	$\top$	$\top$	$\top$	0.69	3.15	0.91
φ_2	2	–	$\top$	$\top$	T/O	3.15	0.91
φ_3	1	*uu*	$\bot$	$\bot$	1.35	2.10	0.85
φ_3	2	–	$\bot$	$\bot$	T/O	2.10	0.85
φ_4	1	$\top$	*uu*	$\top$	0.03	4.78	0.36
φ_4	2	–	*uu*	$\top$	T/O	4.78	0.36

Table 2: Benchmarks results for Rover scenario (all the times are in seconds), where T/O is reported when time superior at 2 hours (timeout).

It is important to note that, Table 2 does not only show that our approach can be faster than (Belardinelli et al., 2022, 2023), but it also demonstrates empirically that our approach can conclude the verification in cases where (Belardinelli et al., 2022, 2023) cannot. In more detail, this happens for formulas φ_1 and φ_3, where (Belardinelli et al., 2022) is not capable of concluding the verification (the undefined outcome is returned). Instead, our approach correctly concludes the violation of φ_1 and φ_3. Note that, in both cases (and also for φ_2), (Belardinelli et al., 2023) also concludes the correct verification outcome for the formulas. However, to show that our approach can indeed conclude where (Belardinelli et al., 2023) cannot, we also experimented an additional formula:

$$\varphi_4 = \langle\langle rover, mechanic \rangle\rangle \mathsf{F}\, cpw$$

Such ATL formula specifies whether it is possible for the rover and the mechanic (in coalition) to have the robotic wheels checked. Now, even though this formula is quite simple, (Belardinelli et al., 2023) is not capable of concluding its verification with a conclusive outcome (it returns *uu*). Instead, our approach concludes the verification and correctly returns $\top$ (indeed, if the mechanic is in the coalition, φ_4 is satisfied).

Thanks to the set of experiments reported in Table 2, we demonstrated empirically that our approach concludes where (Belardinelli et al., 2022, 2023) conclude, but there are scenarios where our approach concludes and the ones from (Belardinelli et al., 2022, 2023) do not.

7. Related Work

Formal Verification on MAS Various approaches for the verification of specifications in ATL under imperfect information have been recently put forward. In (Lomuscio & Michaliszyn, 2014, 2015, 2016) the authors consider non-uniform strategies for which the corresponding model checking problem is decidable. Their aim, therefore, is rather to speed-up the verification process and not, as we do here, to provide a sound approximation to an otherwise undecidable problem. In (Jamroga et al., 2019) is presented a translation of ATL formulas under imperfect information and imperfect recall strategies that provide lower and upper bounds for their truth values. In another line, restrictions are made on how information is shared amongst the agents, so as to retain decidability (Berthon, Maubert, Murano, Rubin, & Vardi, 2021). In a related line, interactions amongst agents are limited to public actions only (Belardinelli, Lomuscio, Murano, & Rubin, 2017b, 2017a). 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 (Belardinelli et al., 2018, 2022) where an abstraction method over the strategies is defined and to (Belardinelli et al., 2019; Belardinelli & Malvone, 2020; Belardinelli et al., 2023) where an approximation to the information is presented. While the orthogonality between our approach and (Belardinelli et al., 2018, 2022) 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 (Belardinelli et al., 2019; Belardinelli & Malvone, 2020; Belardinelli et al., 2023). With more detail, in (Belardinelli et al., 2019; Belardinelli & Malvone, 2020; Belardinelli et al., 2023) an abstract model abstracts the imperfect information by using a state for each equivalence class, here we do not have an abstraction on the states, that is each state remains as it is in the original model or it is removed. Additionally, while in (Belardinelli et al., 2019; Belardinelli & Malvone, 2020; Belardinelli et al., 2023) 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.

Runtime Verification RV has never been used before in a strategic context, where monitors check whether a coalition of agents satisfies a strategic property. This can be obtained by combining Model Checking on MAS with RV. The combination of Model Checking with RV is not new; in a position paper dating back to 2014, Hinrichs et al. suggested to "model check what you can, runtime verify the rest" (Hinrichs, Sistla, & Zuck, 2014). Their work presented several realistic examples where such mixed approach would give advantages, but no technical aspects were addressed. Desai et al. (Desai, Dreossi, & Seshia, 2017) present a framework to combine model checking and runtime

verification for robotic applications. They represent the discrete model of their system and extract the assumptions deriving from such abstraction. Kejstová et al. (Kejstová, Rockai, & Barnat, 2017) extend an existing software model checker, DIVINE (Barnat, Brim, Havel, Havlíček, Kriho, Lenčo, Ročkai, Štill, & Weiser, 2013), with a runtime verification mode. The system under test consists of a user program in C or C++, along with the environment. Other blended approaches exist, such as a verification-centric software development process for Java making it possible to write, type check, and consistency check behavioural specifications for Java before writing any code (Zimmerman & Kiniry, 2009). Although it integrates a static checker for Java and a runtime assertion checker, it does not properly integrate model checking and RV. In all the previously mentioned works, both Model Checking and RV were used to verify temporal properties, such as LTL. Instead, we focus on strategic properties, we show how combining Model Checking of ATL* properties with RV, and we can give results; even in scenarios where Model Checking alone would not suffice. Because of this, our work is closer in spirit to (Hinrichs et al., 2014); in fact, we use RV to support Model Checking in verifying at runtime what the model checker could not at static time.

8. Conclusion 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 and Runtime Verification 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 models 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. Additionally, we plan to extend our techniques to more expressive languages for strategic reasoning like Strategy Logic (Mogavero, Murano, Perelli, & Vardi, 2014). Furthermore, we are planning to extend the RV part of our work by considering a more predictive flavour (Zhang, Leucker, & Dong, 2012), this can be done by recognising that by verifying at static time part of the system, we can use this information at runtime to predict future events and conclude the runtime verification in advance.

References

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

Baier, C., & Katoen, J.-P. (2008). *Principles of Model Checking (Representation and Mind Series)*. ACM.

Barnat, J., Brim, L., Havel, V., Havlíček, J., Kriho, J., Lenčo, M., Ročkai, P., Štill, V., & Weiser, J. (2013). Divine 3.0 – an explicit-state model checker for multithreaded c & c++ programs. In Sharygina, N., & Veith, H. (Eds.), *Computer Aided Verification*, pp. 863–868, Berlin, Heidelberg. Springer Berlin Heidelberg.

Bartocci, E., Falcone, Y., Francalanza, A., & Reger, G. (2018). Introduction to runtime verification. In Bartocci, E., & Falcone, Y. (Eds.), *Lectures on Runtime Verification - Introductory and Advanced Topics*, Vol. 10457 of *Lecture Notes in Computer Science*, pp. 1–33. Springer.

Bauer, A., Leucker, M., & Schallhart, C. (2011). Runtime verification for LTL and TLTL. *ACM Trans. Softw. Eng. Methodol.*, *20*(4), 14:1–14:64.

Belardinelli, F., Ferrando, A., & Malvone, V. (2023). An abstraction-refinement framework for verifying strategic properties in multi-agent systems with imperfect information. *Artif. Intell.*, *316*, 103847.

Belardinelli, F., Lomuscio, A., & Malvone, V. (2018). Approximating perfect recall when model checking strategic abilities. In Thielscher, M., Toni, F., & Wolter, F. (Eds.), *Principles of Knowledge Representation and Reasoning: Proceedings of the Sixteenth International Conference, KR 2018, Tempe, Arizona, 30 October - 2 November 2018*, pp. 435–444. AAAI Press.

Belardinelli, F., Lomuscio, A., & Malvone, V. (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*, pp. 6030–6037. AAAI Press.

Belardinelli, F., Lomuscio, A., Malvone, V., & Yu, E. (2022). Approximating perfect recall when model checking strategic abilities: Theory and applications. *J. Artif. Intell. Res.*, *73*, 897–932.

Belardinelli, F., Lomuscio, A., Murano, A., & Rubin, S. (2017a). Verification of broadcasting multi-agent systems against an epistemic strategy logic. In Sierra, C. (Ed.), *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI 2017, Melbourne, Australia, August 19-25, 2017*, pp. 91–97. ijcai.org.

Belardinelli, F., Lomuscio, A., Murano, A., & Rubin, S. (2017b). Verification of multi-agent systems with imperfect information and public actions. In Larson, K., Winikoff, M., Das, S., & Durfee, E. H. (Eds.), *Proceedings of the 16th Conference on Autonomous Agents and MultiAgent Systems, AAMAS 2017, São Paulo, Brazil, May 8-12, 2017*, pp. 1268–1276. ACM.

Belardinelli, F., & Malvone, V. (2020). A three-valued approach to strategic abilities under imperfect information. In Calvanese, D., Erdem, E., & Thielscher, M. (Eds.), *Proceedings of the 17th International Conference on Principles of Knowledge Representation and Reasoning, KR 2020, Rhodes, Greece, September 12-18, 2020*, pp. 89–98.

Berthon, R., Maubert, B., Murano, A., Rubin, S., & Vardi, M. Y. (2021). Strategy logic with imperfect information. *ACM Trans. Comput. Log.*, *22*(1), 5:1–5:51.

Desai, A., Dreossi, T., & Seshia, S. A. (2017). Combining model checking and runtime verification for safe robotics. In Lahiri, S. K., & Reger, G. (Eds.), *Runtime Verification - 17th International Conference, RV 2017, Seattle, WA, USA, September 13-16, 2017, Proceedings*, Vol. 10548 of *Lecture Notes in Computer Science*, pp. 172–189. Springer.

Dima, C., & Tiplea, F. L. (2011). Model-checking ATL under imperfect information and perfect recall semantics is undecidable. *CoRR*, *abs/1102.4225*.

Fagin, R., Halpern, J. Y., Moses, Y., & Vardi, M. (1995). *Reasoning about Knowledge.* MIT.

Ferrando, A., & Malvone, V. (2021). Strategy RV: A tool to approximate ATL model checking under imperfect information and perfect recall. In Dignum, F., Lomuscio, A., Endriss, U., & Nowé, A. (Eds.), *AAMAS '21: 20th International Conference on Autonomous Agents and Multiagent Systems, Virtual Event, United Kingdom, May 3-7, 2021*, pp. 1764–1766. ACM.

Ferrando, A., & Malvone, V. (2022). Towards the combination of model checking and runtime verification on multi-agent systems. In Dignum, F., Mathieu, P., Corchado, J. M., & de la Prieta, F. (Eds.), *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*, Vol. 13616 of *Lecture Notes in Computer Science*, pp. 140–152. Springer.

Ferrando, A., & Malvone, V. (2023). Towards the verification of strategic properties in multi-agent systems with imperfect information. In Agmon, N., An, B., Ricci, A., & Yeoh, W. (Eds.), *Proceedings of the 2023 International Conference on Autonomous Agents and Multiagent Systems, AAMAS 2023, London, United Kingdom, 29 May 2023 - 2 June 2023*, pp. 793–801. ACM.

Hinrichs, T. L., Sistla, A. P., & Zuck, L. D. (2014). Model check what you can, runtime verify the rest. In Voronkov, A., & Korovina, M. V. (Eds.), *HOWARD-60: A Festschrift on the Occasion of Howard Barringer's 60th Birthday*, Vol. 42 of *EPiC Series in Computing*, pp. 234–244. EasyChair.

Jamroga, W., Knapik, M., Kurpiewski, D., & Mikulski, L. (2019). Approximate verification of strategic abilities under imperfect information. *Artif. Intell.*, *277*.

Jamroga, W., & van der Hoek, W. (2004). Agents that know how to play. *Fundam. Informaticae*, *63*(2-3), 185–219.

Kejstová, K., Rockai, P., & Barnat, J. (2017). From model checking to runtime verification and back. In Lahiri, S. K., & Reger, G. (Eds.), *Runtime Verification - 17th International Conference, RV 2017, Seattle, WA, USA, September 13-16, 2017, Proceedings*, Vol. 10548 of *Lecture Notes in Computer Science*, pp. 225–240. Springer.

Kouvaros, P., & Lomuscio, A. (2016). Parameterised verification for multi-agent systems. *Artif. Intell.*, *234*, 152–189.

Leucker, M., & Schallhart, C. (2009). A brief account of runtime verification. *J. Log. Algebraic Methods Program.*, *78*(5), 293–303.

Lomuscio, A., & Michaliszyn, J. (2014). An abstraction technique for the verification of multi-agent systems against ATL specifications. In Baral, C., Giacomo, G. D., & Eiter, T. (Eds.), *Principles of Knowledge Representation and Reasoning: Proceedings of the Fourteenth International Conference, KR 2014, Vienna, Austria, July 20-24, 2014*. AAAI Press.

Lomuscio, A., & Michaliszyn, J. (2015). Verifying multi-agent systems by model checking three-valued abstractions. In Weiss, G., Yolum, P., Bordini, R. H., & Elkind, E. (Eds.), *Proceedings of the 2015 International Conference on Autonomous Agents and Multiagent Systems, AAMAS 2015, Istanbul, Turkey, May 4-8, 2015*, pp. 189–198. ACM.

Lomuscio, A., & Michaliszyn, J. (2016). Verification of multi-agent systems via predicate abstraction against ATLK specifications. In Jonker, C. M., Marsella, S., Thangarajah, J., & Tuyls, K.

(Eds.), *Proceedings of the 2016 International Conference on Autonomous Agents & Multiagent Systems, Singapore, May 9-13, 2016*, pp. 662–670. ACM.

Mogavero, F., Murano, A., Perelli, G., & Vardi, M. Y. (2014). Reasoning about strategies: On the model-checking problem. *ACM Trans. Comput. Log.*, *15*(4), 34:1–34:47.

NASA (2024). Mars curiosity rover. https://mars.nasa.gov/msl/home/. Accessed on 2024-05-07.

Pnueli, A. (1977). The temporal logic of programs. In *18th Annual Symposium on Foundations of Computer Science, Providence, Rhode Island, USA, 31 October - 1 November 1977*, pp. 46–57. IEEE Computer Society.

Schobbens, P. (2003). Alternating-time logic with imperfect recall. In van der Hoek, W., Lomuscio, A., de Vink, E. P., & Wooldridge, M. J. (Eds.), *1st International Workshop on Logic and Communication in Multi-Agent Systems, LCMAS 2003, Eindhoven, The Netherlands, June 29, 2003*, Vol. 85 of *Electronic Notes in Theoretical Computer Science*, pp. 82–93. Elsevier.

Vardi, M. Y., & Wolper, P. (1994). Reasoning about infinite computations. *Inf. Comput.*, *115*(1), 1–37.

Zhang, X., Leucker, M., & Dong, W. (2012). Runtime Verification with Predictive Semantics. In *NASA Formal Methods*, Vol. 7226 of *LNCS*, pp. 418–432. Springer.

Zimmerman, D. M., & Kiniry, J. R. (2009). A verification-centric software development process for java. In Choi, B. (Ed.), *Proceedings of the Ninth International Conference on Quality Software, QSIC 2009, Jeju, Korea, August 24-25, 2009*, pp. 76–85. IEEE Computer Society.