An Abstraction-refinement Framework for Verifying Strategic Properties in Multi-agent Systems with Imperfect Information

Francesco Belardinelli[a,b], Angelo Ferrando[c], Vadim Malvone[d]

[a]*Imperial College London, United Kingdom*
[b]*Université d'Evry, France*
[c]*Università di Genova, Italy*
[d]*Télécom Paris, France*

Abstract

We investigate the verification of Multi-Agent Systems against strategic properties expressed in Alternating-time Temporal Logic under the assumptions of imperfect information and perfect recall. To this end, we develop a three-valued semantics for concurrent game structures upon which we define an abstraction method. We prove that concurrent game structures with imperfect information admit perfect information abstractions that preserve three-valued satisfaction. Furthermore, to deal with cases in which the value of a specification is undefined, we develop a novel automata-theoretic technique for the linear-time logic (LTL), then apply it to finding "failure" states. The latter can then be fed into a refinement procedure, thus providing a sound, albeit incomplete, verification method. We illustrate the overall procedure in a variant of the Train Gate Controller scenario and a simple voting protocol under imperfect information and perfect recall. We also present an implementation of our procedure and provide preliminary experimental results.

Keywords: Multi-Agent Systems, Strategic Ability, Alternating-time Temporal Logic, Model Checking, Concurrent Games

1. Introduction

Motivation. Logic-based languages to reason about the strategic abilities of agents are a thriving area of research in the applications of formal methods to artificial intelligence, particularly knowledge reasoning and representation [34, 27]. Over the years, several logics for reasoning about strategies have been introduced, including Alternating-time Temporal Logic [3], Coalition Logic [50], Strategy Logic [21, 49]. The analysis of their verification problems, including their complexity, has also contributed to the development of effective model checking tools [2, 48, 43].

A key challenge for the wider adoption of these logics for strategies is represented by their verification in contexts of imperfect information. Indeed, the model checking problem for the Alternating-time Temporal Logic ATL under the assumption of *perfect*

information is well-known to be PTIME-complete [3], and therefore amenable to automated verification tools. However, under *imperfect information* it ranges between Δ_2^P-completeness and undecidability, depending on the underlying assumptions on memory [35, 25]. Unfortunately, when reasoning about distributed and multi-agent systems (MAS), where agents might only have partial observations of the surrounding environment, the assumption of perfect information is either unrealistic or computationally costly. Thus, if logics for strategies are to be deployed in real-life MAS, it is crucial to develop even partial verification methods capable of tackling contexts of imperfect information. To this end, several methods to retain decidability have been put forward in recent years, focusing on how the information is shared amongst agents [18, 12], or developing notions of constructive knowledge [1] and bounded recall [10, 11], or again approximating strategy operators by using the μ-calculus [20] (see Section 1 for a detailed comparison with related work).

Contribution. We advance the state of the art in reasoning about strategic abilities of agents under the assumptions of imperfect information and perfect recall. More precisely, at the heart of the present contribution is the idea that, under a three-valued semantics, MAS with imperfect information can be approximated (or *abstracted*) by perfect information systems. This enables us to design a sound, albeit incomplete, verification procedure for the strategy logics ATL and ATL^*, under imperfect information and perfect recall. In more detail, given a concurrent game structure with imperfect information (iCGS) representing a MAS, we build a perfect information abstraction that preserves satisfaction for a three-valued variant of ATL^*. As we show, if the ATL^* specification is true (resp. false) in the (perfect information) abstraction, then it is also true (resp. false) in the original iCGS with imperfect information. On the other hand, if the specification is undefined, we can proceed to refining the abstraction in an attempt to give it a defined truth value. In particular, the intermediate step is to find "failure" states to refine the abstract model. In [7] such a procedure was provided but only for the Alternating μ-calculus (AMC) under perfect information. Here we consider the arguably more complex case of full ATL^* under imperfect information, whose model checking problem is undecidable in general, differently from AMC. Moreover, in the process we prove novel results on automata-theoretic techniques for linear-time temporal logic (LTL) interpreted on a three-valued semantics, that we deem of independent interest. To conclude, the original model checking problem is undecidable; so no guarantee can be given that by successive refinements, a given property's truth or falsity can ever be established in general. However, the procedure provides a constructive method to partially model check ATL^* under imperfect information and perfect recall.

Structure of the Paper. In Sec. 2 we present the syntax of ATL^*, as well as its semantics given on concurrent game structures with imperfect information (iCGS), and define the (undecidable) model checking problem when perfect recall is also assumed. In Sec. 3 we introduce the novel three-valued semantics for ATL^* and provide preservation results. In Sec. 4 we define our knowledge-based abstraction procedure, and prove a preservation result from the three-valued to the two-valued semantics. Then, in Sec. 5 we develop novel automata-theoretic techniques for three-valued LTL. Specifically, we show how to construct Büchi automata accepting all traces making an LTL

formula undefined. This result are then used in Sec. 6 to find failure states, which can then be fed into the abstraction refinement algorithm presented in Sec. 7. In Sec. 8, we provide a translation to handle verification under the three-valued semantics in the two-valued model checker MCMAS [48]. Then, in Sec. 9 we present the overall model checking procedure that makes use of the algorithms developed all along the previous sections. Finally, in Sec. 10 we describe an implementation of our methodology and provide experimental results. We conclude in Sec. 11 by summarizing our results and main limitations, and pointing to future work.

Related work

Recently several approaches to the verification of ATL^* under imperfect information and perfect recall have been put forward. Typically, these contributions assume restrictions either on the syntax or the semantics of the specification language, or develop abstraction and approximation methods. In the first line, decidability results have been proved for hierarchical [4, 17] and broadcast systems [12, 13, 14]. In the second line, techniques to construct syntactic [20, 36] and semantic [10, 11, 28, 29] approximations, including partial order reduction [39, 47], have been investigated. Our contribution falls in the second line, specifically semantic approximations, even though it differs from [10, 11], where memory is abstracted to achieve decidability, as here we approximate information instead.

(Multi-valued) Abstraction and Refinement. At the heart of the method we describe is the notion of abstraction and refinement of MAS models, as well as three-valued semantics for modal languages. An abstraction-refinement framework for the temporal logic CTL over a three-valued semantics was first studied in [52, 53], with the specific case of hierarchical systems analyzed in [4]. Further, in [33] an abstraction-refinement technique for the full μ-calculus is introduced. Here we consider the arguably more complex case of ATL^*.

As regards contexts of perfect information, an abstraction-refinement procedure for network games was introduced in [5] and a symbolic abstraction-refinement approach to the solution of two-player games with reachability or safety goals is shown in [24]. Games with incomplete information are studied in [26] by considering safety goals only and, as we do in this paper, abstraction and refinement are used to generate from an imperfect information game, a new one with perfect information.

Model checking MAS by abstraction in an epistemic context was originally investigated in [9, 23]. Three-valued abstractions for the verification of ATL properties have also been put forward in [7, 44, 45, 46]. There are, however, considerable differences between these approaches and the one here pursued. In fact, the methods above focus on decidable settings. In [7, 52] ATL^* is interpreted under perfect information; while [44, 45, 46] considers *non-uniform* strategies [51]. In both cases the corresponding model checking problem is decidable. Their aim, therefore, is rather to speed-up the verification task and not, as we do here, to provide a sound approximation to an otherwise undecidable problem. Moreover, the three-valued semantics for ATL^* is different, as formulas of type $\langle\langle\Gamma\rangle\rangle\psi$ being false is defined in terms of the strategic abilities of opponent coalition $\overline{\Gamma}$ (intuitively, $\overline{\Gamma}$ has a strategy to falsify ψ), rather than by

underapproximating the capabilities of coalition Γ, as we do here. So, the two semantics are really incomparable. Finally, [38, 37] put forward a multi-valued semantics for ATL^* that is a conservative extension of the classical two-valued variant. Mainly, they consider the model checking problem for perfect information games, but they also refer at the imperfect information case by giving an undecidable result in general and an exponential-time result for singleton coalitions.

Three-valued Automata. Regarding the three-valued automata technique for LTL used in this work, the closest related approaches appear in [42, 22, 19]. In particular, [19] consider a reduction from multi-valued to two-valued LTL, but they do not provide automata-theoretic techniques. On the other hand, [22] present an automata-theoretic approach to general multi-valued LTL following the tableau-based construction in [31]. Also, [42] is devoted to general multi-valued automata. Specifically, the authors define lattices, deterministic and non-deterministic automata, as well as their extensions to Büchi acceptance conditions. As an application of their theoretical results, they provide an automata construction for multi-valued LTL, but only in passing, without a clear explanation of states and transitions. To sum up, differently from [42, 22, 19], the approach we propose here modifies minimally the automata-theoretic construction for two-valued LTL [6] and extends it to a three-valued interpretation. In this sense we claim that our contribution is novel w.r.t. the current literature. Furthermore, it is not clear how the techniques in [42, 22, 19] could be used in our construction. As mentioned above, [19] does not really deal with LTL. The approach in [22] is more suitable for on-the-fly verification. Finally, in [42] the authors only briefly discuss model checking, and their approach is tailored for multi-valued logics more generally.

Previous works by the same authors. Some of the material appearing in this paper has already been published in [16, 15]. In [16] we outlined the abstraction-refinement method to solve the model checking problem for ATL^* under imperfect information and perfect recall. Here we extend the contribution in [16] by adding full details and proofs. Moreover, we have fixed several other technical points related to the three-valued semantics, such as the definition of outcome of $must$- and may-strategies in the semantics. In [15] we presented an algorithm to find the failure states needed in the refinement procedure in [16], via the construction of a three-valued automaton for LTL. In this contribution we use the same example as [15] with slight changes about the structure of the game, also adding further examples to guide the reader. Moreover, we have extended the proofs, added examples of three-valued automata (see Examples 3 and 4), and added an additional lemma about the language accepted by our automata (Lemma 4). Also in this case, we fixed several technical points that were not always spelled in detail in [15]. Finally, Sec. 8, 9, and 10 are entirely original of the present submission. In particular, we introduce new algorithms to handle the model checking problem for three-valued semantics via two-valued satisfaction. Then, in Sec. 10 we have introduced an implementation of our verification procedure. No experimental evaluation appears in either [16] or [15].

2. Classic Imperfect Information

In this section we introduce the standard two-valued semantics for the Alternating-time Temporal Logic ATL^* and ATL under imperfect information and perfect recall [40, 25]. To fix the notation, we assume sets $Ag = \{1, \ldots, m\}$ of *agents* and $AP = \{a_1, a_2, \ldots\}$ of *atomic propositions*, or simply atoms. Given a set U, $\overline{U}$ denotes its complement. We denote the length of a tuple v as $|v|$, and its i-th element either as v_i or $v.i$. Let $last(v) = v_{|v|}$ be the last element in v. For $i \leq |v|$, let $v_{\geq i}$ be the suffix $v_i, \ldots, v_{|v|}$ of v starting at v_i and $v_{\leq i}$ its prefix $v_1, \ldots, v_i$. Notice that we start enumerations with index 1.

2.1. Concurrent Game Structures

We begin by giving a formal account of multi-agent systems by means of concurrent game structures with imperfect information [3, 40].

Definition 1 (iCGS). *Given sets Ag of agents and AP of atoms, a* concurrent game structure with imperfect information (iCGS) *is a tuple $M = \langle S, s_0, \{Act_i\}_{i \in Ag}, \{\sim_i\}_{i \in Ag}, d, \delta, V \rangle$ such that:*

- *S is a finite, non-empty set of* states*, with* initial state $s_0 \in S$.
- *For every $i \in Ag$, Act_i is a finite, nonempty set of (individual)* actions*.*

  Let $Act = \bigcup_{i \in Ag} Act_i$ be the set of all actions, and $ACT = \prod_{i \in Ag} Act_i$ the set of all joint actions*, i.e., tuples of individual actions.*
- *For every $i \in Ag$, $\sim_i$ is a relation of* indistinguishability *between states, that is, an equivalence relation on S.*

  Given states $s, s' \in S$, $s \sim_i s'$ iff s and s' are said to be observationally indistinguishable *for agent i.*
- *The* protocol function $d : Ag \times S \to (2^{Act} \setminus \emptyset)$ *defines the availability of actions so that for every $i \in Ag$, $s \in S$, (i) $d(i, s) \subseteq Act_i$ and (ii) $s \sim_i s'$ implies $d(i, s) = d(i, s')$.*
- *The* (deterministic) transition function $\delta : S \times ACT \to S$ *assigns a successor state $s' = \delta(s, \vec{\alpha})$ to each state $s \in S$, for every joint action $\vec{\alpha} \in ACT$ such that $a_i \in d(i, s)$ for every $i \in Ag$, that is, $\vec{\alpha}$ is* enabled *at s.*
- *$V : S \times AP \to \{\mathrm{tt}, \mathrm{ff}\}$ is a* two-valued labelling function*.*

By Def. 1 an iCGS describes the interactions of a group Ag of agents, starting from the initial state $s_0 \in S$, according to the transition function δ. The latter is constrained by the availability of actions to agents, as specified by the protocol function d. Further, we assume that every agent i has imperfect information of the exact state of the system; so in any state s, i considers epistemically possible all states s' that are indistinguishable for i from s [27]. 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 [3]. Hereafter

we consider both the class $iCGS$ of all iCGS, and its subclass CGS of all CGS with perfect information.

Given a set $\Gamma \subseteq Ag$ of agents and a joint action $\vec{\alpha} \in ACT$, let $\vec{\alpha}_\Gamma$ (resp. $\vec{\alpha}_{\overline{\Gamma}}$) be the tuple comprising only of actions for the agents in Γ (resp. $\overline{\Gamma}$). We also write $\vec{\alpha}_i$ and $\vec{\alpha}_{\overline{i}}$ for $\vec{\alpha}_{\{i\}}$ and $\vec{\alpha}_{\overline{\{i\}}}$ respectively. Finally, for $\vec{\alpha}$ and $\vec{\beta}$ in ACT, $(\vec{\alpha}_\Gamma, \vec{\beta}_{\overline{\Gamma}})$ denotes the joint action where the actions for the agents in Γ (resp. $\overline{\Gamma}$) are taken from $\vec{\alpha}$ (resp. $\vec{\beta}$).

A history $h \in S^+$ is a finite (non-empty) sequence of states. The indistinguishability relations are extended to histories in a synchronous, pointwise way, *i.e.*, histories $h, h' \in S^+$ are *indistinguishable* for agent $i \in Ag$, or $h \sim_i h'$, iff (i) $|h| = |h'|$ and (ii) for all $j \leq |h|$, $h_j \sim_i h'_j$.

2.2. Alternating-time Temporal Logic

To reason about the strategic abilities of agents in iCGS with imperfect information, we use the Alternating-time Temporal Logic ATL^* [3].

Definition 2 (ATL^*). *The state (φ) and path (ψ) formulas in ATL^* are defined as follows, where $a \in AP$ and $\Gamma \subseteq Ag$:*

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

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

As customary, 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 linear-time operators *next* X and *until* U is standard [6]. Operators *unavoidable* $[\![\Gamma]\!]$, *release* R, *finally* F, and *globally* G can be introduced as usual.

The formulas in the ATL fragment of ATL^* are obtained from Def. 2 by restricting path formulas ψ as follows, where φ is a state formula and R is the *release* operator:

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

In the rest of the paper we also consider the fragment of Γ-formulas, *i.e.*, formulas in which the strategic operator $\langle\langle\Gamma\rangle\rangle$ ranges only over some fixed coalition $\Gamma \subseteq Ag$.

When giving a semantics to ATL^* formulas we assume that agents are endowed with *uniform strategies* [40], *i.e.*, they perform the same action whenever they have the same information.

Definition 3 (Perfect Recall Uniform Strategy). *A* uniform strategy with perfect recall *for agent $i \in Ag$ is a function $f_i : S^+ \to Act_i$ such that for all histories $h, h' \in S^+$, (i) $f_i(h) \in d(i, last(h))$; and (ii) $h \sim_i h'$ implies $f_i(h) = f_i(h')$.*

By Def. 3 any strategy for agent i has to return actions that are enabled for i. Also, whenever two histories are indistinguishable for i, then the same action is returned. Notice that, for the case of (perfect information) CGS, condition (ii) is satisfied by any function $f_i : S^+ \to Act_i$.

Given an iCGS M, a *path* $p \in S^\omega$ is an infinite sequence $s_1 s_2 \dots$ of states such that, for all $j \geq 1$, $s_{j+1} = \delta(s_j, \vec{\alpha})$ for some joint action $\vec{\alpha}$. Given a joint strategy $F_\Gamma = \{f_i \mid i \in \Gamma\}$, comprising of one strategy for each agent in coalition Γ, a path p is F_Γ-*compatible* iff for every $j \geq 1$, $p_{j+1} = \delta(p_j, \vec{\alpha})$ for some joint action $\vec{\alpha}$ such that for every $i \in \Gamma$, $\alpha_i = f_i(p_{\leq j})$, and for every $i \in \overline{\Gamma}$, $\alpha_i \in d(i, p_j)$. Let $out(s, F_\Gamma)$ be the set of all F_Γ-compatible paths from state s.

We can now assign a meaning to ATL^* formulas on iCGS based on a semantics with two truth values: ff and tt.

Definition 4 (Satisfaction). *The two-valued satisfaction relation $\models^2$ for an iCGS M, state $s \in S$, path $p \in S^\omega$, atom $a \in AP$, state formula φ, and path formula ψ is defined as follows:*

$$
\begin{array}{lll}
(M,s) \models^2 a & \textit{iff} & V(s,a) = \mathrm{tt} \\
(M,s) \models^2 \neg\varphi & \textit{iff} & (M,s) \not\models^2 \varphi \\
(M,s) \models^2 \varphi \wedge \varphi' & \textit{iff} & (M,s) \models^2 \varphi \textit{ and } (M,s) \models^2 \varphi' \\
(M,s) \models^2 \langle\!\langle \Gamma \rangle\!\rangle \psi & \textit{iff} & \textit{for some joint strategy } F_\Gamma, \\
 & & \textit{for all paths } p \in out(s, F_\Gamma),\ (M,p) \models^2 \psi \\
(M,p) \models^2 \varphi & \textit{iff} & (M,p_1) \models^2 \varphi \\
(M,p) \models^2 \neg\psi & \textit{iff} & (M,p) \not\models^2 \psi \\
(M,p) \models^2 \psi \wedge \psi' & \textit{iff} & (M,p) \models^2 \psi \textit{ and } (M,p) \models^2 \psi' \\
(M,p) \models^2 X\psi & \textit{iff} & (M,p_{\geq 2}) \models^2 \psi \\
(M,p) \models^2 \psi U \psi' & \textit{iff} & \textit{for some } k \geq 1,\ (M,p_{\geq k}) \models^2 \psi', \textit{ and} \\
 & & \textit{for all } j,\ 1 \leq j < k \textit{ implies } (M,p_{\geq j}) \models^2 \psi
\end{array}
$$

We say that formula φ is *true* in an iCGS M, or $M \models^2 \varphi$, iff $(M, s_0) \models^2 \varphi$.

Notice that the satisfation clause for the release operator R can be derived as follows, by assuming that $\psi R \psi' ::= \neg(\neg\psi U \neg\psi')$:

$$
\begin{array}{lll}
(M,p) \models^2 \psi R \psi' & \text{iff} & \text{for all } k \geq 1,\ (M,p_{\geq k}) \models^2 \psi', \text{ or} \\
 & & \text{for some } j \geq 1,\ (M,p_{\geq j}) \models^2 \psi, \text{ and} \\
 & & \text{for all } j',\ 1 \leq j' \leq j \text{ implies } (M,p_{\geq j'}) \models^2 \psi'
\end{array}
$$

We observe that the semantics considered here corresponds to the *objective interpretation* of ATL under imperfect information [40], as opposed to the *subjective interpretation*, whereby a strategy has to be successful for all states s' indistinguishable from the current state s. Both interpretations have been extensively analysed in the model theory of logics for strategic reasoning, each of them with its own pros and cons. We refrain here from an in-depth comparison of these accounts and refer to [40] for further details. We here adopt the objective interpretation, as it is consistent with our application scenario and it offers simpler satisfaction clauses, which is helpful, especially when presenting the three-valued semantics and abstraction procedure. We remark that the abstraction and refinement technique presented here can be extended to the subjective interpretation with minor adaptations.

We now state the model checking problem within the two-valued semantics.

Definition 5 (Model Checking). *Given an iCGS M and a formula ϕ, the model checking problem concerns determining whether $M \models^2 \phi$.*

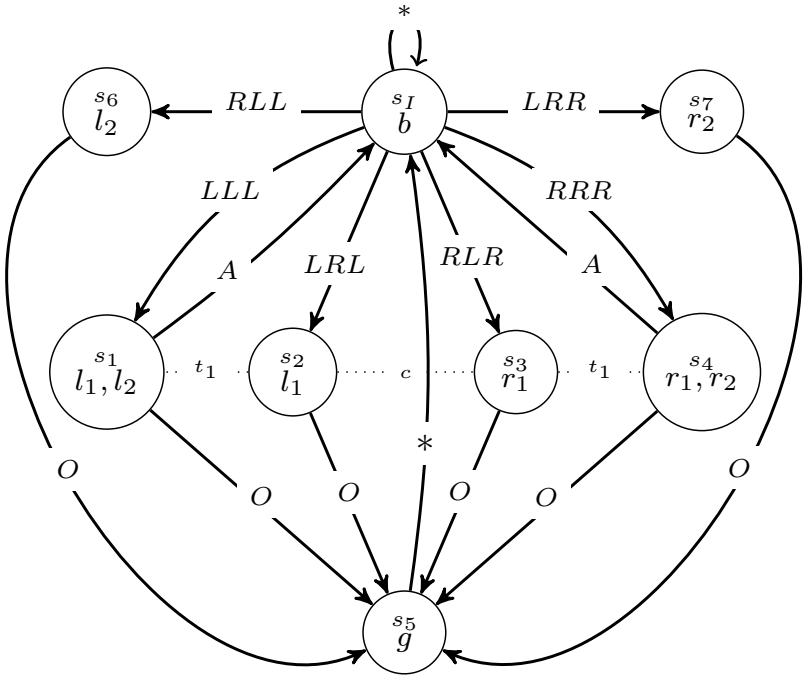

Figure 1: The iCGS M for Example 1. Notice that the transitions are generated by triples of actions. To improve readability, occurrences of the action *idle* (I) are omitted. Moreover, $*$ denotes any joint action for which a transition is not given explicitly. Further, dotted lines are used for indistinguishable states, each one labelled with the relevant agent.

Since the semantics provided in Def. 4 is the standard interpretation of ATL^* [3, 40], it is well known that model checking ATL, *a fortiori* ATL^*, against iCGS with imperfect information and perfect recall is undecidable [25]. In the rest of the paper we develop methods to obtain partial solutions to this problem; but first we illustrate the formal machine above with our running example.

Example 1. *In Fig. 1 we present a coordination game played by two trains t_1 and t_2, and a controller c at a junction. The objective of each train is to cross the junction. To do this, they need to select the same direction as the controller. In this example we analyse the t_1's viewpoint. Train t_1 (resp. t_2) and c need to coordinate and select the same direction, left (action L) or right (action R), to move from the initial state s_I. After this first step, the controller can still change their mind for safety reasons. Specifically, they can request a new selection to the trains (action A) if both of them selected the same direction, or execute it (action O). Further, train t_1 cannot observe the choice of t_2, i.e., if t_1 plays L (resp. R), then she cannot distinguish whether t_2*

selects R or L, and c has partial observability on the choices of t_1 and t_2: she cannot distinguish the identities of t_1 and t_2, that is, she does not distinguish between the result of joint actions RLR and LRL[1]. Finally, we use six atoms: b to label the initial state s_I only; l_1 (resp. l_2) for coordination on the left between c and t_1 (resp. t_2); r_1 (resp. r_2) for coordination on the right between c and t_1 (resp. t_2); and g(oal) to mark that the players have coordinated.

More formally, the iCGS M is comprised of the agents in $Ag = \{t_1, t_2, c\}$, atoms in $AP = \{b, l_1, l_2, r_1, r_2, g\}$, states in $S = \{s_I, s_1, s_2, s_3, s_4, s_5, s_6, s_7\}$ with initial state s_I, actions in $Act_{t_1} = Act_{t_2} = \{L, R, I\}$, $Act_c = \{A, O, I, L, R\}$. Transitions are given as in Fig. 1, and we have the following indistinguishability relations between different states (indistinguishability is reflexive as well): $s_1 \sim_{t_1} s_2$, $s_3 \sim_{t_1} s_4$, and $s_2 \sim_c s_3$. The transitions of the iCGS M are depicted in Fig. 1.

As an example of specification in ATL^*, consider formula $\varphi = \langle\langle\Gamma\rangle\rangle F(l_1 \wedge \neg bUg)$, for $\Gamma = \{t_1, c\}$, which can be read as: t_1 and c have a joint strategy such that eventually they tentatively coordinate on the left, but then an agreement has to be reached before visiting the initial state again. Notice that φ is true in M by the joint strategy whereby t_1 chooses L in s_I and I in all other states, and c chooses L in s_I and O in all other states. However, we want to be able to model check such specifications in general.

3. Three-valued Imperfect Information

In this section we introduce a generalisation of iCGS in terms of over- and under-approximations. Then, we develop a three-valued semantics for ATL^*, and show that it conservatively extends the two-valued semantics in the previous section. In the rest of the paper, for $x = may$ (resp. $must$), we set $\overline{x} = must$ (resp. may).

Definition 6 (Generalized iCGS). *Given sets Ag of agents and AP of atoms, a* generalized iCGS *is a tuple $M = \langle S, s_0, \{Act_i\}_{i\in Ag}, \{\sim_i\}_{i\in Ag}, d^{may}, d^{must}, \delta^{may}, \delta^{must}, V\rangle$ such that:*

1. *S, s_0, $\{Act_i\}_{i\in Ag}$, $\{\sim_i\}_{i\in Ag}$ are defined as in Def. 1.*
2. *d^{may} and d^{must} are* protocol functions *from $Ag \times S$ to $2^{Act} \setminus \emptyset$ such that for every $i \in Ag$ and $s \in S$, (i) $d^{must}(i, s) \subseteq d^{may}(i, s) \subseteq Act_i$ and (ii) $s \sim_i s'$ implies $d^x(i, s) = d^x(i, s')$.*
3. *δ^{may} and δ^{must} are* transition relations *on $S \times ACT \times S$ such that $s' \in \delta^x(s, \vec{\alpha})$ is defined for some $s' \in S$ only if $a_i \in d^x(i, s)$ for every $i \in Ag$. Moreover, $\delta^{must}(s, \vec{\alpha}) \subseteq \delta^{may}(s, \vec{\alpha})$.*
4. *$V : S \times AP \to \{\mathrm{tt}, \mathrm{ff}, \mathrm{uu}\}$ is a* three-valued labelling function.

[1]Note that, for simplicity, we considered imperfect information only on the side of the coordination between t_1 and c. To make the iCGS in Fig. 1 symmetric we only need to add $s_1 \sim_{t_2} s_6$, $s_4 \sim_{t_2} s_7$, and $s_6 \sim_c s_7$, as well as reflexive points.

Intuitively, $must$-components are more restrictive than may-components: $must$-transitions can be interpreted as under-approximations of the actual transitions in the iCGS, while may-transitions can be thought of as over-approximations. The undefined value uu can be interpreted in various ways, for instance, unknown, unspecified, or inconsistent, depending on the application at hand. These interpretations all appear in multi-valued abstraction-based methods [52, 7], we do not discuss this point further. We say that the truth value τ is *defined* whenever $\tau \neq \mathrm{uu}$. In the case that under- and over-approximations coincide, *i.e.*, $d^{may} = d^{must}$ and $\delta^{may} = \delta^{must}$, and the truth value of every atom is defined, then we have a standard iCGS as per Def. 1. On the other hand, if each equivalence relation $\sim_i$ is the identity, then we have a generalized CGS (with perfect information).

Next, we introduce $must$- and may-strategies.

Definition 7 (Perfect Recall Uniform x-Strategy). *For $x \in \{may, must\}$, a* uniform x-strategy with perfect recall *for agent $i \in Ag$ is a function $f_i^x : S^+ \to Act_i$ such that for every history $h, h' \in S^+$, (i) $f_i^x(h) \in d^x(i, last(h))$; and (ii) $h \sim_i h'$ implies $f_i^x(h) = f_i^x(h')$.*

Here we distinguish between may and $must$ strategies to over- and under-approximate the strategic abilities of agents. Again, the distinction collapses in the case of standard (two-valued) iCGS.

For $x \in \{may, must\}$ and a joint strategy $F_\Gamma^x = \{f_i^x \mid i \in \Gamma\}$, a path $p \in S^\omega$ is F_Γ^x-*compatible* iff for every $j \geq 1$, $p_{j+1} = \delta^{\overline{x}}(p_j, \vec{\alpha})$ for some joint action $\vec{\alpha}$ such that for every $i \in \Gamma$, $\alpha_i = f_i^x(p_{\leq j})$, and for every $i \notin \Gamma$, $\alpha_i \in d^{\overline{x}}(i, p_j)$. Then, $out(s, F_\Gamma^x)$ is the set of all F_Γ^x-compatible paths starting from s, that is:

$$out(s, F_\Gamma^x) \;=\; \{p \in S^\omega \mid \text{for all } j \geq 0, p_{j+1} \in \delta^{\overline{x}}(p_j, (F_\Gamma^x(p_{\leq j}), \vec{\alpha}_{\overline{\Gamma}})) \text{ and for all } i \in \overline{\Gamma}, \alpha_i \in d^{\overline{x}}(i, p_j)\}$$

Intuitively, when computing the outcomes of a joint strategy F_Γ^{must} from state s, we adopt a "conservative" stance with respect to the abilities of agents in Γ, by considering only actions enabled according to the under-approximated protocol d^{must}, as well as an "optimistic" stance about the capabilities of agents in $\overline{\Gamma}$, as given by the over-approximated protocol d^{may} and transition δ^{may}. For $out(s, F_\Gamma^{may})$ the reasoning is reversed. However, since F_Γ^{may} returns may-actions and paths in $out(s, F_\Gamma^{may})$ are generated by considering the δ^{must}-transitions, $out(s, F_\Gamma^{may})$ may be empty. By the definition of the semantics below, in such cases the formula will be undefined. This modelling choice is in line with similar three-valued semantics for logics of strategies [7, 46].

Formally we define the three-valued semantics for ATL^* as follows.

Definition 8 (Satisfaction). *The three-valued satisfaction relation $\models^3$ for an iCGS M, state $s \in S$, path $p \in S^\omega$, atom $a \in AP$, $v \in \{\mathrm{tt}, \mathrm{ff}\}$, state formula φ, and path formula ψ is defined as follows:*

$$\begin{array}{lll}
((M,s) \models^3 a) = v & \textit{iff} & V(s,a) = v \\
((M,s) \models^3 \neg\varphi) = v & \textit{iff} & ((M,s) \models^3 \varphi) = \neg v \\
((M,s) \models^3 \varphi \wedge \varphi') = \mathrm{tt} & \textit{iff} & ((M,s) \models^3 \varphi) = \mathrm{tt} \textit{ and } ((M,s) \models^3 \varphi') = \mathrm{tt}
\end{array}$$

$((M,s) \models^3 \varphi \wedge \varphi') = \mathrm{ff}$ *iff* $((M,s) \models^3 \varphi) = \mathrm{ff}$ *or* $((M,s) \models^3 \varphi') = \mathrm{ff}$

$((M,s) \models^3 \langle\langle\Gamma\rangle\rangle\psi) = \mathrm{tt}$ *iff* *for some joint strategy* F^{must}_Γ, *for all paths* $p \in out(s, F^{must}_\Gamma)$, $((M,p) \models^3 \psi) = \mathrm{tt}$

$((M,s) \models^3 \langle\langle\Gamma\rangle\rangle\psi) = \mathrm{ff}$ *iff* *for every joint strategy* F^{may}_Γ, *for some path* $p \in out(s, F^{may}_\Gamma)$, $((M,p) \models^3 \psi) = \mathrm{ff}$

$((M,p) \models^3 \varphi) = v$ *iff* $((M,p_1) \models^3 \varphi) = v$

$((M,p) \models^3 \neg\psi) = v$ *iff* $((M,p) \models^3 \psi) = \neg v$

$((M,p) \models^3 \psi \wedge \psi') = \mathrm{tt}$ *iff* $((M,p) \models^3 \psi) = \mathrm{tt}$ *and* $((M,p) \models^3 \psi') = \mathrm{tt}$

$((M,p) \models^3 \psi \wedge \psi') = \mathrm{ff}$ *iff* $((M,p) \models^3 \psi) = \mathrm{ff}$ *or* $((M,p) \models^3 \psi') = \mathrm{ff}$

$((M,p) \models^3 X\psi) = v$ *iff* $((M,p_{\geq 2}) \models^3 \psi) = v$

$((M,p) \models^3 \psi U \psi') = \mathrm{tt}$ *iff* *for some* $k \geq 1$, $((M,p_{\geq k}) \models^3 \psi') = \mathrm{tt}$, *and for all* j, $1 \leq j < k \Rightarrow ((M,p_{\geq j}) \models^3 \psi) = \mathrm{tt}$

$((M,p) \models^3 \psi U \psi') = \mathrm{ff}$ *iff* *for all* $k \geq 1$, $((M,p_{\geq k}) \models^3 \psi') = \mathrm{ff}$, *or for some* $j \geq 1$, $((M,p_{\geq j}) \models^3 \psi) = \mathrm{ff}$, *and for all* j', $1 \leq j' \leq j$ *implies* $((M,p_{\geq j'}) \models^3 \psi') = \mathrm{ff}$

In all other cases the value of ϕ *is* uu.

Observe that, in the clauses for ATL^* operators $must$-strategies are used to check the truth of formulas, while may-strategies appear in the clauses for falsehood. Specifically, to check whether $((M,s) \models^3 \langle\langle\Gamma\rangle\rangle\psi) = \mathrm{tt}$ we consider all paths in $out(s, F^{must}_\Gamma)$, which are defined by δ^{may}-transitions. This restricts the choices available to coalition Γ, while increasing the number of paths in which the formula needs to be satisfied. Similarly, to verify whether $((M,s) \models^3 \langle\langle\Gamma\rangle\rangle\psi) = \mathrm{ff}$ we need to use δ^{must}-transitions over the paths in $out(s, F^{may}_\Gamma)$, so as to decrease the number of candidates witnessing the falsehood of the formula. Notice also that, as regards Boolean operators, our semantics correspond to Kleene's three-valued logic [41]. Further, for ATL, the truth value of path formulas $\varphi_1 R \varphi_2$ can be derived as follows:

$((M,p) \models^3 \psi R \psi') = \mathrm{tt}$ iff for all $k \geq 1$, $((M,p_{\geq k}) \models^3 \psi') = \mathrm{tt}$, or for some $j \geq 1$, $((M,p_{\geq j}) \models^3 \psi) = \mathrm{tt}$, and for all j', $1 \leq j' \leq j$ implies $((M,p_{\geq j'}) \models^3 \psi') = \mathrm{tt}$

$((M,p) \models^3 \psi R \psi') = \mathrm{ff}$ iff for some $k \geq 1$, $((M,p_{\geq k}) \models^3 \psi') = \mathrm{ff}$, and for all j, $1 \leq j < k$ implies $((M,p_{\geq j}) \models^3 \psi) = \mathrm{ff}$

Also, in what follows we will use the following (equivalent) clause for the until operator U:

$((M,p) \models^3 \psi U \psi') = \mathrm{ff}$ iff for all $k \geq 1$, either $((M,p_{\geq k}) \models^3 \psi') = \mathrm{ff}$, or for some $j < k$, $((M,p_{\geq j}) \models^3 \psi) = \mathrm{ff}$.

Finally, $(M \models^3 \varphi) = \mathrm{tt}$ (resp. ff) iff $((M,s_0) \models^3 \varphi) = \mathrm{tt}$ (resp. ff). Otherwise, $(M \models^3 \varphi) = \mathrm{uu}$.

We conclude this section by proving some auxiliary results on conservative extensions and the model checking problem.

Lemma 1 (Conservativeness). *Let M be a standard iCGS, that is, $d^{may} = d^{must}$, $\delta^{may} = \delta^{must}$ are functions, and the truth value of every atom is defined (i.e., it is*

equal to either tt *or* ff*). Then, for every formula* ϕ *in* ATL^**,*

$$((M,s) \models^3 \phi) = \mathrm{tt} \quad \Leftrightarrow \quad (M,s) \models^2 \phi \tag{1}$$
$$((M,s) \models^3 \phi) = \mathrm{ff} \quad \Leftrightarrow \quad (M,s) \not\models^2 \phi \tag{2}$$

Proof. The result is proved by induction on the structure of formula ϕ. The base case for atoms and the inductive cases for Boolean operators are immediate. The case for formulas of type $\langle\langle\Gamma\rangle\rangle\psi$ follows directly from the fact that sets F_Γ^{may} and F_Γ^{must} of strategies coincide in M, whenever M is a (two-valued) iCGS, and therefore the two- and three-valued semantics collapse. □

By Lemma 1 the three-valued semantics for ATL^* is a conservative extension of its two-valued semantics, as the two coincide whenever we consider standard iCGS with defined atoms. Thus, from the results in the previous section it immediately follows that model checking ATL^* formulas under the three-valued semantics, with imperfect information and perfect recall is also undecidable. However, for perfect information we can show the following.

Theorem 1. *The model checking problem for generalized CGS (with perfect information) is 2EXPTIME-complete for* ATL^* *and PTIME-complete for* ATL*.*

Proof. The lower bounds follow immediately from the corresponding complexity results for two-valued CGS, of which these problems are particular instances, as shown in Lemma 1.

As regards the upper bound for ATL, consider a generalised iCGS M and an ATL formula φ. Similarly to standard labelling algorithms for model checking two-valued ATL, we label each state s in M with pairs (ψ, v), where ψ is a subformulas of φ and $v \in \{\mathrm{tt}, \mathrm{ff}, \mathrm{uu}\}$. We do this in a bottom-up fashion, starting from the innermost state subformulas of φ. For the base case of atoms and the inductive case of Boolean operators, the labeling procedure is immediate. The interesting cases are the strategy operators. Suppose that we are given a subformula of type $\langle\langle A\rangle\rangle(\varphi_1 U \varphi_2)$ and assume that the states in M have already been labelled with φ_1, φ_2. Then, we can use the model checking procedure in [3] for $\langle\langle A\rangle\rangle(\varphi_1 U \varphi_2)$, if $v = \mathrm{tt}$. For $v = \mathrm{ff}$, we use the procedure to check formulas of type $\langle\langle A\rangle\rangle(\varphi_1 R \varphi_2)$, while considering the sets of states where φ_1 and φ_2 are false. Since we have determined the set $S_{\varphi=\mathrm{tt}}$ of states that satisfy formula φ, and the set $S_{\varphi=\mathrm{ff}}$ of those that do not, then the states where the formula is undefined (uu) are those in $S \setminus (S_{\varphi=\mathrm{tt}} \cup S_{\varphi=\mathrm{ff}})$. Since we invoke polynomial procedures at most $\mathcal{O}(|\varphi|)$ times, the complexity remains in PTIME.

As for the upper bound for ATL^*, again we use a labelling algorithm working bottom-up on the structure of formula φ to be checked over a model M. The inductive cases for Boolean operators are immediate. As regards the inductive case of strategy subformulas $\varphi' = \langle\langle\Gamma\rangle\rangle\psi$, we adapt the algorithm for model checking two-valued ATL^* [3] as follows. For $\varphi' = \langle\langle\Gamma\rangle\rangle\psi$, we can assume that ψ is a formula in LTL, as the satisfaction of all state subformulas of ψ has already been determined. To check for $v = \mathrm{tt}$, we construct a Rabin tree automaton $\mathcal{A}_\psi$ that accepts precisely the trees that satisfy the CTL^* formula $A\psi$, and for each state s in M, we construct a Büchi tree automaton $\mathcal{A}_{M,s,\Gamma}$ that accepts precisely the (s,Γ)-execution trees, that is, the trees in

M that correspond to some $must$-strategy for coalition Γ, executed from state s. The product of the two automata $\mathcal{A}_\psi$ and $\mathcal{A}_{M,s,\Gamma}$ is a Rabin tree automaton that accepts precisely the (s,Γ)-execution trees that satisfy $A\psi$. Now, recall that $((M,s) \models \varphi') = \mathrm{tt}$ iff there is a joint strategy F_Γ^{must} for the agents in Γ so that all s-paths in M that are outcomes of F_Γ^{must} satisfy ψ. Since each (s,Γ)-execution tree corresponds to a joint strategy F_Γ^{must}, it follows that $((M,s) \models \varphi') = \mathrm{tt}$ iff the product automaton is nonempty.
As for $v = \mathrm{ff}$, we construct a Rabin tree automaton $\mathcal{A}_\psi$ that accepts precisely the trees for which the CTL^* formula $A\psi$ is false. Then we consider the complement Streett tree automaton, which then accepts the trees for which $A\psi$ is either true or undefined. Further, for each state s in M, we construct a Büchi tree automaton $\mathcal{A}_{M,s,\Gamma}$ that accepts precisely the (s,Γ)-execution trees. The product of the two automata $\mathcal{A}_\psi$ and $\mathcal{A}_{M,s,\Gamma}$ is a Streett tree automaton that accepts precisely the (s,Γ)-execution trees that make $A\psi$ either true or undefined. Now, recall that $((M,s) \models \varphi') = \mathrm{ff}$ iff for every joint strategy F_Γ^{may} for the agents in Γ, there is some s-path in M that is an outcome of F_Γ^{may} falsifying ψ. Since each (s,Γ)-execution tree corresponds to a joint strategy F_Γ^{may}, it follows that $((M,s) \models \varphi') = \mathrm{ff}$ iff the product automaton is empty.

For $v = \mathrm{uu}$, we combine the cases for $v = \mathrm{tt}$ and $v = \mathrm{ff}$. The different procedures are all in 2EXPTIME. □

In the following section we leverage on the decidable model checking problem for the three-valued semantics under perfect information to develop a sound, albeit incomplete, abstraction-based method to verify imperfect information.

4. Abstraction

We now define perfect information, three-valued abstractions for iCGS. Then, we show that defined truth values for ATL^* formulas transfer from such abstractions to the original iCGS with imperfect information. Since the model checking problem on the former is decidable (as per Theorem 1), this preservation result can be used to define a sound, albeit partial, verification procedure under imperfect information and perfect recall.

To begin with, given a coalition $\Gamma \subseteq Ag$ of agents, we define the *common knowledge relation* $\sim_\Gamma^C$ as the reflexive and transitive closure $(\bigcup_{i\in\Gamma} \sim_i)^*$ of the union of indistinguishability relations $\sim_i$ for $i \in \Gamma$ [27]. That is, $s \sim_\Gamma^C s'$ iff s' is reachable from s by a sequence $s_1, \ldots, s_n$ of states such that (i) $s_1 = s$, (ii) $s_n = s'$, and (iii) for every $j < n$, $s_j \sim_i s_{j+1}$ for some $i \in \Gamma$. Clearly, $\sim_\Gamma^C$ is an equivalence relation. Now, let $[s]_\Gamma = \{s' \in S \mid s' \sim_\Gamma s\}$ be the equivalence class of s according to $\sim_\Gamma$. The relation $\sim_\Gamma^C$ is extended to histories in a synchronous, pointwise way, *i.e.*, given $h, h' \in S^+$, $h \sim_\Gamma^C h'$ iff (i) $|h| = |h'|$ and (ii) for all $j \leq |h|$, $h_j \sim_\Gamma^C h'_j$. So, we introduce the notation $[h]_\Gamma = \{h' \in S^+ \mid h' \sim_\Gamma^C h\}$.

Definition 9 (Abstract CGS). *Given an iCGS* $M = \langle S, s_0, \{Act_i\}_{i\in Ag}, \{\sim_i\}_{i\in Ag}, d, \delta, V\rangle$ *and a coalition* $\Gamma \subseteq Ag$*, the* abstract (generalized) CGS $M_\Gamma = \langle S_\Gamma, [s_0]_\Gamma, \{Act_i\}_{i\in Ag}, d_\Gamma^{may}, d_\Gamma^{must}, \delta_\Gamma^{may}, \delta_\Gamma^{must}, V_\Gamma\rangle$ *is defined such that:*

1. *$S_\Gamma = \{[s]_\Gamma \mid s \in S\}$ is the set of equivalence classes for all states $s \in S$, with initial state $[s_0]_\Gamma$;*
2. *for every $t, t' \in S_\Gamma$ and joint action $\vec{\alpha}$, $t' \in \delta_\Gamma^{may}(t, \vec{\alpha})$ iff for some $s \in t$ and $s' \in t'$, $\delta(s, \vec{\alpha}) = s'$;*
3. *for every $t, t' \in S_\Gamma$ and joint action $\vec{\alpha}$, $t' \in \delta_\Gamma^{must}(t, \vec{\alpha})$ iff for all $s \in t$ there is $s' \in t'$ such that $\delta(s, \vec{\alpha}) = s'$;*
4. *for $x \in \{may, must\}$, $t \in S_\Gamma$, and $i \in Ag$, $d_\Gamma^x(i, t) = \{\alpha_i \in Act_i \mid \delta_\Gamma^x(t, (\alpha_i, \vec{\alpha}_{\bar{i}}))$ is defined for some tuple of actions $\vec{\alpha}_{\bar{i}}\}$;*
5. *for $v \in \{\mathrm{tt}, \mathrm{ff}\}$, $p \in AP$, and $t \in S_\Gamma$, $V_\Gamma(t, p) = v$ iff $V(s, p) = v$ for all $s \in t$; otherwise, $V_\Gamma(t, p) = \mathrm{uu}$.*

We now show that the abstraction of an iCGS is indeed a generalized CGS as defined in Def. 6. In particular, the indistinguishability relation for every $i \in Ag$ is assumed to be the identity relation.

Lemma 2. *For every coalition $\Gamma \subseteq Ag$, any abstraction M_Γ of an iCGS M is a generalized CGS.*

Proof. We consider items 1-4 in Def. 6.

1. S_Γ is a non-empty set of states, with initial state $[s_0]_\Gamma$. Each Act_i is a non-empty set of actions, and $\sim_i$ is assumed to be the identity relation for every agent $i \in Ag$.
2. For every $i \in Ag$ and $t \in S_\Gamma$, $d_\Gamma^{must}(i, t) \subseteq d_\Gamma^{may}(i, t) \subseteq Act_i$, as $\alpha \in d_\Gamma^{must}(i, t)$ iff by Def. 9.4, $\delta_\Gamma^{must}(t, (\alpha, \vec{\alpha}_{\bar{i}}))$ is defined for some tuple of actions $\vec{\alpha}_{\bar{i}}$. Then, $\delta_\Gamma^{may}(t, (\alpha, \vec{\alpha}_{\bar{i}}))$ is also defined (see below), and therefore $\alpha \in d_\Gamma^{may}(i, t)$.
3. For $x \in \{may, must\}$, if $\delta_\Gamma^x(t, \vec{\alpha})$ is defined, then for every $i \in Ag$, $\delta_\Gamma^x(t, (\alpha, \vec{\alpha}_{\bar{i}}))$ is defined for some $\vec{\alpha}_{\bar{i}}$, that is, $\alpha \in d_\Gamma^x(i, t)$.

   Further, for every $t \in S_\Gamma$ and $\vec{\alpha} \in ACT$, $\delta_\Gamma^{must}(t, \vec{\alpha}) \subseteq \delta_\Gamma^{may}(t, \vec{\alpha})$, as by Def. 9.2-3, if for all $s \in t$ there is $s' \in t'$ such that $\delta(s, \vec{\alpha}) = s'$, then in particular there exist $s \in t$, $s' \in t'$ such that $\delta(s, \vec{\alpha}) = s'$.
4. Clearly, $V_\Gamma : S_\Gamma \times AP \to \{\mathrm{tt}, \mathrm{ff}, \mathrm{uu}\}$ is a (three-valued) labelling function.

□

In the rest of the paper we will also consider the restrictions of abstract CGSs over *must* and *may* components only. We refer to them as M^x, for $x \in \{must, may\}$. Specifically, in M^{must} (resp. M^{may}) there only appears the protocol function d^{must} (resp. d^{may}) and the transition function δ^{must} (resp. δ^{may}). As such, M^{must} and M^{may} can be thought of as standard (two-valued) CGSs.

We can now state the main theoretical result in this section. First recall that a Γ-formula is a formula in which the strategic operator $\langle\langle\Gamma\rangle\rangle$ ranges only over some fixed coalition $\Gamma \subseteq Ag$. By next result, if a Γ-formula has a defined truth value in an abstract CGS M_Γ, built on an iCGS M, then the Γ-formula has the same truth value in M.

Theorem 2. *Given an iCGS M, state s, and coalition $\Gamma \subseteq Ag$, for every Γ-formula ϕ in ATL^*, we have that*

$$((M_\Gamma, [s]_\Gamma) \models^3 \phi) = \mathrm{tt} \quad \Rightarrow \quad (M, s) \models^2 \phi \tag{3}$$

$$((M_\Gamma, [s]_\Gamma) \models^3 \phi) = \mathrm{ff} \quad \Rightarrow \quad (M, s) \not\models^2 \phi \tag{4}$$

Proof. The proofs for (3) and (4) are by mutual induction on the structure of the formula (needed for the case of negation). We consider the cases in which the main operator is the strategic modality. The other cases are immediate and thus omitted.

(3) By Def. 8 $((M_\Gamma, [s]_\Gamma) \models^3 \langle\langle\Gamma\rangle\rangle\psi) = \mathrm{tt}$ iff for some joint strategy F_Γ^{must}, for all paths $p \in out([s]_\Gamma, F_\Gamma^{must})$, $((M_\Gamma, p) \models_I^3 \psi) = \mathrm{tt}$. Given F_Γ^{must} we construct a joint uniform strategy F'_Γ in M as follows: for every agent $i \in \Gamma$ and history $h \in S^+$, define $f'_i(h) = f_i^{must}(k)$, where (i) $|h| = |k|$ and (ii) for all $j \leq |h|$, $h_j \in k_j$. Notice that each f'_i so defined is uniform, as $h \sim_i h'$ implies $h \sim_\Gamma^C h'$ by definition of $\sim_\Gamma^C$, and therefore there is a unique $k \in S_\Gamma^+$ such that both h and h' belong to k, that is, $f'_i(h) = f_i^{must}(k) = f'_i(h')$. Given such F'_Γ, observe that if $p' \in out(s, F'_\Gamma)$, then $[p']_\Gamma = [p'_1]_\Gamma, [p'_2]_\Gamma \ldots$ belongs to $out([s]_\Gamma, F_\Gamma^{must})$. Since $((M_\Gamma, [p']_\Gamma) \models^3 \psi) = \mathrm{tt}$ by assumption, we obtain that $(M, p') \models^2 \psi$ by induction hypothesis, and therefore $(M, s) \models^2 \langle\langle\Gamma\rangle\rangle\psi$.

(4) By Def. 8 $((M_\Gamma, [s]_\Gamma) \models^3 \langle\langle\Gamma\rangle\rangle\psi) = \mathrm{ff}$ iff for every F_Γ^{may}, for some $p \in out([s]_\Gamma, F_\Gamma^{may})$, $((M_\Gamma, p) \models^3 \psi) = \mathrm{ff}$. Now every joint (uniform) strategy F'_Γ in M induces several joint strategies F_Γ^{may} in M_Γ: for every $k \in S_\Gamma^+$, $i \in Ag$, $f_i^{may}(k) = f'_i(h)$ for some $h \in S^+$ such that $|h| = |k|$ and $h_j \in k_j$ for every $j \leq |h|$. In particular, for every $p \in out([s]_\Gamma, F_\Gamma^{may})$ there exists $p' \in out(s, F'_\Gamma)$ such that $p'_j \in p_j$ for every $j \geq 0$, and therefore $(M, p') \not\models^2 \psi$ by induction hypothesis. As a result, $(M, s) \not\models^2 \langle\langle\Gamma\rangle\rangle\psi$. □

By Theorem 2 a defined answer to the model checking problem w.r.t. abstract, generalized CGS, which is decidable, can be transferred to the concrete, two-valued iCGS, whose model checking problem is undecidable in general. Obviously, if the returned value is undefined (uu), then no conclusive answer can be drawn.

In what follows we will provide a procedure to refine the abstraction in a conservative way. This refinement procedure assumes the existence of a "failure" state in which the truth value of the relevant formula is undefined. In Sec. 6 we describe the algorithm to find failure states, but first we prove some general results on automata for three-valued LTL in Sec. 5.

To conclude, we illustrate the abstraction procedure with our Train Gate Controller scenario in Example 1.

Example 2. *In Fig. 2 we show the abstract CGS obtained from the iCGS in Example 1 by considering formula $\varphi = \langle\langle\Gamma\rangle\rangle F(l_1 \wedge \neg bUg)$ for $\Gamma = \{t_1, c\}$. Specifically, abstraction M_Γ includes five abstract states according to the equivalence relation $\sim_{\{t_1,c\}}^C$. Notice that formula φ is undefined in M_Γ due to the undefined value of atom l_1 in state a_2.*

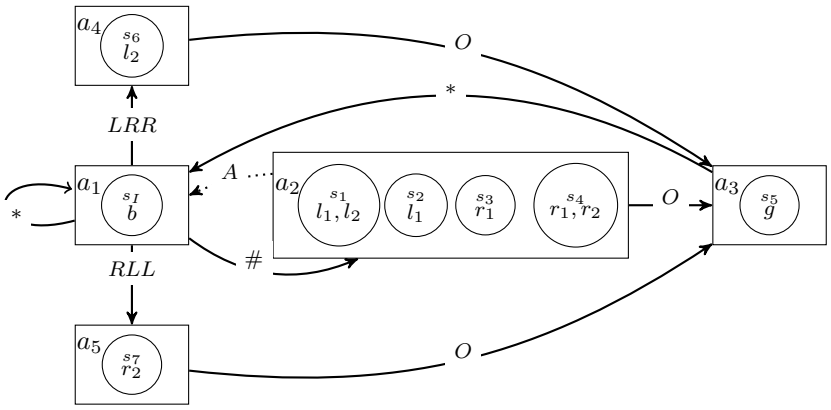

Figure 2: The abstract CGS for the iCGS in Example 1, where $\# \in \{LLL, LRL, RLR, RRR\}$, $*$ represents any action, $must$-transitions are depicted with continuous lines, and may-transitions are the continuous and dashed lines.

5. Automata for Three-valued LTL

In this section we introduce an automata-theoretic approach to the verification of the three-valued linear-time logic LTL. We refer to [6] for a detailed presentation of LTL; here we observe that the syntax of LTL can be obtained from Def. 2 by considering as state formulas only atoms (*i.e.*, $\varphi ::= a$). Then, the three-valued semantics for LTL follows from Def. 8 by considering only the clauses concerning the operators in the syntax of LTL. The results in this section will be used in Sec. 6 to find failure states.

We first recall the definition of generalized non-deterministic Büchi automaton.

Definition 10 (GNBA). *A* generalized non-deterministic Büchi automaton *is a tuple* $A = \langle Q, Q_0, \Sigma, \pi, \mathcal{F} \rangle$ *where*

- Q *is a finite set of* states *with* $Q_0 \subseteq Q$ *as the set of* initial states*;*
- Σ *is an* alphabet*;*
- $\pi : Q \times \Sigma \to 2^Q$ *is the (non-deterministic)* transition relation*;*
- $\mathcal{F}$ *is a (possibly empty) subset of* 2^Q*, whose elements are called* acceptance sets*.*

Given an infinite run $\rho = q_0 q_1 q_2 \ldots \in Q^\omega$, let *Inf*$(\rho)$ be the set of states q for which there are infinitely many indices i with $q = q_i$, that is, q appears infinitely often in ρ. Then, run ρ is *accepting* if for each acceptance set $F \in \mathcal{F}$, *Inf*$(\rho) \cap F \neq \emptyset$, that is, there are infinitely many indices i in ρ with $q_i \in F$. The *accepted language* $L(A)$ of automaton A consists of all infinite words $w \in \Sigma^\omega$ for which there exists at least one accepting run $\rho = q_0 q_1 q_2 \ldots \in Q^\omega$ such that for all $i \geq 0$, $q_{i+1} \in \pi(q_i, w_i)$.

We now show that for every LTL formula ψ, there exists an automaton $A_{\psi,\mathrm{uu}}$ that accepts exactly the infinite paths that evaluate ψ to undefined (uu). We first provide some definitions necessary for the construction.

Definition 11 (Closure and Elementarity). *The* closure $cl(\psi)$ *of an LTL formula ψ is the set consisting of all subformulas ϕ of ψ as well as their negation $\neg\phi$.*

A set $B \subseteq cl(\psi)$ is consistent *w.r.t. propositional logic iff for all $\psi_1 \wedge \psi_2, \neg\neg\phi \in cl(\psi)$: (i) $\psi_1 \wedge \psi_2 \in B$ iff $\psi_1 \in B$ and $\psi_2 \in B$; (ii) $\neg(\psi_1 \wedge \psi_2) \in B$ iff $\neg\psi_1 \in B$ or $\neg\psi_2 \in B$; (iii) if $\phi \in B$ then $\neg\phi \notin B$; (iv) $\neg\neg\phi \in B$ iff $\phi \in B$.*

Further, B is locally consistent *w.r.t. the until operator iff for all $\psi_1 U \psi_2 \in cl(\psi)$: (i) if $\psi_2 \in B$ then $\psi_1 U \psi_2 \in B$; (ii) if $\neg(\psi_1 U \psi_2) \in B$ then $\neg\psi_2 \in B$; (iii) if $\psi_1 U \psi_2 \in B$ and $\psi_2 \notin B$ then $\psi_1 \in B$; (iv) if $\neg\psi_1, \neg\psi_2 \in B$, then $\neg(\psi_1 U \psi_2) \in B$.*

Finally, B is elementary *iff it is both consistent and locally consistent.*

Notice that, differently from the standard construction for two-valued LTL [6], here we do not require elementary sets to be maximal (*i.e.*, either $\phi \in B$ or $\neg\phi \in B$), but we do require extra conditions (ii) and (iv) on consistency, and (ii) and (iv) on local consistency. Basically, these extra conditions reflect the satisfaction clauses for false (ff) in the three-valued semantics. E.g., as regards (ii), $(\psi_1 \wedge \psi_2)$ is false iff either ψ_1 or ψ_2 is false. While such conditions can be derived in the standard two-valued semantics by considering maximality, here they need to be assumed. Hereafter $Lit = AP \cup \{\neg a \mid a \in AP\}$ is the set of *litterals*.

Definition 12 (Automaton $A_{\psi,\mathrm{uu}}$). *Let ψ be a formula in LTL. We define the automaton $A_{\psi,\mathrm{uu}} = \langle Q, Q_0, 2^{Lit}, \pi, \mathcal{F}\rangle$ as follows:*

- *Q is the set of all elementary sets $B \subseteq cl(\psi)$ with $Q_0 = \{B \in Q \mid \psi \notin B$ and $\neg\psi \notin B\}$.*
- *The transition relation π is given by: let $A \subseteq Lit$. If $A \neq B \cap Lit$, then $\pi(B, A) = \emptyset$; otherwise $\pi(B, A)$ is the set of all elementary sets B' of formulas such that for every $X\phi, \psi_1 U \psi_2 \in cl(\psi)$: (i) $X\phi \in B$ iff $\phi \in B'$; (ii) $\neg X\phi \in B$ iff $\neg\phi \in B'$; (iii) $\psi_1 U \psi_2 \in B$ iff $\psi_2 \in B$ or, $\psi_1 \in B$ and $\psi_1 U \psi_2 \in B'$; (iv) $\neg(\psi_1 U \psi_2) \in B$ iff $\neg\psi_2 \in B$ and, $\neg\psi_1 \in B$ or $\neg(\psi_1 U \psi_2) \in B'$.*
- *$\mathcal{F} = \{F_{\psi_1 U \psi_2} \mid \psi_1 U \psi_2 \in cl(\psi)\} \cup \{Q\}$, where $F_{\psi_1 U \psi_2} = \{B \in Q \mid \psi_1 U \psi_2 \in B$ implies $\psi_2 \in B$ and $\neg\psi_2 \in B$ implies $\neg(\psi_1 U \psi_2) \in B\}$.*

By Def. 12 the transition relation works as follows: if the automaton reads a set A of literals that do not appear in the state, then the transition is not defined. Otherwise, the automaton checks the transitions enabled w.r.t. the semantics of the LTL operators. Notice that in Def. 12 we need to provide conditions on negated formulas as well, as elementary sets are not necessarily maximal. Furthermore, set $\mathcal{F}$ of accepting sets always includes set Q of all states. This guarantees that, whenever there are no until subformulas, any run in Q is accepting. This is convenient for the implementation described in Sec. 10. Finally, we can define automata $A_{\psi,\mathrm{tt}}$ and $A_{\psi,\mathrm{ff}}$ accepting exactly the infinite paths that evaluate ψ to true (resp. false), by setting $Q_0^{\mathrm{tt}} = \{B \in Q \mid \psi \in B\}$ and $Q_0^{\mathrm{ff}} = \{B \in Q \mid \neg\psi \in B\}$ respectively. For both automata we can prove results similar to Theorem 3 below.

To illustrate the automaton construction, we present two examples of automata for the next and the until operators.

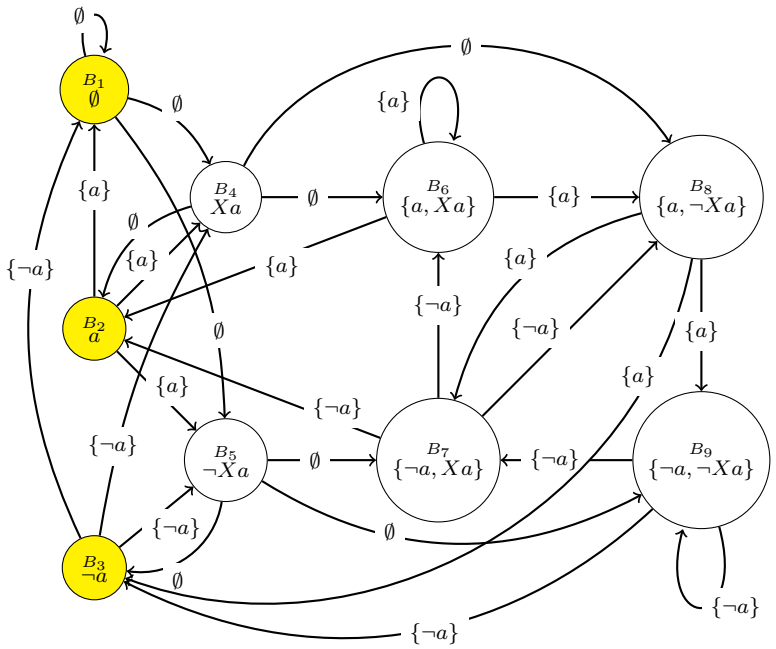

Figure 3: The automaton $A_{\psi,\mathrm{uu}}$ for formula $\psi = Xa$. Initial states are marked in yellow.

Example 3. *Consider $\psi = Xa$. The GNBA $A_{\psi,\mathrm{uu}}$ in Fig. 3 is obtained as indicated in Def. 12. Namely, the state space Q consists of all elementary sets of formulas contained in $cl(\psi) = \{a, \neg a, Xa, \neg Xa\}$: $B_1 = \emptyset$, $B_2 = \{a\}$, $B_3 = \{\neg a\}$, $B_4 = \{Xa\}$, $B_5 = \{\neg Xa\}$, $B_6 = \{a, Xa\}$, $B_7 = \{a, \neg Xa\}$, $B_8 = \{\neg a, Xa\}$, $B_9 = \{\neg a, \neg Xa\}$. The initial states of $A_{\psi,\mathrm{uu}}$ are the elementary sets $B \in Q$ with $\psi, \neg\psi \notin B$. That is, $Q_0 = \{B_1, B_2, B_3\}$. The transitions are depicted in Fig. 3. The set $\mathcal{F}$ is equal to $\{Q\}$ as ψ does not contain the until operator. Hence, every infinite run in $A_{\psi,\mathrm{uu}}$ is accepting.*

Example 4. *Consider the formula $\psi = aUb$. The GNBA $A_{\psi,\mathrm{uu}}$ in Fig. 4 is obtained as indicated in Def. 12. Namely, the state space Q of $A_{\psi,\mathrm{uu}}$ consists of all the elementary sets of formulas contained in $cl(\psi) = \{a, b, \neg a, \neg b, aUb, \neg(aUb)\}$: $B_1 = \emptyset$, $B_2 = \{a\}$, $B_3 = \{\neg a\}$, $B_4 = \{aUb, a\}$, $B_5 = \{aUb, b\}$, $B_6 = \{\neg(aUb), \neg b\}$, $B_7 = \{aUb, a, b\}$, $B_8 = \{aUb, \neg a, b\}$, $B_9 = \{aUb, a, \neg b\}$, $B_{10} = \{\neg(aUb), a, \neg b\}$, $B_{11} = \{\neg(aUb), \neg a, \neg b\}$. The initial states of $A_{\psi,\mathrm{uu}}$ are the elementary sets $B \in Q$ with $\psi, \neg\psi \notin B$. Thus, $Q_0 = \{B_1, B_2, B_3\}$. The transitions are depicted in Fig.4. Finally, $\mathcal{F} = \{F_\psi, Q\}$, where $F_\psi = \{B_1, B_2, B_3, B_5, B_6, B_7, B_8, B_{10}, B_{11}\}$.*

We now prove that the paths that evaluate ψ as undefined are exactly those included in the language of $A_{\psi,\mathrm{uu}}$. To prove this result, we make use of the following lemma.

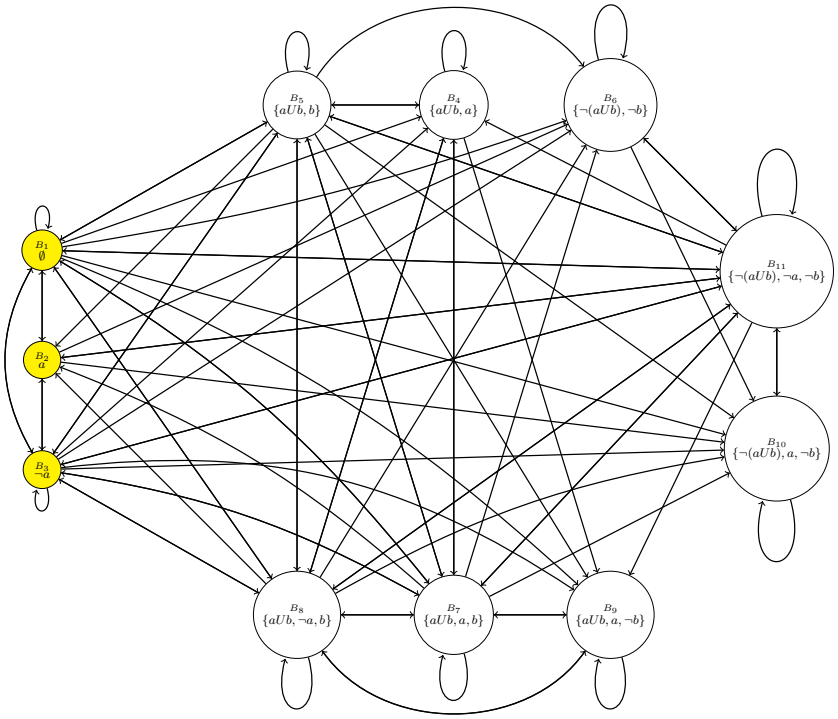

Figure 4: The automaton $A_{\psi,\mathrm{uu}}$ for formula $\psi = aUb$. Initial states are marked in yellow and the labelling of transitions is omitted for clarity.

Lemma 3. *Let run $\rho = B_1 B_2 \dots$ in $A_{\psi,\mathrm{uu}}$ and path $p = p_1 p_2 \dots$ in $(2^{Lit})^\omega$ be such that*

(i) for all $i \geq 0$, $B_{i+1} \in \pi(B_i, p_i)$;

(ii) ρ is accepting.

Then, for all $\phi \in cl(\psi)$,

(a) $\phi \in B_1$ iff $(p \models^3 \phi) = \mathrm{tt}$;

(b) $\neg\phi \in B_1$ iff $(p \models^3 \phi) = \mathrm{ff}$.

Proof. The proof is by induction on the structure of ϕ, where the induction hypothesis is that for all $i \geq 0$, $\phi \in B_i$ iff $(p_i p_{i+1} \dots \models^3 \phi) = \mathrm{tt}$ and $\neg\phi \in B_i$ iff $(p_i p_{i+1} \dots \models^3 \phi) = \mathrm{ff}$. Notice that by construction, $\pi(B_i, p_i)$ is defined iff $p_i = B_i \cap Lit$.

Base case: The statement for $\phi = a \in AP$ follows directly from the fact that $(p_i p_{i+1} \dots \models^3 \phi) = \mathrm{tt}$ iff $a \in p_i = B_i \cap Lit$, iff $a \in B_i$. Similarly, $(p_i p_{i+1} \dots \models^3 \phi) = \mathrm{ff}$ iff $\neg a \in p_i = B_i \cap Lit$, iff $\neg a \in B_i$.

Inductive steps: based on the induction hypothesis that the claim holds for formulas $\psi_1, \psi_2 \in cl(\psi)$, we need to prove that it also holds for $\phi = X\psi_1$, $\phi = \psi_1 \wedge \psi_2$, and $\phi = \psi_1 U \psi_2$ in $cl(\psi)$.

As regards $\phi = \psi_1 \wedge \psi_2$, $(p_i p_{i+1} \ldots \models^3 \phi) = \mathrm{tt}$ iff $(p_i, p_{i+1} \ldots \models^3 \psi_1) = \mathrm{tt}$ and $(p_i, p_{i+1} \ldots \models^3 \psi_1) = \mathrm{tt}$, iff $\psi_1, \psi_2 \in B_i$ by induction hypothesis, iff $\phi \in B_i$ by consistency. Similarly, $(p_i p_{i+1} \ldots \models^3 \phi) = \mathrm{ff}$ iff $(p_i, p_{i+1} \ldots \models^3 \psi_1) = \mathrm{ff}$ or $(p_i, p_{i+1} \ldots \models^3 \psi_2) = \mathrm{ff}$, iff $\neg\psi_1 \in B_i$ or $\neg\psi_2 \in B_i$ by induction hypothesis, iff $\neg\phi \in B_i$ by consistency.

As for $\phi = X\psi_1$, $(p_i p_{i+1} \ldots \models^3 \phi) = \mathrm{tt}$ iff $(p_{i+1}, p_{i+2} \ldots \models^3 \psi_1) = \mathrm{tt}$, iff $\psi_1 \in B_{i+1}$ by induction hypothesis, iff $\phi \in B_i$ by Def. 12. Similarly, $(p_i p_{i+1} \ldots \models^3 \phi) = \mathrm{ff}$ iff $(p_{i+1}, p_{i+2} \ldots \models^3 \psi_1) = \mathrm{ff}$, iff $\neg\psi_1 \in B_{i+1}$ by induction hypothesis, iff $\neg\phi \in B_i$ again by Def. 12.

For $\phi = \psi_1 U \psi_2$. Let $p = p_i p_{i+1} \ldots \in (2^{Lit})^\omega$ and $B_i B_{i+1} \ldots \in Q^\omega$ satisfy conditions (i) and (ii), we then show that $\phi \in B_i$ iff $(p_i p_{i+1} \ldots \models^3 \phi) = \mathrm{tt}$.

($\Leftarrow$) Suppose that $(p_i p_{i+1} \ldots \models^3 \psi_1 U \psi_2) = \mathrm{tt}$. Then, for some $j \geq i$, $(p_j p_{j+1} \ldots \models^3 \psi_2) = \mathrm{tt}$ and $(p_k p_{k+1} \ldots \models^3 \psi_1) = \mathrm{tt}$ for all $i \leq k < j$. By the induction hypothesis applied to ψ_1 and ψ_2, it follows that $\psi_2 \in B_j$ and $\psi_1 \in B_k$ for all $i \leq k < j$. Since B_j is elementary, $\psi_1 U \psi_2 \in B_j$ as well. Further, by definition of π, we obtain that $\psi_1 U \psi_2 \in B_k$ for all $i \leq k < j$. In particular, $\psi_1 U \psi_2 \in B_i$.

($\Rightarrow$) Suppose that $\psi_1 U \psi_2 \in B_i$. Since B_i is elementary, then either $\psi_1 \in B_i$ or $\psi_2 \in B_i$. If $\psi_2 \in B_i$, it follows from the induction hypothesis that $(p_i p_{i+1} \ldots \models^3 \psi_2) = \mathrm{tt}$, and therefore $(p_i p_{i+1} \ldots \models^3 \psi_1 U \psi_2) = \mathrm{tt}$. On the other hand, if $\psi_2 \notin B_i$, then both $\psi_1 \in B_i$ and $\psi_1 U \psi_2 \in B_i$. To obtain a contradiction suppose that $\psi_2 \notin B_j$ for all $j \geq i$. By the definition of π, by using an inductive argument we have that $\psi_1 \in B_j$ and $\psi_1 U \psi_2 \in B_j$ for all $j \geq i$. Further, since $B_i B_{i+1} \ldots$ is accepting and satisfies (ii), for infinitely many $k \geq i$, we have $B_k \in F_{\psi_1 U \psi_2}$, that is, $\psi_1 U \psi_2 \in B_k$ implies $\psi_2 \in B_k$. But for all $j \geq i$, $\psi_1 U \psi_2 \in B_j$ and $\psi_2 \notin B_j$ iff $B_j \notin F_{\psi_1 U \psi_2}$, which is a contradiction. Thus, $\psi_2 \in B_j$ for some $j > 0$. Assume that k is the smallest index such that $\psi_2 \in B_k$. By the induction hypothesis applied to ψ_1 and ψ_2 it follows $(p_k p_{k+1} \ldots \models^3 \psi_2) = \mathrm{tt}$ and $(p_j p_{j+1} \ldots \models^3 \psi_1) = \mathrm{tt}$ for all $i \leq j < k$. Hence $(p_i p_{i+1} \ldots \models^3 \psi_1 U \psi_2) = \mathrm{tt}$.

Now we show that $\neg\phi \in B_i$ iff $(p_i p_{i+1} \ldots \models^3 \phi) = \mathrm{ff}$.

($\Leftarrow$) Suppose that $(p_i p_{i+1} \ldots \models^3 \psi_1 U \psi_2) = \mathrm{ff}$. Then, either (i) for all $k \geq i$, $(p_k p_{k+1} \ldots \models^3 \psi_2) = \mathrm{ff}$, or (ii) for some $j \geq 0$, $(p_j p_{j+1} \ldots \models^3 \psi_1) = \mathrm{ff}$ and for all j, $i \leq j \leq k$ implies $(p_j p_{j+1} \ldots \models^3 \psi_2) = \mathrm{ff}$. If (i), then by induction hypothesis $\neg\psi_2 \in B_k$ for all $k \geq i$. By assumption (ii), there are infinitely many $j \geq i$ such that $\neg(\psi_1 U \psi_2) \in B_j$. By definition of π, $\neg(\psi_1 U \psi_2) \in B_k$ for all $k \geq i$. In particular, $\neg(\psi_1 U \psi_2) \in B_i$. If (ii), by induction hypothesis, for some $k \geq i$, $\neg\psi_1 \in B_k$, and for all j, $i \leq j \leq k$ implies $\neg\psi_2 \in B_j$. By local consistency we obtain that $\neg(\psi_1 U \psi_2) \in B_k$, and by definition of π, we derive that for all j, $i \leq j \leq k$ implies $\neg(\psi_1 U \psi_2) \in B_j$. In particular, $\neg(\psi_1 U \psi_2) \in B_i$.

($\Rightarrow$) Suppose that $\neg(\psi_1 U \psi_2) \in B_i$. We use the equivalent clause for falsehood of the until operator U and prove by induction that for every $k \geq i$, either (i) $\neg\psi_2 \in B_k$ or (ii) for some j, $i \leq j < k$ and $\neg\psi_1 \in B_j$. Consider $k = i$, then $\neg\psi_2 \in B_i$ by local consistency. Now consider $k \geq i$ such that $\neg\psi_2 \in B_k$ or for some j, $i \leq j < k$ and $\neg\psi_1 \in B_j$. If $\neg\psi_2 \in B_{k+1}$, then we are fine. On the other hand, suppose that

$\neg\psi_2 \notin B_{k+1}$. By definition of π, we have that either $\neg\psi_1 \in B_k$ or $\neg(\psi_1 U \psi_2) \in B_{k+1}$. In the former case, the result follows as for some j, $i \leq j < k+1$ and $\neg\psi_1 \in B_j$. In the latter, we have $\neg(\psi_1 U \psi_2) \in B_{k+1}$, and $\neg\psi_2 \in B_{k+1}$ by local consistency. Finally, by induction hypothesis, for all $k \geq 1$, either $(p_k p_{k+1} \ldots \models^3 \psi_2) = \mathrm{ff}$ or for some j, $i \leq j < k$ and $(p_j p_{j+1} \ldots \models^3 \psi_1) = \mathrm{ff}$, that is, $(p_i p_{i+1} \ldots \models^3 \psi_1 U \psi_2) = \mathrm{ff}$. $\square$

Finally, we prove the main theoretical result in this section. Hereafter, $Paths(\psi, \mathrm{uu})$ is the set of paths $p \in (2^{Lit})^\omega$ such that $(p \models^3 \psi) = \mathrm{uu}$.

Theorem 3. *For every LTL formula ψ there exists a GNBA $A_{\psi,\mathrm{uu}}$ (given as in Def. 12) s.t. $L(A_{\psi,\mathrm{uu}}) = Paths(\psi, \mathrm{uu})$. Moreover, the size of $A_{\psi,\mathrm{uu}}$ is exponential in the size of ψ.*

Proof. Clearly, by Def. 12 the size of $A_{\psi,\mathrm{uu}}$ in terms of number of states is exponential in the size of ψ. Then, we prove the set inclusions in both directions.

(1) Let $p = p_1 p_2 \ldots \in Paths(\psi, \mathrm{uu})$. For $i \geq 0$, define sets B_i of formulas as $\{\phi \in cl(\psi) \mid (p_i p_{i+1} \ldots \models^3 \phi) = \mathrm{tt}\} \cup \{\neg\phi \in cl(\psi) \mid (p_i p_{i+1} \ldots \models^3 \phi) = \mathrm{ff}\}$. Note that every B_i is elementary, *i.e.*, $B_i \in Q$. Now, we prove that $B_1 B_2 \ldots$ is an accepting run for p. Observe that $B_{i+1} \in \pi(B_i, p_i)$, since for all $i > 0$,

- $p_i = B_i \cap Lit$.
- For $X\phi \in cl(\psi)$, we have $X\phi \in B_i$ iff $(p_i p_{i+1} \ldots \models^3 X\phi) = \mathrm{tt}$, iff $(p_{i+1} p_{i+2} \ldots \models^3 \phi) = \mathrm{tt}$, iff $\phi \in B_{i+1}$.
- Similarly, $\neg X\phi \in B_i$ iff $(p_i p_{i+1} \ldots \models^3 X\phi) = \mathrm{ff}$ iff $(p_{i+1} p_{i+2} \ldots \models^3 \phi) = \mathrm{ff}$ iff $\neg\phi \in B_{i+1}$.
- For $\psi_1 U \psi_2 \in cl(\psi)$, we have $\psi_1 U \psi_2 \in B_i$ iff $(p_i p_{i+1} \ldots \models^3 \psi_1 U \psi_2) = \mathrm{tt}$ iff $(p_i p_{i+1} \ldots \models^3 \psi_2) = \mathrm{tt}$ or, $(p_i p_{i+1} \ldots \models^3 \psi_1) = \mathrm{tt}$ and $(p_{i+1} p_{i+2} \ldots \models^3 \psi_1 U \psi_2) = \mathrm{tt}$, iff $\psi_2 \in B_i$ or, $\psi_1 \in B_i$ and $\psi_1 U \psi_2 \in B_{i+1}$.
- Similarly, $\neg(\psi_1 U \psi_2) \in B_i$ iff $(p_i p_{i+1} \ldots \models^3 \psi_1 U \psi_2) = \mathrm{ff}$ iff $(p_i p_{i+1} \ldots \models^3 \psi_2) = \mathrm{ff}$, and $(p_i p_{i+1} \ldots \models^3 \psi_1) = \mathrm{ff}$ or $(p_{i+1} p_{i+2} \ldots \models^3 \psi_1 U \psi_2) = \mathrm{ff}$ iff $\neg\psi_2 \in B_i$, and $\neg\psi_1 \in B_i$ or $\neg(\psi_1 U \psi_2) \in B_{i+1}$.

The above shows that $B_1 B_2 \ldots$ is a run in $A_{\psi,\mathrm{uu}}$. Now, we prove that it is accepting, *i.e.*, for each subformula $\psi_{1,j} U \psi_{2,j} \in cl(\psi)$, $B_i \in F_j$ for infinitely many i, by contradiction. Consider there are finitely many i such that $B_i \in F_j$, then $B_i \notin F_j = F_{\psi_{1,j} U \psi_{2,j}}$ implies that either (i) $\psi_{1,j} U \psi_{2,j} \in B_i$ and $\psi_{2,j} \notin B_i$, or (ii) $\neg\psi_{2,j} \in B_i$ and $\neg\psi_{1,j} U \psi_{2,j} \notin B_i$. As regards (i), by construction of B_i, we have that $(p_i p_{i+1} \ldots \models^3 \psi_{1,j} U \psi_{2,j}) = \mathrm{tt}$ and $(p_i p_{i+1} \ldots \models^3 \psi_{2,j}) \neq \mathrm{tt}$. In particular, for some $k > i$ we have $(p_k p_{k+1} \ldots \models^3 \psi_{2,j}) = \mathrm{tt}$. By Lemma 3, it follows that $\psi_{2,j} \in B_k$, and by definition of F_j, $B_k \in F_j$. So, if $B_i \in F_j$ for finitely many i, then $B_k \in F_j$ for infinitely many k, which is a contradiction. As regards (ii), if there are finitely many i such that $B_i \in F_j$, then for some index m, for all $k \geq m$, we have $\neg\psi_{2,j} \in B_k$ and $\neg(\psi_{1,j} U \psi_{2,j}) \notin B_k$. By Lemma 3, we have that $(p_k p_{k+1} \ldots \models^3 \psi_{2,j}) = \mathrm{ff}$ and $(p_k p_{k+1} \ldots \models^3 \psi_{1,j} U \psi_{2,x}) \neq \mathrm{ff}$. But in particular, $(p_i p_{i+1} \ldots \models^3 \psi_{1,j} U \psi_{2,j}) = \mathrm{ff}$, which is a contradiction.

Algorithm 1 $FailureState(s, \varphi)$

Input: state s, state formula φ.
Output: failure state s_f, subformula of φ.

1: **if** $\varphi = a$ **then**
2: **return** (s, φ)
3: **if** $\varphi = \neg\varphi'$ **then**
4: **return** $FailureState(s, \varphi')$
5: **if** $\varphi = \varphi_1 \wedge \varphi_2$ **then**
6: **let** $i = min\{1, 2\}$ such that $((M, s) \models^3 \varphi_i) = \mathrm{uu}$ **return** $FailureState(s, \varphi_i)$
7: **if** $\varphi = \langle\langle\Gamma\rangle\rangle\psi$ **then**
8: **let** $\phi_1, \ldots, \phi_m$ by all maximal $\langle\langle\Gamma\rangle\rangle$-subformulas of ψ;
9: $AP := AP \cup \{atom_{\phi_1}, \ldots, atom_{\phi_m}\}$;
10: **for** $v \in \{\mathrm{tt}, \mathrm{ff}, \mathrm{uu}\}$, $V'(atom_{\phi_i}, v) := MC3(M, \phi_i, v)$;
11: $\psi' := \psi[\phi_1/atom_{\phi_1}, \ldots, \phi_m/atom_{\phi_m}]$;
12: **if** $Paths(M' \otimes A_{\psi',\mathrm{uu}}) = \emptyset$ **then**
13: **return** (s, φ)
14: **else**
15: **let** $w \in L(M' \otimes A_{\psi',\mathrm{uu}})$
16: **return** $FailurePath(w_{|_S}, \psi)$

(2) Let $p = p_1 p_2 \ldots \in L(A_{\psi,\mathrm{uu}})$, *i.e.*, there is an accepting run $B_1 B_2 \ldots$ for p in $A_{\psi,\mathrm{uu}}$. By the definition of $A_{\psi,\mathrm{uu}}$, we have that $\pi(B, A) = \emptyset$ for all pairs (B, A) with $A \neq B \cap Lit$. Then, it follows that $p_i = B_i \cap Lit$ for all $i \geq 0$. Thus, $p = (B_1 \cap Lit)(B_2 \cap Lit)\ldots$ and we need to prove that $((B_1 \cap Lit)(B_2 \cap Lit)\ldots \models^3 \psi) = \mathrm{uu}$. This follows by Lemma 3 and the fact that neither ψ nor $\neg\psi$ belong to B_1. □

This completes the proof of Theorem 3, which will be used in the next section.

6. Finding Failure States

In Sec. 4 we mentioned that the refinement procedure takes as input a "failure" state s_f in which some subformula of the specification to be checked is undefined. However, no hint was given as to how to find such state s_f. Hereafter we tackle this problem, but first we recall the notion of failure state from [7].

Definition 13 (Failure State). *A state s is a* failure state with respect to formula φ *iff* $((M, s) \models^3 \varphi) = \mathrm{uu}$ *and, either* $\varphi = a \in AP$, *or* $\varphi = \langle\langle\Gamma\rangle\rangle\psi$ *and either* $((M, p) \models^3 \psi) = \mathrm{tt}$ *for every may-path p starting from s, or* $((M, p) \models^3 \psi) = \mathrm{ff}$ *for every must-path p starting from s*

Intuitively, s is a failure state with respect to φ iff $((M, s) \models^3 \varphi) = \mathrm{uu}$ even though M has definite truth values for all (proper) subformulas of φ in the relevant states.

To introduce the procedure to find failure states, we first define the product between abstract CGS and GNBA.

Algorithm 2 $FailurePath(p, \psi)$

Input: path p, LTL-formula ψ
Output: call to $FailureState$ with state s appearing in p and state subformula φ of ψ

1: **if** $\psi = \varphi$ **then**
2: **return** $FailureState(p_1, \varphi)$
3: **if** $\psi = \neg\psi'$ **then**
4: **return** $FailurePath(p, \psi')$
5: **if** $\psi = \psi_1 \wedge \psi_2$ **then**
6: Let $i = min\{1, 2\}$ such that $((M, p) \models^3 \psi_i) = \text{uu}$ **return** $FailurePath(p, \psi_i)$
7: **if** $\psi = X\psi'$ **then**
8: **return** $FailurePath(p_{\geq 2}, \psi')$
9: **if** $\psi = \psi_1 U \psi_2$ **then**
10: $check_{\psi_1} := check_{\psi_2} := true$; $i := 0$
11: **while** $check_{\psi_1} = true \wedge check_{\psi_2} = true$ **do**
12: $i := i + 1$
13: **if** $((M, p_{\geq i}) \models^3 \psi_2) = \text{uu}$ **then**
14: $check_{\psi_2} := false$
15: **else if** $((M, p_{\geq i}) \models^3 \psi_1) = \text{uu}$ **then**
16: $check_{\psi_1} := false$
17: **if** $check_{\psi_2} = false$ **then**
18: **return** $FailurePath(p_{\geq i}, \psi_2)$
19: **else**
20: **return** $FailurePath(p_{\geq i}, \psi_1)$

Definition 14 (Product). *Given an abstract CGS* $M = \langle S, s_0, \{Act_i\}_{i\in Ag}, d^{may}, d^{must}, \delta^{may}, \delta^{must}, V\rangle$ *and a GNBA* $A = \langle Q, \Sigma, \pi, Q_0, \mathcal{F}\rangle$*, their product* $M \otimes A = \langle S \times Q, \widehat{S}_0, \{Act_i\}_{i\in Ag}, \widehat{d}^{may}, \widehat{d}^{must}, \widehat{\delta}^{may}, \widehat{\delta}^{must}, \widehat{V}\rangle$ *is the CGS s.t. for* $s, t \in S$, $q, q' \in Q$, $q_0 \in Q_0$, *and* $x \in \{may, must\}$.

- $\widehat{S}_0 = \{(s_0, q) \mid q \in \pi(q_0, \{p \mid V(s_0, p) = \text{tt}\} \cup \{\neg p \mid V(s_0, p) = \text{ff}\})\}$;
- $\widehat{d}^x((s, q)) = d^x(s)$;
- $\widehat{\delta}^x((s, q), \vec{\alpha}) = (s', q')$ *iff* $\delta^x(s, \vec{\alpha}) = s'$ *and* $q' \in \pi(q, \{p \mid V(s, p) = \text{tt}\} \cup \{\neg p \mid V(s, p) = \text{ff}\})$;
- $\widehat{V}(s, q) = q$.

Product $M \otimes A$ is so defined that its state space is the cartesian product of the states spaces of CGS M and automaton A. Then, the initial states of $M \otimes A$ are pairs in which the first element is the initial state s_0 of CGS M and the second is a state of the automata that is determined by the automaton transition from its initial state by reading the atoms true in the initial state s_0 of the CGS. The protocol function is defined by considering only the first component of state-pairs, whereas the transition function does in parallel a transition in the CGS to determine the first component and

a transition in the automaton for the second component. Finally, the labelling function is the identity on the second component (*i.e.*, the state of the automaton).

Given product $M \otimes A$, we define its language $L(M \otimes A)$ as the set of (infinite) words $w = (s_1, q_1), (s_2, q_2), \ldots$, such that (i) $(s_1, q_1) \in \widehat{S}_0$; (ii) for every $i \geq 1$, $(s_{i+1}, q_{i+1}) = \widehat{\delta}^x((s_i, q_i), \vec{\alpha})$ for some joint action $\vec{\alpha}$ and $x \in \{may, must\}$; and (iii) the restriction $w_{|_Q}$ to Q-components belongs to $L(A)$.

Lemma 4. *For every* $w \in L(M \otimes A_{\psi,\mathrm{uu}})$, $((M, w_{|_S}) \models^3 \psi) = \mathrm{uu}$.

Proof. By item (iii) above, the restriction $w_{|_Q}$ belongs to $L(A_{\psi,\mathrm{uu}})$, and by Theorem 3, $w_{|_Q} \in Paths(\psi, \mathrm{uu})$. Further, by items (i) and (ii), $w_{|_S}$ is a path in model M, and since $w_{|_S}$ and $w_{|_Q}$ agree pointwise on the labelling function, we obtain that $((M, w_{|_S}) \models^3 \psi) = \mathrm{uu}$. □

The procedure $FailureState()$ to find failure states and relevant subformulas is depicted in Algorithm 1, where a maximal $\langle\langle\Gamma\rangle\rangle$-subformula is a subformula of type $\langle\langle\Gamma\rangle\rangle\psi$ that does not appear within any other such subformula. It takes as input a state s and a formula φ such that $((M, s) \models^3 \varphi) = \mathrm{uu}$ and returns state s' and subformula φ' of φ. It calls procedure $FailurePath(p, \psi)$ in Algorithm 2 to deal with path subformulas. Note that, to work recursively on formulas of type $\varphi = \langle\langle\Gamma\rangle\rangle\psi$, we preprocess φ by substituting maximal $\langle\langle\Gamma\rangle\rangle$-subformulas of ψ with new atoms, whose truth value is determined by procedure $MC3()$, which is presented in Algorithm 7.

We prove that the procedure $FailureState()$ is sound.

Proposition 1. *Suppose that* $((M, s) \models^3 \varphi) = \mathrm{uu}$. *If* $FailureState(s, \varphi) = (s', \varphi')$ *then* s' *is a failure state w.r.t.* φ'.

Proof. We prove the soundness of $FailureState()$ by induction. Given a model M, state s, and formula φ with no nested strategy operators, the algorithm $FailureState(s, \varphi)$ starts by considering the base case in which φ is an atom (lines 1-2). Here, φ is a failure state since the atom a is undefined on it. For the inductive step, we have the following cases. For Boolean operators (lines 3-6), the procedure propagates over subformulas. To deal with the strategic operator (lines 7-16), the algorithm checks whether there is a path in the product $M \otimes A_{\psi,\mathrm{uu}}$ (Def. 14). By Lemma 4, the product between model M and automaton $A_{\psi,\mathrm{uu}}$ "accepts" all words w whose projection $w_{|_S}$ on the S-component is a path in M that makes the subformula ψ undefined. If there is no such word, then the procedure returns the current state and formula. Otherwise, procedure $FailurePath(p, \varphi)$ in Algorithm 2 is called, where p is a path obtained by projecting on the S-component some word accepted by the product $M \otimes A_{\psi,\mathrm{uu}}$, *i.e.*, $p = w_{|_S}$ for some $w \in L(M \otimes A_{\psi,\mathrm{uu}})$. In procedure $FailurePath(p, \varphi)$ the base case for state formulas (lines 1-2) returns to $FailureState()$ by taking as input the first state of path p. In lines 3-6, $FailurePath()$ handles the Boolean operators, and in lines 7-8 solves the next operator in accordance with its semantics. The main point of interest is the until operator U (lines 9-20). To prove that the **while** loop on line 11 terminates, we make use of Lemma 5 below, whereby we can show that the case of the until operator U in procedure $FailurePath()$ terminates after a finite number of step. □

Lemma 5. *Consider an abstract CGS M, path p, and formula $\varphi = \psi U \psi'$. If $((M,p) \models^3 \varphi) = \mathrm{uu}$ then for some $i \geq 0$, either $((M, p_{\geq i}) \models^3 \psi) = \mathrm{uu}$ or $((M, p_{\geq i}) \models^3 \psi') = \mathrm{uu}$.*

Proof. We prove the lemma by contradiction. Suppose that $((M,p) \models^3 \varphi) = \mathrm{uu}$ and for all $i \geq 0$, for some $v, v' \in \{\mathrm{tt}, \mathrm{ff}\}$, $((M, p_{\geq i}) \models^3 \psi) = v$ and $((M, p_{\geq i}) \models^3 \psi') = v'$. We then consider the following cases:

1. If $((M, p_{\geq i}) \models^3 \psi') = \mathrm{ff}$ (resp. tt) for all $i \geq 0$, then by the three-valued semantics we have $((M,p) \models^3 \varphi) = \mathrm{ff}$ (resp. tt), which is a contradiction.

2. If (1) is not the case, then formula ψ' is sometimes true and sometimes false, but always defined by assumption. Consider the smallest $i \geq 0$ such that $((M, p_{\geq i}) \models^3 \psi') = \mathrm{tt}$. Then, we can only have one of the following:

   (a) If for all $1 \leq j < i$, $((M, p_{\geq j}) \models^3 \psi) = \mathrm{tt}$, then by the three-valued semantics we have $((M,p) \models^3 \varphi) = \mathrm{tt}$, which is a contradiction.

   (b) Otherwise, there exists $1 \leq j < i$ such that $((M, p_{\geq j}) \models^3 \psi) = \mathrm{ff}$. Since we assumed that i is the smallest index for which ψ' is true, then for all $1 \leq k \leq j$ we have $((M, p_{\geq k}) \models^3 \psi') = \mathrm{ff}$. Hence, by the three-valued semantics it follows that $((M,p) \models^3 \varphi) = \mathrm{ff}$, which is again a contradiction. □

This concludes the proof of Proposition 1.

Example 5. *As an example of the application of our procedure to find failure states, we consider again the model proposed in Example 2. Now, as input for procedure $FailureState()$ we consider formula $\varphi = \langle\langle\Gamma\rangle\rangle F(l_1 \wedge \neg bUg)$ and state a_1 in Fig. 2. In Example 2 we observed that $((M_\Gamma, a_1) \models^3 \varphi) = \mathrm{uu}$. Since the main operator in φ is the strategic modality $\langle\langle\Gamma\rangle\rangle$, procedure $FailureState()$ goes to line 7. In particular, it constructs the automaton $A_{\psi,\mathrm{uu}}$ that accepts all paths where $\psi = true\, U(l_1 \wedge \neg bUg)$ is undefined. Now, the language of the product of model M_Γ and $A_{\psi,\mathrm{uu}}$ is not empty, consider for instance word $w = (a_1,q_1)(a_2,q_2)(a_3,q_3)\ldots$, where q_1, q_2, q_3 are states in $A_{\psi,\mathrm{uu}}$. Hence, the procedure calls $FailurePath()$ with input $p = a_1a_2a_3\ldots$, i.e., the projection on the M_Γ-component of word w. Since formula ψ has until U as the main operator, the procedure goes to line 9. Now, $((M_\Gamma, p_{\geq 2}) \models^3 \psi) = \mathrm{uu}$ and we call $FailurePath(p_{\geq 2}, \psi')$ with $\psi' = l_1 \wedge \neg bUg$. Observe that the main operator in ψ' is a conjunction $\wedge$, and therefore we go to line 5. Here, $((M_\Gamma, p_{\geq 2}) \models^3 l_1) = ((M_\Gamma, p_2) \models^3 l_1) = \mathrm{uu}$, then we call $FailurePath(p_{\geq 2}, l_1)$, and by lines 1-2, $FailureState(p_2, l_1)$ finally returns failure state a_2 and atom l_1.*

7. Refinement

By Theorem 2 if a formula is undefined on abstraction M_Γ, then no conclusion can be drawn on the model checking problem for M. In this section we provide a refinement procedure taking as input the "failure" state s_f in M_Γ given by Algorithm 1, and returning a refined CGS M^r_Γ, whose state space is bigger than M_Γ, but still

smaller than M in general, and for which we are able to prove Theorem 4 below, a preservation result similar to Theorem 2.

The procedure $\mathit{Refinement}(M_\Gamma, M, s_f)$ is described in Algorithm 3. Intuitively, we look at incoming transitions into s_f. For concrete states s and s' in s_f, if the Γ-component of actions ending respectively in s and s' are different, any uniform strategy for Γ will visit either s or s'. As a result, the abstract state s_f can be split "safely" into an s- and an s'-component. More precisely, the procedure $\mathit{Refinement}()$ begins by initializing as true the values of a matrix m that stores the relation "having uniform incoming transitions", between the concrete states in s_f (lines 1-2). Then, the algorithm calls the subroutine $\mathit{Check}_1(M_\Gamma, M, s_f, m)$ in Algorithm 4, which updates the values in m by considering the concrete transition function δ in M. In particular, at each iteration $\mathit{Check}_1()$ considers one predecessor t_f of s_f (line 1). Then, two other loops iterate on pairs of states s and s' in the abstract state s_f and pairs of states t and t' in predecessor t_f. If s and s' are indistinguishable for some agent $i \in \Gamma$ and i performs the same action in the transitions from t and t' to s and s' respectively (lines 2-4), then we update the value of the corresponding cell in m to false (line 5). The subroutine reported in $\mathit{Check}_1()$ carries out the first round of updates on m. Further updates in the $\mathit{Refinement}()$ algorithm are performed by the subroutine $\mathit{Check}_2(M_\Gamma, s_f, m, \mathit{update})$ reported in Algorithm 5, which considers the "indirect" binding that some concrete states may have in an abstract state. Specifically, given the states s and s' in the abstract state s_f that have true as value in m (lines 2-3), we need to consider the relation that s and s' have with the other states in s_f (lines 4-6): if the values in m for both states related with some other state t are false, then we update the value of cell $m[s, s']$ to false as well. Subroutine $\mathit{Check}_2()$ is called repeatedly in algorithm $\mathit{Refinement}()$ as long as guard update remains true. When update becomes false, we proceed to check whether there is at least an element true in m (line 8). If this is the case, we assign the related concrete states s and s' to two different, new abstract states v and w (line 10). Finally, we populate the new abstract states v and w with the other concrete states in the old abstract state s_f (which is removed) according to matrix m (lines 11-15).

Hereafter we present the formal definition of the refined CGS M^r_Γ as obtained by the application of the $\mathit{Refinement}()$ algorithm.

Definition 15 (Refined CGS). *Given an abstract CGS* $M_\Gamma = \langle S_\Gamma, s_0, \{\mathit{Act}_i\}_{i\in Ag}, d^{may}_\Gamma, d^{must}_\Gamma, \delta^{may}_\Gamma, \delta^{must}_\Gamma, V_\Gamma\rangle$*, its refinement* $M^r_\Gamma = \langle S^r_\Gamma, s^r_0, \{\mathit{Act}_i\}_{i\in Ag}, d^{may}_\Gamma, d^{must}_\Gamma, \delta^{may}_\Gamma, \delta^{must}_\Gamma, V^r_\Gamma\rangle$ *as obtained by an application of algorithm* $\mathit{Refinement}(M_\Gamma, M, s_f)$ *is defined as follows:*

1. *If* $\mathit{Refinement}(M_\Gamma, M, s_f)$ *returns* $\mathit{split} = \mathrm{tt}$*, then* $S^r_\Gamma = (S_\Gamma \setminus \{s_f\}) \cup \{v, w\}$*, that is,* S^r_Γ *is the set* S_Γ *of states in* M_Γ *without the "failure" state* s_f*, but with the new states* v, w *added by Alg. 3. Otherwise (if* $\mathit{split} = \mathrm{ff}$*), then* $S^r_\Gamma = S_\Gamma$*. Moreover,* s^r_0 *is the state in* S^r_Γ *such that* $s_0 \in s^r_0$*, for* $s_0 \in M$*.*

2. *For* $x \in \{may, must\}$*, the transitions relations* δ^x_Γ *and the protocol functions* d^x_Γ *are defined as in Def. 9. In particular,*

   (a) *for every* $t, t' \in S^r_\Gamma$ *and joint action* $\vec{\alpha}$*,* $t' \in \delta^{may}_\Gamma(t, \vec{\alpha})$ *iff for some* $s \in t$ *and* $s' \in t'$*,* $\delta(s, \vec{\alpha}) = s'$*;*

Algorithm 3 $Refinement(M_\Gamma, M, s_f)$

Input: CGS M_Γ, iCGS M, failure state s_f
Output: CGS M_Γ, truth value $split$

```
1: for s, s' ∈ s_f do
2:    m[s, s'] := true
3: Check_1(M_Γ, M, s_f, m); update := true
4: while update = true do
5:    Check_2(M_Γ, s_f, m, update)
6: split := false
7: while s, s' ∈ s_f ∧ split = false do
8:    if m[s, s'] = true then
9:       split := true; remove(s_f, S_Γ)
10:      add(v, S_Γ); add(w, S_Γ); add(s, v); add(s', w)
11:      for t ∈ s_f do
12:         if m[s, t] = true then
13:            add(t, w)
14:         else
15:            add(t, v)
16: return (M_Γ, split)
```

Algorithm 4 $Check_1(M_\Gamma, M, s_f, m)$

Input: CGS M_Γ, iCGS M, failure state s_f, matrix m
Output: updated matrix m

```
1: for t_f ∈ Pre(s_f), s, s' ∈ s_f, t, t' ∈ t_f do
2:    if δ(t, α⃗) = s ∧ δ(t', β⃗) = s' then
3:       for i ∈ Γ do
4:          if s ∼_i s' ∧ α⃗_i = β⃗_i then
5:             m[s, s'] := false
```

(b) for every $t, t' \in S^r_\Gamma$ and joint action $\vec{\alpha}$, $t' \in \delta^{must}_\Gamma(t, \vec{\alpha})$ iff for all $s \in t$ there is $s' \in t'$ such that $\delta(s, \vec{\alpha}) = s'$;

(c) for every $t \in S^r_\Gamma$, and $i \in Ag$, $d^x_\Gamma(i, t) = \{\alpha_i \in Act_i \mid \delta^x_\Gamma(t, (\alpha_i, \vec{\alpha}_{\bar{i}}))$ is defined for some $\vec{\alpha}_{\bar{i}}\}$.

3. *For $v \in \{\mathrm{tt}, \mathrm{ff}\}$, $p \in AP$, and $t \in S^r_\Gamma$, $V^r_\Gamma(t, p) = v$ iff $V(s, p) = v$ for all $s \in t$; otherwise, $V^r_\Gamma(s, p) = \mathrm{uu}$.*

By Def. 15 the components of the refined CGS M^r_Γ coincide with those in abstraction M_Γ, except possibly as regards the "failure" state s_f and new states introduced by $Refinement()$. On the new states, the transition relations and protocol functions are defined in analogy with M_Γ.

We now show a property of the refined CGS M^r_Γ, which will be useful to prove the main preservation result Theorem 4. Intuitively, must strategies in M^r_Γ respect uniformity *on the set of their outcomes*.

Algorithm 5 $Check_2(M_\Gamma, s_f, m, update)$

Input: CGS M_Γ, failure state s_f, matrix m, truth value $update$
Output: updated matrix m, truth value $update$

1: $update := false$
2: **for** $s, s' \in s_f$ **do**
3: **if** $m[s, s'] = true$ **then**
4: **for** $t \in s_f$ **do**
5: **if** $m[s, t] = false \wedge m[s', t] = false$ **then**
6: $m[s, s'] := false$; $update := true$

Lemma 6. *In M^r_Γ for every joint strategy F^{must}_Γ, for all $p, \hat{p} \in out(t, F^{must}_\Gamma)$, all $p' \in p$, $\hat{p}' \in \hat{p}$, and all $i \in \Gamma$, $j \in \mathbb{N}$, if $p'_{\leq j} \sim_i \hat{p}'_{\leq j}$ then $f^{must}_i(p_{\leq j}) = f^{must}_i(\hat{p}_{\leq j})$.*

Proof. For illustration, the lemma holds trivially in the abstraction M_Γ as, if $p'_{\leq j} \sim_i \hat{p}'_{\leq j}$ for some $i \in \Gamma$, $j \in \mathbb{N}$, then $[p'_{\leq j}]_\Gamma = [\hat{p}'_{\leq j}]_\Gamma$ and immediately $f^{must}_i([p'_{\leq j}]_\Gamma) = f^{must}_i([\hat{p}'_{\leq j}]_\Gamma)$. So, it is left to show that the refinement procedure does not break this feature. To obtain a contradiction, suppose that $f^{must}_i(p_{\leq j}) \neq f^{must}_i(\hat{p}_{\leq j})$. This means that for some minimal $k \leq j$, p_k and $\hat{p}_k$ have been obtained by splitting some "failure" state s_f. But then, by the structure of the $Refinement()$ procedure, p and $\hat{p}$ cannot be both paths in $out(t, F^{must}_\Gamma)$, as there would be prefixes $p_{\leq k-1} = \hat{p}_{\leq k-1}$ and a unique incoming action $f^{must}_i(p_{\leq k-1}) = f^{must}_i(\hat{p}_{\leq k-1})$ for both states p_k and $\hat{p}_k$, which contradicts the fact that they have been obtained through a split. □

By Lemma 6 we can prove the main preservation result of this section. In particular, the lemma is used in the inductive step for strategy operators.

Theorem 4. *Given an iCGS M, state s, coalition Γ, its abstract CGS M_Γ with refinement M^r_Γ, and state $s^r_\Gamma \ni s$, for every Γ-formula ϕ in ATL^*,*

$$((M^r_\Gamma, s^r_\Gamma) \models^3 \phi) = \mathrm{tt} \quad \Rightarrow \quad (M, s) \models^2 \phi \tag{5}$$
$$((M^r_\Gamma, s^r_\Gamma) \models^3 \phi) = \mathrm{ff} \quad \Rightarrow \quad (M, s) \not\models^2 \phi \tag{6}$$

Proof. The proofs for both (5) and (6) are by mutual induction on the structure of the formula. The induction hypotheses are as follows:

$$((M^r_\Gamma, t^r) \models^3 \varphi) = \mathrm{tt} \quad \Rightarrow \quad (M, t) \models^2 \varphi \tag{7}$$
$$((M^r_\Gamma, t^r) \models^3 \varphi) = \mathrm{ff} \quad \Rightarrow \quad (M, t) \not\models^2 \varphi \tag{8}$$

for $t \in t^r$ and state formula φ, and

$$((M^r_\Gamma, p^r) \models^3 \psi) = \mathrm{tt} \quad \Rightarrow \quad (M, p) \models^2 \psi \tag{9}$$
$$((M^r_\Gamma, p^r) \models^3 \psi) = \mathrm{ff} \quad \Rightarrow \quad (M, p) \not\models^2 \psi \tag{10}$$

for $p_j \in p^r_j$ for every $j \geq 0$, and path formula ψ. We consider the case where the main operator is the strategic modality. The other cases are immediate and thus omitted.

(5) By Def. 4 $((M^r_\Gamma, s^r_\Gamma) \models \langle\langle\Gamma\rangle\rangle\psi) = \mathrm{tt}$ iff for some joint strategy F^{must}_Γ, for all paths $p \in out(s^r_\Gamma, F^{must}_\Gamma)$, $((M^r_\Gamma, p) \models \psi) = \mathrm{tt}$. Given F^{must}_Γ and $out(s^r_\Gamma, F^{must}_\Gamma)$ we construct a joint uniform strategy F'_Γ to be used in M as follows: for every agent $i \in \Gamma$ and history $h \in S^+$, we define $f'_i(h) = \vec{\alpha}_i$ if there is a path $p \in out(s^r_\Gamma, F^{must}_\Gamma)$ such that (i) for all $j \leq |h|$, $h_j \in p_j$, (ii) $f^{must}_i(p_{\leq |h|}) = \vec{\alpha}_i$, and (iii) $\delta^{must}_\Gamma(p_{|h|}, \vec{\alpha}) = p_{|h|+1}$. For all histories that do not satisfy the conditions above the only important thing is to preserve uniformity. To do so, for each h where f'_i is undefined, if there exists h' on which f'_i is defined and h and h' are indistinguishable to i, then $f'_i(h) = f'_i(h')$. Otherwise, we can set any action in accordance with the transition function. We observe that by Lemma 6, such F'_Γ is well-defined, as F^{must}_Γ is "uniform" on $out(s^r_\Gamma, F^{must}_\Gamma)$. Given such F'_Γ, we obtain that for all $p \in out(s, F'_\Gamma)$ there is $p' \in out(s^r_\Gamma, F^{must}_\Gamma)$ such that $p_j \in p'_j$, for all $j \geq 0$. Then, by induction hypothesis, for all $p \in out(s, F'_\Gamma)$, $(M, p) \models^2 \psi$, and therefore $(M, s) \models^2 \langle\langle\Gamma\rangle\rangle\psi$.

(6) By Def. 4 $((M^r_\Gamma, s^r_\Gamma) \models \langle\langle\Gamma\rangle\rangle\psi) = \mathrm{ff}$ iff for every joint strategy F^{may}_Γ, for some path $p \in out(s^r_\Gamma, F^{may}_\Gamma)$, $((M^r_\Gamma, p) \models \psi) = \mathrm{ff}$. Now, every joint (uniform) strategy F'_Γ in M induces several joint may-strategies F^{may}_Γ in M^r_Γ depending on the witness of choice. Namely, given a strategy f'_i for agent $i \in Ag$, we define a may-strategy f^{may}_i such that for every history $k \in (S^r_\Gamma)^+$, $f^{may}_i(k) = f'_i(h)$ for some history $h \in S^+$ such that $|h| = |k|$ and $h_j \in k_j$ for every $j \leq |h|$. Given such strategies, for every $p \in out(s^r_\Gamma, F^{may}_\Gamma)$ there exists $p' \in out(s, F'_\Gamma)$ such that $p'_j \in p_j$ for every $j \geq 0$, and therefore $(M, p') \not\models^2 \psi$ by induction hypothesis. As a result, $(M, s) \not\models^2 \langle\langle\Gamma\rangle\rangle\psi$. □

By Theorem 4 defined truth values are preserved from the refined CGS to the original iCGS, similarly to Theorem 2.

Example 6. *As an example of the application of our refinement procedure, we can consider the model proposed in Example 2 and a_2 as failure state (as produced in Example 5). The $Refinement()$ algorithm initializes the matrix m used to split the abstract state a_2 with all the entries as true (lines 1-2). Then, $Check_1()$ starts by considering the predecessors of a_2, i.e., a_1 only. Since a_1 is composed of only one concrete state s_I, $Check_1()$ considers the transitions from s_I to all the states in a_2. In detail, by considering the model in Example 1, we need to check four transitions: (s_I, s_1), (s_I, s_2), (s_I, s_3), and (s_I, s_4). For the last one, we need to consider the actions pertaining to coalition Γ appearing in formula $\varphi = \langle\langle t_1, c\rangle\rangle F(l_1 \wedge \neg bUg)$, i.e., $\Gamma = \{t_1, c\}$. $Check_1()$ modifies the matrix m by updating to $false$ the elements $m[s_1, s_2]$ and $m[s_3, s_4]$ since some agents in coalition Γ performs the same actions. With more detail, $m[s_1, s_2]$ becomes $false$ due to the action L for train t_1 and $m[s_3, s_4]$ becomes $false$ due to the action R for controller c. After this step, $Check_2()$ checks whether there is a partition according to matrix m. In particular, in our example such a partition exists, whereas the condition in line 5 of $Check_2()$ never holds. Since there are no further updates in matrix m, the algorithm exits the loop in lines 4-6 of $Refinement()$. Finally, because of the loop in lines 7-15, state a_2 is split into two new states a^1_2, including concrete states s_1 and s_2, and a^2_2 with concrete states s_3 and s_4 (Figure 5).*

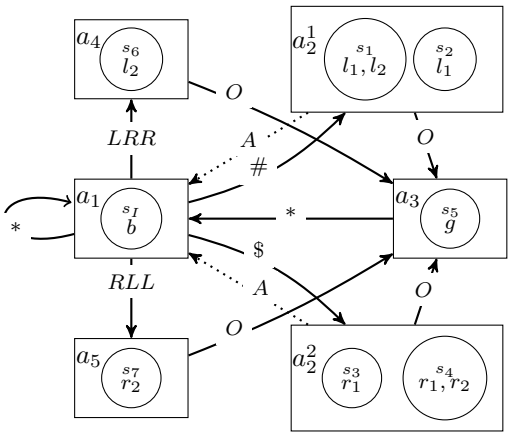

Figure 5: The refinement for the CGS in Example 2, where $\# \in \{LLL, LRL\}$, $\$ \in \{RLR, RRR\}$, $*$ is any action, $must$-transitions are depicted with continuous lines, and may-transitions are the continuous and dashed lines.

8. Translating from Three-valued to Two-valued Semantics

To decide the model checking problem in the three-valued, perfect information semantics, we make use of some auxiliary procedures to update the model and the formula, so as to handle the three-valued semantics in the two-valued model checker MCMAS. The results presented in this section are inspired to the procedures presented in [11], the main difference here is that we need to handle models with *must* and *may* transitions. Given a model M we use the procedure $Duplicate\_atoms(M)$ to produce a new model M' that differs from M as regards atoms and the labeling function as follows:

1. For each atom $a \in AP$, the procedure introduces two new atoms a_{tt} and a_{ff} and add them to the new set of atoms $AP' = \{a_{\mathrm{tt}}, a_{\mathrm{ff}} \mid a \in AP\}$.

2. For each state $s \in S$, the procedure defines the (two-valued) labeling function V' on s as $V'(s, a_{\mathrm{tt}}) = \mathrm{tt}$ iff $V(s, a) = \mathrm{tt}$, and $V'(s, a_{\mathrm{ff}}) = \mathrm{tt}$ iff $V(s, a) = \mathrm{ff}$.

As regards the formula to check, we introduce Algorithm 6 that, given an LTL-formula φ on AP and a truth value $v \in \{\mathrm{tt}, \mathrm{ff}\}$, returns a new formula $Transl(\varphi, v)$ on AP', which handles the new atoms generated by $Duplicate\_atoms()$. In particular, Algorithm 6 restricts a similar translation in [11] to LTL formulas only. To this end, we show the following result.

Lemma 7. *Given an iCGS M and LTL formula φ, let $M' = Duplicate\_atoms(M)$, $\varphi_{\mathrm{tt}} = Transl(\varphi, \mathrm{tt})$, and $\varphi_{\mathrm{ff}} = Transl(\varphi, \mathrm{ff})$. Then, for every path p,*

$$(M', p) \models^2 \varphi_{\mathrm{tt}} \quad \Leftrightarrow \quad ((M, p) \models^3 \varphi) = \mathrm{tt} \tag{11}$$

$$(M', p) \models^2 \varphi_{\mathrm{ff}} \quad \Leftrightarrow \quad ((M, p) \models^3 \varphi) = \mathrm{ff} \tag{12}$$

$$(M', p) \models^2 \neg(\varphi_{\mathrm{tt}} \vee \varphi_{\mathrm{ff}}) \quad \Leftrightarrow \quad ((M, p) \models^3 \varphi) = \mathrm{uu} \tag{13}$$

Algorithm 6 $Transl(\varphi, v)$

Input: LTL-formula φ, truth value $v \in \{\mathrm{tt}, \mathrm{ff}\}$
Output: LTL-formula $Transl(\varphi, v)$

switch (φ)
case a**:**
  if $v = \mathrm{tt}$ **then**
    return a_{tt}
  else if $v = \mathrm{ff}$ **then**
    return a_{ff}
case $\neg\psi$**:**
  return $Transl(\psi, \neg v)$ /* where $\neg\mathrm{tt} = \mathrm{ff}$ and $\neg\mathrm{ff} = \mathrm{tt}$ */
case $\psi \wedge \psi'$**:**
  if $v = \mathrm{tt}$ **then**
    return $Transl(\psi, \mathrm{tt}) \wedge Transl(\psi', \mathrm{tt})$
  else if $v = \mathrm{ff}$ **then**
    return $Transl(\psi, \mathrm{ff}) \vee Transl(\psi', \mathrm{ff})$
case $X\psi$**:**
  return $X\ Transl(\psi, v)$
case $\psi U \psi'$**:**
  if $v = \mathrm{tt}$ **then**
    return $Transl(\psi, \mathrm{tt})\ U\ Transl(\psi', \mathrm{tt})$
  else if $v = \mathrm{ff}$ **then**
    return $Transl(\psi, \mathrm{ff})\ R\ Transl(\psi', \mathrm{ff})$
case $\psi R \psi'$**:**
  if $v = \mathrm{tt}$ **then**
    return $Transl(\psi, \mathrm{tt})\ R\ Transl(\psi', \mathrm{tt})$
  else if $v = \mathrm{ff}$ **then**
    return $Transl(\psi, \mathrm{ff})\ U\ Transl(\psi', \mathrm{ff})$

Lemma 7 can be proved as a slightly adaptation of the proof in [11].

We can now present Algorithm 7 to decide the model checking problem in the three-valued semantics (with perfect information), and analyse its complexity.

Algorithm 7 takes as input a (three-valued) model M, an ATL^*-formula φ, and $x \in \{may, must\}$. Then, it returns whether φ is true or false according to the three-value semantics for ATL^*.

Algorithm 7 $MC3(M, \varphi, v)$

Input: iCGS M, a state formula φ in ATL^*, truth value $v \in \{\mathrm{tt}, \mathrm{ff}\}$
Output: Set of states $s \in S$ such that $((M, s) \models^3 \varphi) = v$

switch (φ)
case $\varphi = a$**:**
  return $\{s \in S \mid V(s, a) = v\}$
case $\varphi = \neg\varphi'$**:**
  return $MC3(M, \varphi', \neg v)$
case $\varphi = \varphi' \wedge \varphi''$**:**
  if $v = \mathrm{tt}$ **then**
    return $MC3(M, \varphi', \mathrm{tt}) \cap MC3(M, \varphi'', \mathrm{tt})$;
  else if $v = \mathrm{ff}$ **then**
    return $MC3(M, \varphi', \mathrm{ff}) \cup MC3(M, \varphi'', \mathrm{ff})$;
case $\varphi = \langle\langle\Gamma\rangle\rangle\psi$**:**
  let $\phi_1, \ldots, \phi_m$ by all maximal $\langle\langle\Gamma\rangle\rangle$-subformulas of ψ;
  $AP := AP \cup \{atom_{\phi_1}, \ldots, atom_{\phi_m}\}$;
  for $i \leq m, v \in \{\mathrm{tt}, \mathrm{ff}\}$, $V(atom_{\phi_i}, v) := MC3(M, \phi_i, v)$;
  $\psi := \psi[\phi_1/atom_{\phi_1}, \ldots, \phi_m/atom_{\phi_m}]$;
  if $v = \mathrm{tt}$ **then**
    return $MC2(M^{must}, \langle\langle\Gamma\rangle\rangle Transl(\psi, \mathrm{tt}))$
  else if $v = \mathrm{ff}$ **then**
    return $MC2(M^{may}, [\![\Gamma]\!] Transl(\psi, \mathrm{ff}))$
end switch

Theorem 5. *Algorithm 7 is sound. That is,* $s \in MC3(M, \varphi, v)$ *iff* $((M, s) \models^3 \varphi) = v$. *Moreover, it is in 2EXPTIME for* ATL^**, and in PTIME for* ATL.

Proof. Algorithm 7 works bottom-up on the structure of the state formula φ. The cases for the atomic propositions and Boolean operators are immediate. The inductive case for formulas of type $\langle\langle\Gamma\rangle\rangle\psi$ calls the model checking procedure $MC2(M, \phi)$, which returns the set of states satisfying the ATL^*-formula ϕ in model M, according to the two-valued semantics [3]. This step, which can be performed in 2EXPTIME, requires more discussion. First of all, we assume that ψ is an LTL formula. This can be assumed w.l.o.g. by taking care of updating the model and formula at each iteration of Algorithm 7. In particular, once we have taken care of the innermost formulas of type $\langle\langle\Gamma\rangle\rangle\psi$, we extend the set AP of atoms with a new atom $atom_{\langle\langle\Gamma\rangle\rangle\psi}$, whose interpretation is provided by the model checking procedure, that is, $V(atom_{\langle\langle\Gamma\rangle\rangle\psi}, s) = v$ iff $s \in MC3(M, \langle\langle\Gamma\rangle\rangle\psi, v)$. Furthermore, formula φ is updated by replacing every

occurrence of $\langle\langle\Gamma\rangle\rangle\psi$ with $atom_{\langle\langle\Gamma\rangle\rangle\psi}$. Finally, the soundness of the inductive step for formulas of type $\langle\langle\Gamma\rangle\rangle\psi$ follows by Lemma 7. For instance, as regards the case for $v = \mathrm{tt}$, by Def. 8, $((M, s) \models^3 \langle\langle\Gamma\rangle\rangle\psi) = \mathrm{tt}$ iff for some joint strategy F_Γ^{must}, for all paths $p \in out(s, F_\Gamma^{must})$, $((M, p) \models^3 \psi) = \mathrm{tt}$. Now notice that F_Γ^{must}-strategies in M corresponds to standard F_Γ-strategies on restriction M^{must}. As a result, by Lemma 7, $((M, s) \models^3 \langle\langle\Gamma\rangle\rangle\psi) = \mathrm{tt}$ iff for some joint strategy F_Γ, for all paths $p \in out(s, F_\Gamma)$, $(M^{must}, p) \models^2 Transl(\psi, \mathrm{tt})$, iff $(M^{must}, p) \models^2 \langle\langle\Gamma\rangle\rangle Transl(\psi, \mathrm{tt})$. The case for $v = \mathrm{ff}$ can be proved similarly.

As regard complexity, the procedure recursively solves the formula in a polynomial number of steps (in the size of the formula). In line 17 (resp. 19), it calls subroutine $MC2()$ over restriction M^{must} (resp., M^{may}), that is the model checking procedure under perfect information and perfect recall for two-valued ATL^*. Since the latter problem is known to be 2EXPTIME-complete for the case of ATL^* and PTIME-complete for ATL [3], the complexity result follows. □

Remark 1. *Given Algorithm 7, we can solve the model checking problem simply by checking whether* $s_0 \in MC3(M, \varphi, \mathrm{tt})$. *Furthermore, by Theorem 1, Algorithm 7 is optimal, that is, we can not provide an algorithm that solves the same problem with better complexity. Notice that we can define* $MC3(M, \varphi, \mathrm{uu})$ *as* $S \setminus (MC3(M, \varphi, \mathrm{tt}) \cup MC3(M, \varphi, \mathrm{ff}))$. *Moreover, we overload the symbol* $MC3()$ *by using* $MC3(M, \varphi)$ *to denote the truth value of* φ *in* M, *that is,* $s_0 \in MC3(M, \varphi, v)$ *for* $v = MC3(M, \varphi)$. *In the rest of the paper, when it can be evinced from its context, we refer to Algorithm 7 for both the versions* $MC3(M, \varphi, v)$ *and* $MC3(M, \varphi)$.

9. The Complete Verification Procedure

In Algorithm 8 we report the high-level iterative procedure for model checking ATL^* formulas under the assumptions of imperfect information and perfect recall. Given an iCGS M and Γ-formula φ to check, we first construct the abstract (perfect information) CGS M_Γ based on M and Γ, by using procedure $Abstraction()$ as described in Def. 9. Then, we model check formula φ in the abstract model M_Γ by using procedure $MC3()$ described in Algorithm 7, which is decidable by Theorem 1. If a defined truth value $v \neq \mathrm{uu}$ is returned, we output it, since by Theorem 2 this result can be transferred to the original verification problem. On the other hand, if $(M_\Gamma \models^3 \varphi) = MC3(M_\Gamma, \varphi) = \mathrm{uu}$ then we enter the refinement loop in lines 5-10. We start by calling procedure $FailureState(s_\Gamma, \varphi)$ to find failure state s_f that makes formula φ undefined (line 6). Then, line 7 calls function $Refinement(M_\Gamma, M, s_f)$ with s_f as input. We then check again formula φ in the refined model M_Γ (line 8), and if the result is defined, we exit the loop by returning the defined truth value (line 9).

When the instructions in lines 6-9 are terminated, we check the truth value of the boolean variable $split$ that is returned by the refinement procedure (line 10). We recall that $split$ is false if and only if the model has not been refined in the last iteration of the loop. If this is the case, the loop is exited (and uu is returned), as it is not possible to refine the model in a way to make the formula defined with our procedure.

We now discuss the complexity of our model checking algorithm.

Algorithm 8 $ModelCheckingProcedure(M, \varphi)$

Input: iCGS M, Γ-formula φ.
Output: truth value $v \in \{\mathrm{tt}, \mathrm{ff}, \mathrm{uu}\}$.

1: M_Γ := $Abstraction(M, \Gamma)$;
2: **if** $MC3(M_\Gamma, \varphi) \neq \mathrm{uu}$ **then**
3: **return** $MC3(M_\Gamma, \varphi)$;
4: **else**
5: **repeat**
6: $(s_f, \psi) := FailureState(s_\Gamma, \varphi)$;
7: $(M_\Gamma, split) := Refinement(M_\Gamma, M, s_f)$;
8: **if** $MC3(M_\Gamma, \varphi) \neq \mathrm{uu}$ **then**
9: **return** $MC3(M_\Gamma, \varphi)$;
10: **until** $\neg split$;
11: **return** uu;

Theorem 6. *$ModelCheckingProcedure(M, \varphi)$ terminates in 2EXPTIME if φ is in ATL^*, in PTIME if φ is in ATL.*

Proof. We analyze in some detail Algorithm 8. The procedure of abstraction has to explore a polynomial number of states to generate the abstract model. Since the abstraction procedure returns a CGS with perfect information, we can use Algorithm 7 for the verification of formulas in ATL^* (resp. ATL). By Theorem 5, this leads to a model checking procedure in 2EXPTIME (resp. PTIME). The loop in lines 5-10 is shown to be polynomial by variable $split$ that guarantees termination. In fact, the number of refinements is polynomial in the size of the model. Furthermore, procedure $FailurePath()$ explores a polynomial number of states and formulas, and procedure $FailureState()$ builds an automaton, whose size is exponential in length of the formula to be checked for ATL^*, of constant size for ATL. So, it appears that the refinement procedure (lines 6-7) can be performed in EXPTIME (resp. PTIME) overall. Actually, the automaton $A_{\psi,\mathrm{uu}}$ and product $M \oplus A_{\psi,\mathrm{uu}}$ can be built and checked for emptiness on-the-fly, in principle, by using only a polynomial amount of space. In both cases, the complexity is dominated by model checking ATL^* (resp. ATL), and therefore the overall complexity of our procedure is in 2EXPTIME (resp. PTIME). □

By Theorem 6 the complexity of our partial model checking procedure is high for ATL^*, but only in PTIME for ATL. This is still better than the general undecidability result.

We conclude by presenting the last part of our example.

Example 7. *As an example of the application of our procedure, we consider once again the model proposed in Example 1. By line 1, the abstraction procedure produces the model presented in Example 2. Since the formula $\varphi = \langle\langle\Gamma\rangle\rangle F(l_1 \wedge \neg bUg)$ is undefined, the condition in line 2 is not satisfied and the algorithm enters the loop in lines 5-10. For the initial state a_1 in the abstract model, the algorithm checks whether φ is undefined at a_1. This is indeed the case. As showed in Example 5 the procedure $FailureState()$ returns failure state a_2 and atom l_1. So, the $ModelCheckingProcedure()$*

calls the $Refinement()$ procedure. Here, given a_2 as failure state, as described in Example 6, the $Refinement()$ procedure splits state a_2 into new states a_2^1 with concrete states s_1 and s_2, and a_2^2 with concrete states s_3 and s_4. Then, by calling $MC3()$ once again, formula φ is defined (specifically, true) and this ends the whole procedure.

Finally, notice that $ModelCheckingProcedure()$ does not necessarily terminate *with a defined truth value*. Indeed, the model checking problem for ATL (and *a fortiori* for ATL^*) in case of imperfect information and perfect recall is undecidable in general. This is meant to be a sound, albeit partial, verification algorithm.

10. Implementation

The entire approach presented in this paper has been implemented in Java and the resulting tool is publicly available as a GitHub repository[2]. The tool is structured as follows:

1. Java classes which instantiate the various steps of the proposed approach (Sec. 10.1).
2. A web-based GUI to help the user to interact with the tool (Sec. 10.2).

10.1. Java Classes

The procedure described in Algorithm 8 has been fully implemented in Java. The main component of such procedure is the iCGS, which has its corresponding Java class (*ATLModel* class).

Model reading, storing and representation. In the *ATLModel* class the information about states, transitions, agents, and coalitions are stored. To have a direct way to import and export iCGSs, the *ATLModel* class implements the *JsonObject* interface[3]. Through custom annotations, the class fields are exposed and serialised. In this way, given an *ATLModel* object, it is possible to automatically generate its corresponding Json character string representation, and vice versa. This allows a straightforward and lightweight management of the import/export of iCGSs inside the tool. Note that, when our iCGSs are verified using the MCMAS model checker, they need to be first translated into Interpreted Systems [27]. In fact, MCMAS does not support iCGSs, and it expects Interpreted Systems expressed using a domain specific language called Interpreted Systems Programming Language (ISPL). Thus, a processing step before calling MCMAS is always required, where the iCGS of interest (or better its *ATLModel* object representation) is first translated into its ISPL representation. This is only a technical detail, since iCGSs and Interpreted Systems present the same level of expressiveness [14, 32].

[2] https://github.com/VadimMalvone/A-Tool-for-Verifying-Strategic-Properties-in-MAS-with-Imperfect-Information
[3] https://docs.oracle.com/javaee/7/api/javax/json/JsonObject.html

Abstraction. As described in Algorithm 8, the first step of the procedure consists in abstracting the iCGS according to the chosen coalition (line 1). In the abstract CGS M_Γ, as described in Def. 9, states are clustered according to the common knowledge indistinguishability relation for the chosen coalition of agents. This is obtained in the implementation by generating two different abstract CGSs, named *must* and *may* models (which correspond to the M^{must} and M^{may} models described in Sec. 4). These models represent the abstraction of the input CGSs where only *must* and *may* transitions are considered, respectively. Both *must* and *may* models contain the same states, as only the transitions are different. The transitions inside the *must* model correspond to function δ^{must}, while the transitions inside the *may* model correspond to δ^{may}. We split the abstraction of an iCGS into a *must* and a *may* component because our implementation depends on the MCMAS model checker. Since MCMAS supports only standard CGSs, we need a way to represent abstract CGSs in MCMAS, without modifying the tool (as shown in Algorithm 7). Splitting a CGS into its *must* and *may* components is a practical way to tackle this issue.

Verification in MCMAS. Moving on with Algorithm 8, we find the first call to the model checking procedure $MC3()$, whose structure is reported in Algorithm 7. Here, we check if the CGS M_Γ can be verified as it is (*i.e.*, without performing any refinement step). In Algorithm 7, MCMAS is called on the corresponding $must$ or may model, depending on the structure of the property. Note that, at this point, each atom a is duplicated into a positive a_{tt} and a negative atom a_{ff} through the $Transl()$ function, where a_{tt} (resp. a_{ff}) denotes the occurrence of a being true (resp. false). This is fundamentally different from standard negation in LTL. Here, we need to explicitly recognise when an atom is observed to be true (resp. false), and the easiest way to do so is by splitting each atom in the formula to check accordingly (Algorithm 6). If the verification step returns tt, then the tool reports the satisfaction of the formula. Otherwise, if the verification step concludes ff, then the tool reports its violation. Finally, in case no defined outcome is concluded, the procedure moves on with undefined uu returned by Algorithm 7 (overloaded according to Remark 1).

Refinement. In case Algorithm 7 returns uu, Algorithm 8 progresses to model refinement. First, it calls the $FailureState()$ function to extract the state s_f that may be causing the property to be evaluated as uu in M_Γ. Then, the $Refinement()$ function is called. The refinement of the model is performed according to Algorithm 3. Specifically, the Java implementation of Algorithms 4 and 5 is immediate, with the only difference being that instead of having a single abstract CGS, we have two: the *must* and *may* models. Note that, from a practical perspective, the *FailureState* and *FailurePath* algorithms for a formula φ have been implemented differently (this will be thoroughly explained in the next two paragraphs) from what presented in Algorithm 1 and 2, since they are the only places where the GNBA $A_{\psi,uu}$ is used (the automaton which accepts exactly the infinite paths that evaluate ψ to undefined).

Three-valued automaton. Differently from the approach described in Sec. 5, in the implementation we work only on automata. The results concerning the computations over iCGSs and three-valued automata are obtained by first translating the former into

equivalent automata. Then, all the steps in Algorithm 1 are performed by implementing operations on automata. This choice has been done only for implementation purposes, and it does not affect the final result. The construction of the three-valued automaton is based on Def. 11 and 12 and it is implemented through the Antlr parser generator[4] by defining a customised design pattern. This parser analyses the ATL formula and extracts its closure, which is then used to construct the GNBA. This generation step has been implemented in Java. The GNBA of interest recognises the language of paths that make the ATL property undefined, which differs from the standard GNBA construction, where the automaton aims to satisfy (resp. violate) the property instead. Together with the GNBA, also its product with an iCGS has been implemented in Java. In more detail, the product is obtained by first translating the iCGS (the *ATLModel* object) into the corresponding GNBA. This can be done by following a similar procedure whereby a Kripke structure is transformed into its corresponding GNBA [6]. Note that, since iCGSs do not have the notion of final states, in the resulting GNBA all states are final. Intuitively, we can think of this GNBA as an automaton that recognises the language of paths that characterise the iCGS executions. Once such automaton is obtained, the product with automaton $A_{\psi,uu}$ can be computed (line 12 of Algorithm 1).

Must and May model instantiation. Since at the implementation level, the iCGS is divided into two models ($must$ and may), also the corresponding GNBA needs to be split into two different GNBAs: one for the $must$ and the other for the may model. First, a GNBA corresponding to the iCGS including *must* transitions is generated. Then, the same procedure is followed concerning *may* transitions. The product between the two GNBAs[5], which corresponds to the standard GNBA product definition [6], has been implemented in Java. Once the two product automata are obtained, we can use them to extract the infinite paths in the iCGS that make ψ undefined. Specifically, we first look for paths in the *must* product. Then, in case no such path exists, we look for paths in the *may* product instead. In more detail, we start looking into the *must* product first because its language is included in the language recognised by the *may* product by construction. If we find a suitable path in the *must* product, then it will also appear in the *may* model. In case no path is found, then the procedure moves to the *may* product, where a suitable path can still be found, even though belonging only to the *may* product (*i.e.*, the over-approximation of the model). If no such a path exists in either of the two product automata, which means that they both recognise the empty language, then the *FailureState* procedure terminates (as shown in Algorithm 1). But, if at least one such paths exists, then the *FailurePath* procedure is called (Algorithm 1, line 16). The latter, from an implementation perspective, is challenging w.r.t. how the model checking step has been implemented, that is using paths instead of states. This feature appears in Algorithm 2, lines 6, 13, and 15, where the model checker is supposed to be called with a path as second argument (instead of a state as before). The verification has been achieved by generating the product automaton $M \otimes A_{\varphi,uu}$ of the CGS M

[4]Antlr is a powerful parser generator for reading, processing, executing, and translating structured text: `https://www.antlr.org/`.

[5]Specifically, between the GNBA considering *must* transitions and the undefined automaton $A_{\psi,uu}$, and between the GNBA considering *may* transitions and the undefined automaton $A_{\psi,uu}$.

with the property of interest φ (as explained above). Then, since on lines 6, 13 and 15, the property is verified against a path p, an additional product automaton is generated between $A_{\varphi,uu}$ and the GNBA obtained by p. If such product recognises the empty language, then it means that path p does not make φ undefined in M. But, if such a product recognises at least one path, then it means that p actually makes φ undefined in M.

Practical aspects of automata construction. We here discuss at some length the practical differences in the three-valued automaton construction. In Sec. 5, we introduced the theory behind the generation of the GNBA $A_{\psi,\mathrm{uu}}$ which recognises the paths that make ψ undefined (hence, neither satisfy nor violate it). The resulting implementation corresponds to a direct map of the theory, where the GNBA is generated and handled as a Java class, which stores states, transitions, and atoms (labels), as described in the previous paragraph. However, some additional technical aspects need to be considered at the practical level.

The alphabet needs to be explicit. Differently from two-valued LTL-to-automaton translation [30], we need to duplicate all atoms in order to make them explicit. The reason is similar to the one of the $Transl()$ function. Again, the assumption whereby the lack of an atom is equivalent to its being false is not always convenient. A way to solve this issue is to explicitly specify which atoms are true (a_{tt}) and which are false (a_{ff}). As a consequence, it is necessary to unroll all these atoms and to explicitly populate the corresponding automaton's transitions accordingly to their being true, false, or undefined (and therefore absent).

The rest of Algorithm 8 is immediate. Either it uses methods that we have already described, or it performs computations that are easy to map to the corresponding Java implementation.

10.2. GUI

The GUI is implemented as a web server through the Spring framework[6], which provides a comprehensive programming and configuration model for modern Java-based enterprise applications, on any kind of deployment platform. We decided to use Spring because it is a very flexible framework, and it allowed us to quickly develop a user-friendly web-based GUI for our tool.

Figure 6 shows the GUI when the tool starts. On the top, a brief description of how the tool works is reported. On the left, the input model can be specified (in Json). This is the model that the user wants to analyse. On the right, the results of the model checking procedure are visualised. Here, the user can check the resulting *must* and *may* models that have been produced through the procedure. Additionally, buttons to interact with the tool are present. In more detail:

- A *Transform* button, which calls our model checking procedure and populate the cells on the right of the GUI with its results.

[6] `https://spring.io/`

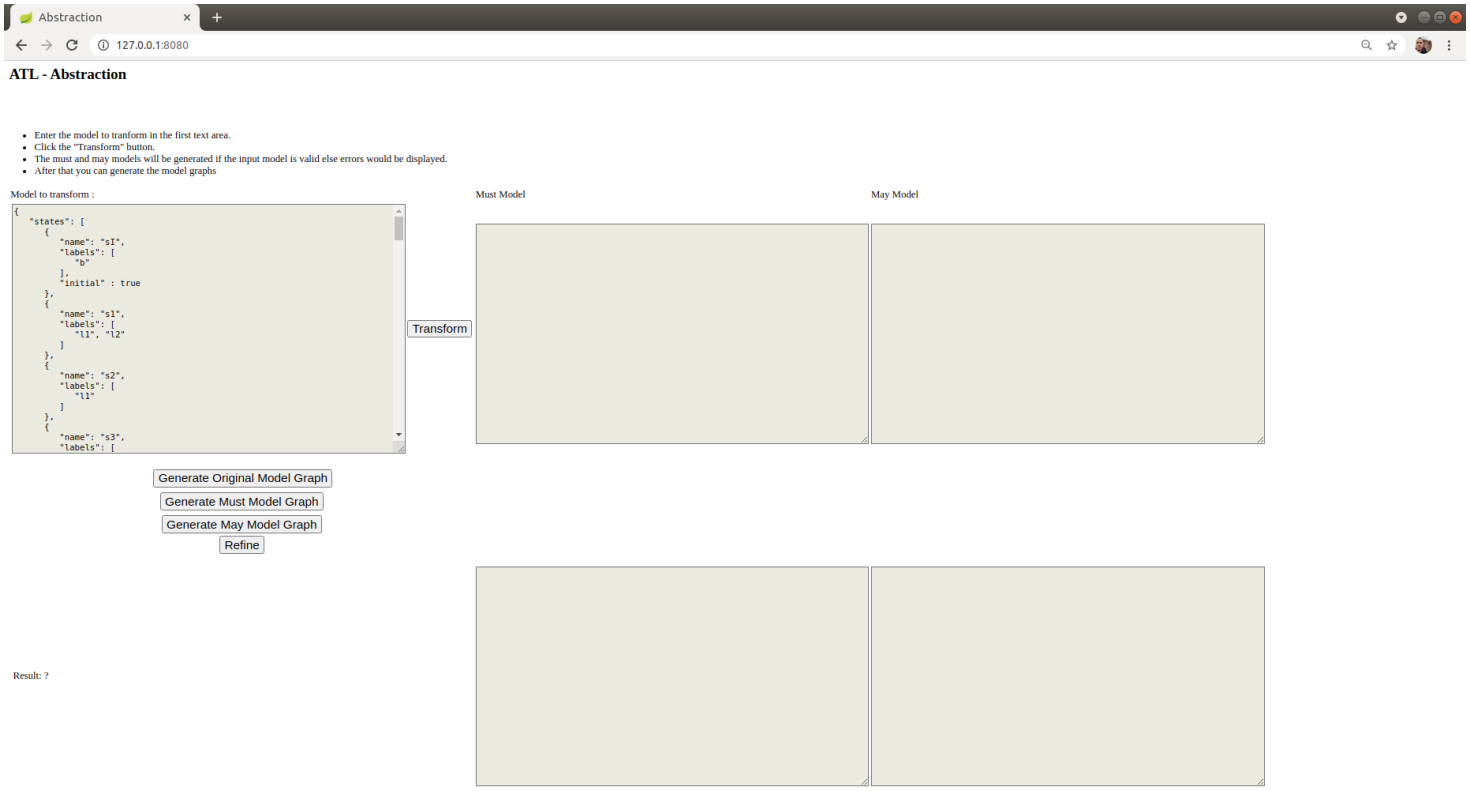

Figure 6: GUI when tool starts.

- A *Generate Original Model Graph* button, which shows a graphical representation of the input model.
- A *Generate Must Model Graph* button, which shows a graphical representation of the *must* model.
- A *Generate May Model Graph* button, which shows a graphical representation of the *may* model.

Figure 7 shows the GUI after the *Transform* button has been pressed. Differently from Figure 6, now the boxes on the right of the GUI are populated with the resulting models and the verification result (as given by MCMAS). In fact, for both the *must* and *may* models, MCMAS is called and depending on the result (as per Algorithm 8) the model checking procedure progresses.

10.3. Experimental Results

We made use of Example 1 to illustrate our theoretical contribution. We showed how the algorithms work on this example and we reported the corresponding results step by step. However, at the implementation level, we have tested our tool on a larger set of iCGSs. Specifically, we carried out two lines of experiments. First, we experimented with our tool on iCGSs of different size (defined as the number of states), while keeping the formula to verify fixed. Then, we did the opposite: we experimented with our tool keeping the iCGS fixed, and varying the size of the formula, in particular the number of strategic operators. Note that, our experiments are restricted to ATL because of limitations of MCMAS, which only supports ATL formulas. However, the theory presented in this paper, as well as its implementation, is more general and can be applied to ATL^*. Thus, if in the future another model checker supporting ATL^* becomes available, then our tool will not need to be changed (*i.e.*, only the corresponding call to the model checker will need updating in case).

Figure 8 reports the results for the first line of experiments. On the x-axis, we may find the number of states of the analysed model, while on the y-axis, we have the corresponding time (in seconds) required by our tool to complete the verification step. Let us now go into a bit more of detail on what kind of models we ran experiments on. First of all, the results reported in Figure 8 have been collected over a range of 7000 randomly generated models (1000 models for each size considered in the experiments). Each model was randomly generated as a transformation from the model presented in Example 1. Which means, starting from that model, we randomly generated an additional number of new states to add to the model and an additional number of new transitions to connect those states. Now, one may wonder why building such models as extensions of the iCGS in Example 1, instead of generating them from scratch. In fact, initially we generated completely random models, but the results were too positive, as most of such models did not need refinement. Consequently, when called on these models, Algorithm 8 would just conclude the satisfaction (or violation) in line 2, returning the corresponding result in line 3. Indeed, our refinement method was not applied, but only standard ATL model checking was. In order to tackle this issue, we decided to generate random models, although based on a well-known structure (*i.e.*, the iCGS in Example 1), which we know requires refinement. In this way, we can

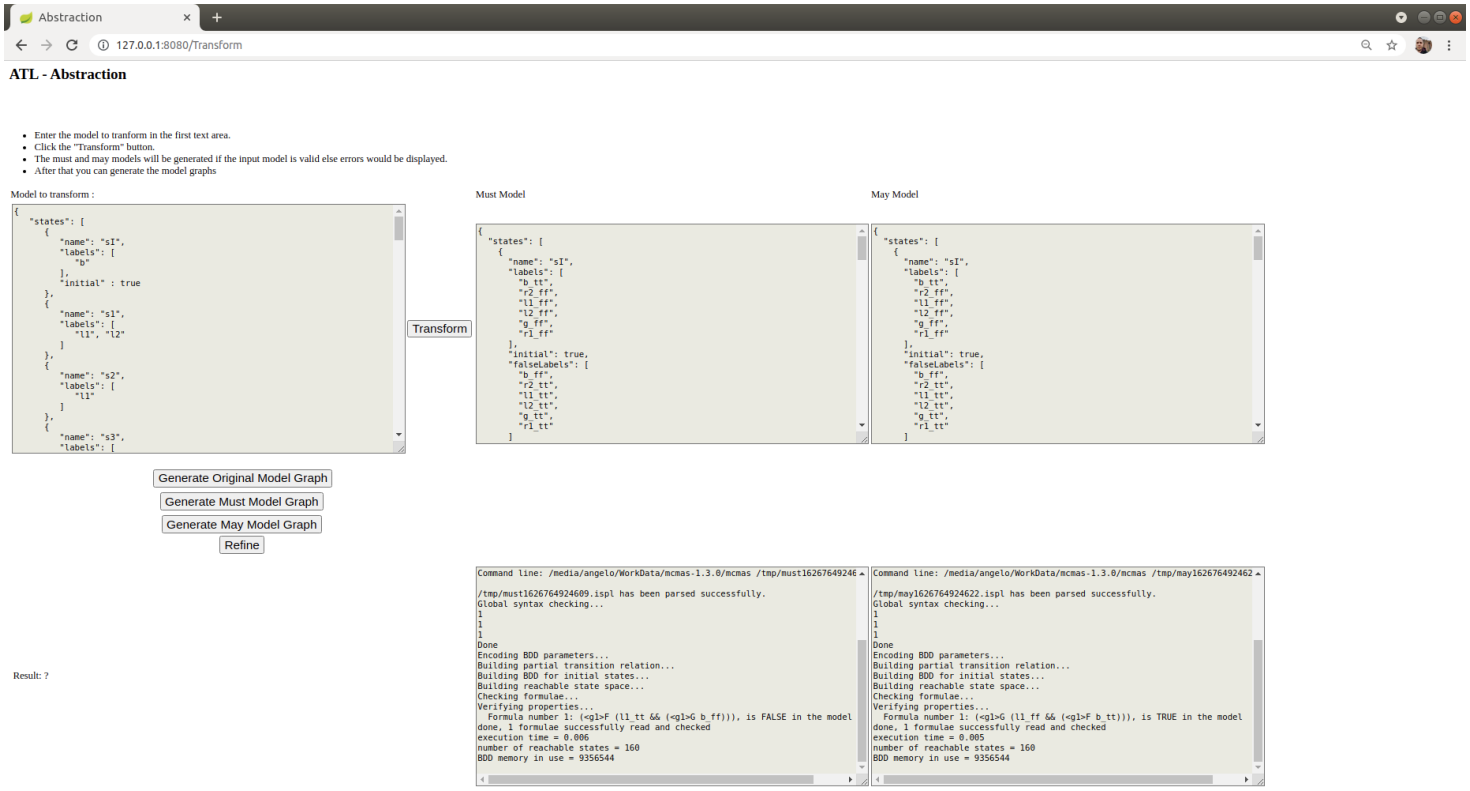

Figure 7: GUI after clicking *Transform* button.

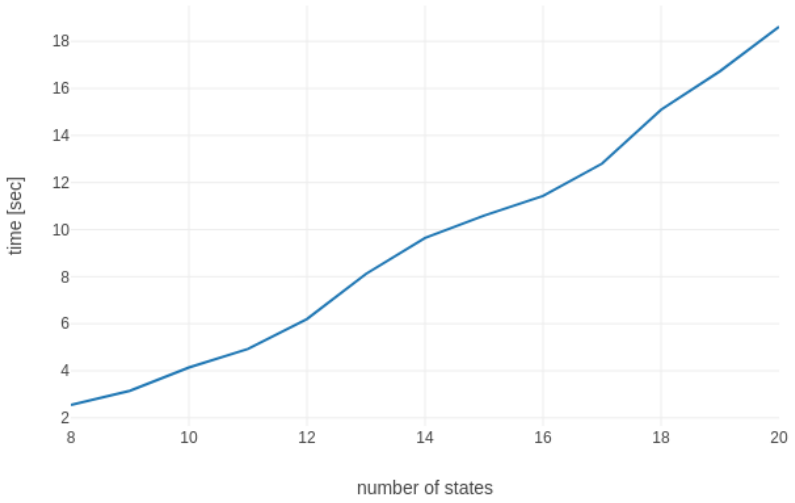

Figure 8: Results obtained by varying the size of the iCGS.

carry out the experiments on a large number of random iCGSs of which we know the base topology. Note that, we may observe a pseudo-quadratic behaviour, instead of the expected exponential one, because MCMAS only supports ATL, and the model checking problem for ATL under perfect information is PTIME-complete [3].

The second line of experiments concerns the behaviour of our tool when varying the size of the ATL formula to verify. In these experiments, we fixed the iCGS to the one in Example 1, and varied the number of strategic operators inside the formula. On the x-axis we reported the number of operators, while on the y-axis the time (in seconds) required by our tool to perform the verification. The results reported in Figure 9 show a pseudo-linear behaviour w.r.t. the size of the formula. The reason of this lies in Algorithms 1 and 7; where the procedures iterate over the strategic operators in formula φ to check. Naturally, these experiments do not imply that in the worst case scenario the algorithm would not grow exponentially (which would be indeed the case), but show how the algorithm behaves in a more random setting, where strategic operators are added.

10.4. Benchmarks

In addition to the experiments presented above, we carried out additional benchmarks considering a variant of the simple voting scenario presented in [36], and used in [11] to evaluate the partial model checking procedure proposed therein. This 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

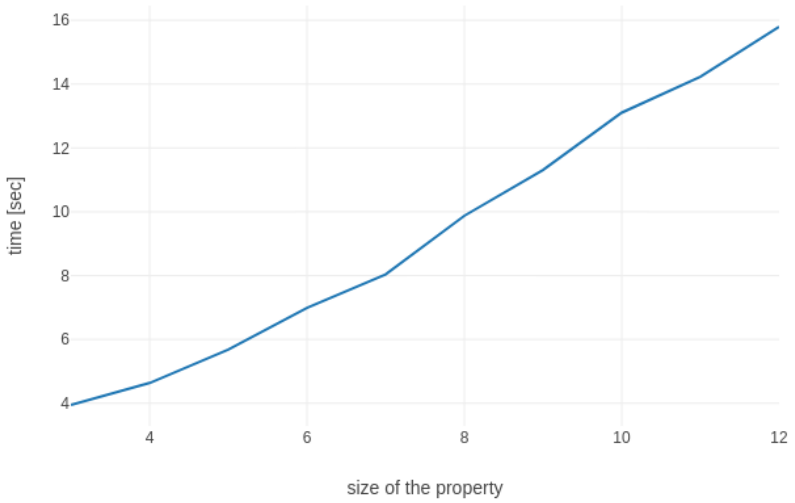

Figure 9: Results obtained by varying the size of the formula.

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 10 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 10 we omit the index i for the voter's actions.

This scenario and the corresponding iCGS is used in [11] to analyse the expressive power of bounded-recall strategies. 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:

$$\varphi_3 \quad = \quad \langle\langle Coercer \rangle\rangle F((vote_{i1} \wedge n\_p_i) \vee (\bigvee_{j=2}^{k} vote_{ij} \wedge p_i) \vee (n\_give_i \wedge p_i))$$

We observe that φ_3 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 10 are indistinguishable for the coercer). However, the coercer has a $(t+1)$-bounded recall strategy (in the sense of [11]) to win the game, where t is the number of waiting steps.

Furthermore, in [11] the authors evaluate the ATL specification φ_4 stating that the voters collectively have a strategy to avoid being punished:

$$\varphi_4 = \langle\langle all\_voters \rangle\rangle G \neg (\bigvee_{i=1}^{n_v} (n\_give_i \wedge n\_p_i) \vee (\neg n\_give_i \wedge p_i))$$

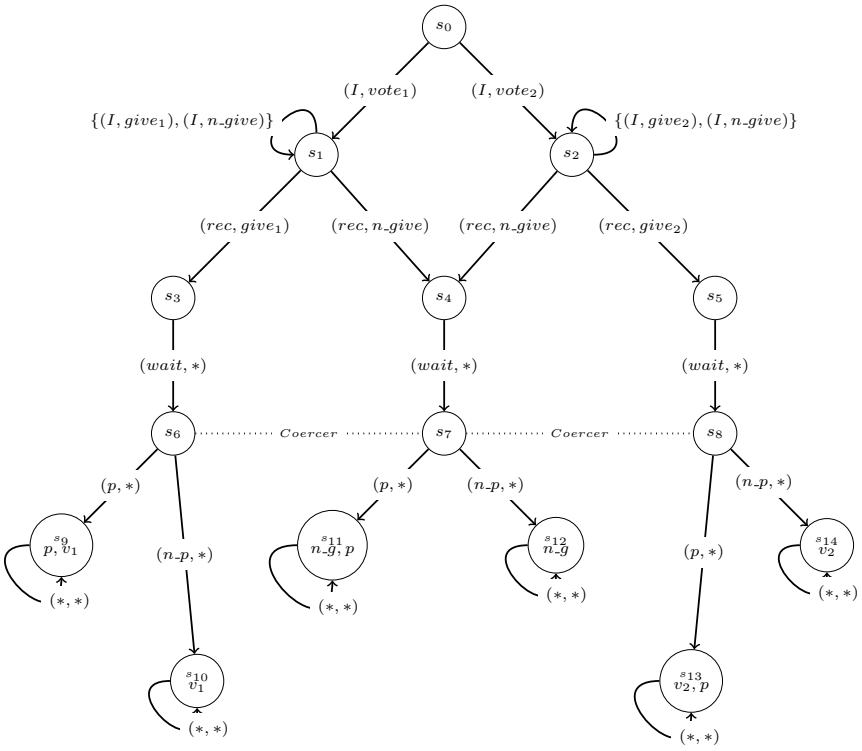

Figure 10: The iCGS M for the simple voting scenario. We consider the setting with one waiting step before the Coercer makes a decision for punishment. Here, action $*$ represents any action available for the agent and action I represents the idle action. The states $s_9, \ldots, s_{14}$ are labeled 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 voting scenario is a good candidate to compare our verification approach with the one proposed in [11]. In that work, the authors consider the same problem as we do here, *i.e.*, ATL^* model checking under imperfect information and perfect recall, but they approximate perfect recall by considering bounded-recall strategies with an increasingly greater bound; while, here, we approximate imperfect information (by an abstraction refinement method).

Table 1 reports the results obtained by applying the two approaches when verifying formula φ_4. Note that, both approaches conclude the same verdict, that is, the violation of the property. However, our approach concludes the verification step requiring longer computation time. This is due specifically to the size of the model and its representation. Indeed, [11] suffers from increasing the bound (of the recall), whereas our approach suffers from increasing the number of states (as usual in model checking techniques). In particular, the voting scenario is a model which best suits the bounded approach presented in [11]. To perform the comparison reported in Table 1, we had to port the ISPL implementation (which was used in [11]) into its equivalent iCGS

Voters	Waiting	Bound	Verdict [11]	Verdict (Ours)	[11]	Ours
2	6	7	ff	ff	**2.8**	**135.3**
2	6	9	ff	ff	5.8	135.3
2	6	11	ff	ff	11.1	135.3
2	6	13	ff	ff	18.1	135.3
2	8	7	uu	ff	3.2	139.1
2	8	9	ff	ff	**6.8**	**139.1**
2	8	11	ff	ff	12.4	139.1
2	8	13	ff	ff	18.5	139.1
2	10	7	–	ff	T/O	143.6
2	10	9	–	ff	T/O	143.6
2	10	11	–	ff	T/O	143.6
2	10	13	–	ff	T/O	143.6
3	6	7	ff	ff	**7.7**	**3508.4**
3	6	9	ff	ff	171.3	3508.4
3	6	11	ff	ff	599.6	3508.4
3	6	13	ff	ff	1503.4	3508.4
3	8	7	uu	ff	7.7	3762.5
3	8	9	ff	ff	**173.6**	**3762.5**
3	8	11	ff	ff	192.0	3762.5
3	8	13	ff	ff	1042.6	3762.5
3	10	7	–	ff	T/O	3915.1
3	10	9	–	ff	T/O	3915.1
3	10	11	–	ff	T/O	3915.1
3	10	13	–	ff	T/O	3915.1

Table 1: Benchmarks results for property φ_4 (all times are in seconds), where T/O is reported when time superior at 2 hours (timeout).

representation (which is the formalism we use). This translation entails an explosion in the number of states, since the iCGS explicitly represents all transitions. For this reason, the verification step takes longer. Nonetheless, it is important to observe that our verification approach is not influenced by the number of waiting steps (which were specifically added to highlight the bounded scenario), while [11] is. Furthermore, by increasing the bound, the bounded approach's computation time grows exponentially, since a greater bound is required to remember visited states, while our approach is marginally influenced (the small increment in the computation time is caused by the additional waiting states that are added). Moreover, Belardinelli *et al.*'s approach requires a bound as parameter to properly function, while we do not require such parameter (our execution times in Table 1 are not influenced by the choice of the bound). Since such bound cannot be known in advance, the real computation time required is actually higher. Indeed, it is the sum of all computation times for each bound tested before finding the one for which a conclusive verdict can be concluded. Another im-

portant aspect to report is that our approach did not need to apply any refinement step to conclude. Thus, only the verification of the *must* and *may* abstractions was necessary.

To further evaluate our tool against the simple voting scenario, we also carried out experiments on φ_3. The verification of φ_3 requires refinement, when the imperfect information is propagated to the leaf states of the iCGS, *i.e.*, $s_9, \ldots, s_{14}$. Note that, the iCGS presented in Figure 10 has imperfect information on s_6, s_7, and s_8 w.r.t. the coercer. However, such imperfect information does not require any additional step of refinement. Instead, if we move the imperfect information on $s_9, \ldots, s_{14}$, then a refinement step is required to verify the property. Thanks to such modification, we could test the whole of our method, which reported the right verdict (*i.e.*, tt), on a more complicated scenario, where the verification on *must* and *may* abstractions was not enough (indeed an additional step of refinement was required).

The previous benchmarks were meant to show that our approach can be used to tackle problems already handled by [11]. However, we also experimented the other way around. Specifically, we translated our train case study into its equivalent ISPL, and we applied Belardinelli *et al.*'s approach. The resulting verification outcome was uu, while with our approach we conclude with a conclusive verdict. Note that, even though such empirical results would make us think that our approach is more general (*i.e.*, all problems that can be solved by [11] can be solved by our approach as well), the two techniques are actually orthogonal. In fact, it is not difficult to find problems for which [11] would return a conclusive result, while our approach would not. Specifically, a good candidate set of problems are all those where the property to verify contains different strategic operators (more precisely, properties with at least two strategic operators based on different agents' coalitions). Such properties are not currently handled by our approach, which focuses on strategic properties with only one coalition of agents involved.

11. Conclusions

As discussed in the introduction, one of the key issues in deploying logics for strategic reasoning, such as ATL and ATL^*, in the context of Multi-agent Systems is that their model checking problem is undecidable under the assumption of perfect recall and incomplete information. Yet, this is one of the most natural and compelling setup in applications. Finding appropriate approximations remains a key question at present.

In this paper we have put forward a notion of abstraction between systems with perfect and imperfect information to overcome this issue. Specifically, we showed that iCGS with imperfect information admit a (perfect information) abstraction which preserves satisfaction back to the original model, when checked under a three-valued semantics. This enabled us to give an incomplete but sound procedure for the original model checking problem, which is undecidable in general. We implemented the whole procedure in a model checking suit and provided experimental results. One important limitation of the proposed approach is that it works for formulas with a fixed coalition Γ of agents appearing in coalition operators (Γ-formulas). Still, within this class of formulas we can deal with arbitrary alternation of temporal operators (i.e., ATL^*-formulas). Moreover, we can envisage an extension of the proposed approach whereby

different abstractions are applied and refined for the different coalition operators appearing in the formula at hand. Such an extension would require careful handling of nested coalition operators.

An interesting question that we would like to explore as future work is to find the "most promising" failure states. It might be possible to devise robust heuristics to find good candidates for refinement. Furthermore, we would like to find a method to handle non-Γ formulas in our procedure. Finally, we plan to extend the verification techniques here developed to more expressive languages including Strategy Logic [21, 49] in the light of the recent tractable verification results in [8].

Acknowledgements. We thank Abbas Slimani for his contribution to the implementation of our tool. F. Belardinelli acknowledges the support of ANR JCJC Project SVeDaS (ANR-16-CE40-0021).

References

[1] T. Ågotnes, V. Goranko, W. Jamroga, and M. Wooldridge. Knowledge and ability. In *Handbook of Logics for Knowledge and Belief*, pages 543–589, 2015.

[2] R. Alur, T. Henzinger, F. Mang, S. Qadeer, S. Rajamani, and S. Tasiran. MOCHA: Modularity in model checking. In *CAV98*, pages 521–525, 1998.

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

[4] B. Aminof, O. Kupferman, and A. Murano. Improved model checking of hierarchical systems. *Inf. Comput.*, 210:68–86, 2012.

[5] G. Avni, S. Guha, and O. Kupferman. An abstraction-refinement methodology for reasoning about network games. In *IJCAI*, pages 70–76, 2017.

[6] C. Baier and J. P. Katoen. *Principles of Model Checking*. MIT Press, 2008.

[7] T. Ball and O. Kupferman. An abstraction-refinement framework for multi-agent systems. In *LICS06*, pages 379–388. IEEE, 2006.

[8] F. Belardinelli, W. Jamroga, D. Kurpiewski, V. Malvone, and A. Murano. Strategy logic with simple goals: Tractable reasoning about strategies. In *IJCAI*, pages 88–94, 2019.

[9] F. Belardinelli and A. Lomuscio. A three-value abstraction technique for the verification of epistemic properties in multi-agent systems. In *JELIA16*, 2016.

[10] F. Belardinelli, A. Lomuscio, and V. Malvone. Approximating perfect recall when model checking strategic abilities. In *KR2018*, 2018.

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

[12] F. Belardinelli, A. Lomuscio, A. Murano, and S. Rubin. Verification of broadcasting multi-agent systems against an epistemic strategy logic. In *IJCAI'17*, pages 91–97, 2017.

[13] F. Belardinelli, A. Lomuscio, A. Murano, and S. Rubin. Verification of multi-agent systems with imperfect information and public actions. In *AAMAS 2017*, pages 1268–1276, 2017.

[14] F. Belardinelli, A. Lomuscio, A. Murano, and S. Rubin. Verification of multi-agent systems with public actions against strategy logic. *Artif. Intell.*, 285:103302, 2020.

[15] F. Belardinelli and V. Malvone. A Three-valued Approach to Strategic Abilities under Imperfect Information. In *KR*, pages 89–98, 2020.

[16] F. Belardinelli, A. Lomuscio, and V. Malvone. An abstraction-based method for verifying strategic properties in multi-agent systems with imperfect information. In *AAAI*, pages 6030–6037, 2019.

[17] R. Berthon, B. Maubert, and A. Murano. Decidability results for ATL* with imperfect information and perfect recall. In *AAMAS'17*, pages 1250–1258, 2017.

[18] R. Berthon, B. Maubert, A. Murano, S. Rubin, and M. Y. Vardi. Strategy logic with imperfect information. In *LICS*, pages 1–12, 2017.

[19] G. Bruns and P. Godefroid. Model checking with multi-valued logics. Technical Report ITD-03-44535H, Bell Labs, 2003.

[20] N. Bulling and W. Jamroga. Alternating epistemic mu-calculus. In *IJCAI-11.*, pages 109–114, 2011.

[21] K. Chatterjee, T. Henzinger, and N. Piterman. Strategy logic. In *CONCUR07*, pages 59–73, 2007.

[22] M. Chechik, B. Devereux, and A. Gurfinkel. Model-checking in finite state-space systems with fine-grained abstractions using spin. In *SPIN*, pages 16–36, 2001.

[23] M. Cohen, M. Dam, A. Lomuscio, and F. Russo. Abstraction in model checking multi-agent systems. In *AAMAS09*, pages 945–952, 2009.

[24] L. de Alfaro and P. Roy. Solving games via three-valued abstraction refinement. *Inf. Comput.*, 208(6):666–676, 2010.

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

[26] R. Dimitrova and B. Finkbeiner. Abstraction refinement for games with incomplete information. In *FSTTCS*, pages 175–186, 2008.

[27] R. Fagin, J.Y. Halpern, Y. Moses, and M.Y. Vardi. *Reasoning about Knowledge.* MIT, 1995.

[28] A. Ferrando and V. Malvone. Strategy RV: A tool to approximate ATL model checking under imperfect information and perfect recall. In *AAMAS21*, pages 1764–1766, 2021.

[29] A. Ferrando and V. Malvone. Towards the combination of model checking and runtime verification on multi-agent systems. In *PAAMS*, pages 140–152. Springer, 2022.

[30] R. Gerth, D. Dolech, D. Peled, M. Vardi, and P. Wolper. Simple on-the-fly automatic verification of linear temporal logic. *Proceedings of the 6th Symposium on Logic in Computer Science*, 12 1995.

[31] R. Gerth, D. Peled, M. Vardi, and P. Wolper. Simple on-the-fly automatic verification of linear temporal logic. In *PSTV95*, pages 3–18, 1995.

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

[33] O. Grumberg, M. Lange, M. Leucker, and S. Shoham. When not losing is better than winning: Abstraction and refinement for the full mu-calculus. *Inf. Comput.*, 205(8):1130–1148, 2007.

[34] W. Jamroga. Logical methods for specification and verification of multi-agent systems, 2018.

[35] W. Jamroga and J. Dix. Model checking abilities under incomplete information is indeed Δ_p^2-complete. In *EUMAS'06*, pages 14–15, 2006.

[36] W. Jamroga, M. Knapik, D. Kurpiewski, and L. Mikulski. Approximate verification of strategic abilities under imperfect information. *Artif. Intell.*, 277, 2019.

[37] W. Jamroga, B. Konikowska, D. Kurpiewski, and W. Penczek. Multi-valued verification of strategic ability. *Fundam. Informaticae*, 175(1-4):207–251, 2020.

[38] W. Jamroga, B. Konikowska, and W. Penczek. Multi-valued verification of strategic ability. In *AAMAS16*, pages 1180–1189, 2016.

[39] W. Jamroga, W. Penczek, T. Sidoruk, P. Dembinski, and A. Mazurkiewicz. Towards partial order reductions for strategic ability. *J. Artif. Intell. Res.*, 68:817–850, 2020.

[40] W. Jamroga and W. van der Hoek. Agents that know how to play. *Fund. Inf.*, 62:1–35, 2004.

[41] S. C. Kleene. *Introduction to Metamathematics*. Princeton, NJ, USA: North Holland, 1952.

[42] O. Kupferman and Y. Lustig. Lattice automata. In *VMCAI*, pages 199–213, 2007.

[43] D. Kurpiewski, W. Jamroga, and M. Knapik. STV: model checking for strategies under imperfect information. In *AAMAS19*, pages 2372–2374, 2019.

[44] A. Lomuscio and J. Michaliszyn. An abstraction technique for the verification of multi-agent systems against ATL specifications. In *KR14*, pages 428–437. AAAI Press, 2014.

[45] A. Lomuscio and J. Michaliszyn. Verifying multi-agent systems by model checking three-valued abstractions. In *AAMAS15*, pages 189–198, 2015.

[46] A. Lomuscio and J. Michaliszyn. Verification of multi-agent systems via predicate abstraction against ATLK specifications. In *AAMAS16*, pages 662–670, 2016.

[47] A. Lomuscio, W. Penczek, and H. Qu. Partial Order Reductions for Model Checking Temporal-epistemic Logics over Interleaved Multi-agent Systems. *Fundamenta Informaticae*, 101(1-2):71–90, 2010.

[48] A. Lomuscio, H. Qu, and F. Raimondi. MCMAS: an open-source model checker for the verification of multi-agent systems. *STTT*, 19(1):9–30, 2017.

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

[50] M. Pauly. A modal logic for coalitional power in games. *J. Log. Comput.*, 12(1):149–166, 2002.

[51] F. Raimondi and A. Lomuscio. The complexity of symbolic model checking temporal-epistemic logics. In *CS&P*, pages 421–432, 2005.

[52] S. Shoham and O. Grumberg. Monotonic abstraction-refinement for CTL. In *TACAS04*, pages 546–560, 2004.

[53] S. Shoham and O. Grumberg. A game-based framework for CTL counterexamples and 3-valued abstraction-refinement. *ACM Trans. Comput. Log.*, 9(1):1, 2007.