



How to Find Good Coalitions to Achieve Strategic Objectives

Angelo Ferrando¹ ^a and Vadim Malvone² ^b

¹Department of Informatics, Bioengineering, Robotics and Systems Engineering, University of Genoa, Genoa, Italy

²LTCI, Telecom Paris, Institut Polytechnique de Paris, Palaiseau, France

Keywords: Logics for the Strategic Reasoning, Alternating-Time Temporal Logic, Coalition of Agents, Formal Verification.

Abstract: Alternating-time Temporal Logic (ATL) is an extension of the temporal logic CTL in which we can quantify over coalition of agents. In the model checking process, the coalitions in a given formula are fixed, so it is assumed that the user knows the specific coalitions to be checked. Unfortunately, this is not true in general. In this paper, we present an extension of MCMAS, a well-known tool that handles ATL model checking, in which we give the ability to a user to characterise the coalition quantifiers with respect to two main features: the number of agents involved in the coalitions and how to group such agents. Moreover, we give details of such extensions and provide experimental results.


1 INTRODUCTION


Given the growing use of concurrent and reactive systems, formal verification for Multi-Agent Systems (MAS) becomes a fundamental task. The main contribution in this line of research is model checking. The latter is divided into three main goals: model the multi-agent system, specify the property of interest, and verify that the model satisfies the specification. To handle the second task in the model checking process, logics for the strategic reasoning have been proposed (Alur et al., 2002; Mogavero et al., 2014). One of the most popular logics for the strategic reasoning is Alternating-time temporal logic (ATL) (Alur et al., 2002). The latter is an extension of Computation Tree Logic (CTL) in which instead of having path quantifiers “there exists a path” E and “for all paths” A , we have strategic operators “there exists a collective strategy for the coalition A ” $\langle\langle A \rangle\rangle$ and “for all the strategies for the coalition A ” $\llbracket A \rrbracket$. The most popular tool for the model checking of multi-agent systems is MCMAS (Lomuscio et al., 2015). In this tool, the multi-agent system is formally modelled as an interpreted system that is a product of local models, one for each agent involved in the multi-agent system to represent its visibility. MCMAS provides the specification of properties via CTL, ATL (Alur et al., 2002), Strategy Logic (Mogavero et al., 2014), and some of their extensions/fragments. The tool handles

the model checking problem by using a Binary Decision Diagram (BDD) representation for models and formulas. Notice that, in the model checking process the coalitions in the strategic operators need to be fixed before the verification process. The latter constraint is not always well-known by the developers/users called to verify the multi-agent system.

In this paper, we present an extension of MCMAS in which we give the ability to the end user to characterise the coalitions in the strategy quantifiers with respect to two main features: the *number of agents* involved in the coalitions and *how to group* the agents. That is, we ask the user to give some information on the coalitions involved in each strategic operator by considering them as a variable of the problem. With more detail, the user can input a minimum and maximum of agents involved in the coalitions and give guidelines with respect to the agents that have to (resp., cannot) stay in the same coalitions. After that, our tool extracts all coalitions of agents that respect the user’s guidelines. Then, for each valid coalition, our tool verifies the formal specification over the multi-agent system. Finally, the coalitions that make the formal specification satisfied in the multi-agent system are returned to the user. We consider our work as a first stone on the development of a more generalised tool for the verification of multi-agent systems.

The paper is structured as follows. In Section 2 we give some related works on formal verification of multi-agent systems. In Section 3 we recall the formal definitions to model multi-agent systems as interpreted systems and to specify ATL properties. We

^a  <https://orcid.org/0000-0002-8711-4670>

^b  <https://orcid.org/0000-0001-6138-4229>

present a variant of the Train Gate Controller in Section 4. Then, in Section 5 we provide details on our idea and methodology. Finally, we give the details on our extension for MCMAS in Section 6 and, in Section 7, provide experimental results on a parameterised version of the Train Gate Controller Scenario. We conclude in Section 8 by recapping our work and open to some interesting future works.

2 RELATED WORK

In the introduction we mentioned another important logic for the strategic reasoning called Strategy Logic (Mogavero et al., 2014). The latter is a powerful formalism for strategic reasoning. As a key aspect, this logic treats strategies as *first-order objects* that can be determined by means of the existential $\exists x$ and universal $\forall x$ quantifiers, which can be respectively read as “*there exists a strategy x* ” and “*for all strategies x* ”. In logics for the strategic reasoning two key notions are the kind of strategies and the agents’ information. A strategy is a generic conditional plan that at each step of the game prescribes an action. With more detail, there are two main classes of strategies: memoryless and memoryful. In the former case, agents choose an action by considering only the current game state while, in the latter case, agents choose an action by considering the full history of the game. Therefore, in Strategy Logic, this plan is not intrinsically glued to a specific agent, but an explicit binding operator (a, x) allows to link an agent a to the strategy associated with a variable x . Unfortunately, the high expressivity of SL comes at a price. Indeed, it has been proved that the model-checking problem for SL becomes non-elementary complete (Mogavero et al., 2014) and the satisfiability undecidable (Mogavero et al., 2017). To gain back elementariness, several fragments of SL have been considered. Among the others, Strategy Logic with Simple-Goals (Belardinelli et al., 2019a) considers SL formulas in which strategic operators, bindings operators, and temporal operators are coupled. It has been shown that Strategy Logic with Simple-Goals strictly subsume ATL and its model checking problem is P-COMPLETE, as it is for ATL (Alur et al., 2002). To conclude this section, we want to focus on the agents’ information. Specifically, we distinguish between *perfect* and *imperfect* information games (Reif, 1984). The former corresponds to a basic setting in which every agent has full knowledge about the game. However, in real-life scenarios it is common to have situations in which agents have to play without having all relevant information at hand. In computer science these situ-

ations occur for example when some variables of a system are internal/private and not visible to an external environment (Kupferman and Vardi, 1997; Bloem et al., 2015). In game models, the imperfect information is usually modelled by setting an indistinguishability relation over the states of the game (Kupferman and Vardi, 1997; Reif, 1984; Pnueli and Rosner, 1990). This feature deeply impacts on the model checking complexity. For example, ATL becomes undecidable in the context of imperfect information and memoryful strategies (Dima and Tiplea, 2011). To overcome this problem, some works have either focused on an approximation to perfect information (Belardinelli et al., 2019b; Belardinelli and Malvone, 2020), developed notions of bounded memory (Belardinelli et al., 2018; Belardinelli et al., 2022), or developed hybrid techniques (Ferrando and Malvone, 2021b; Ferrando and Malvone, 2021a; Ferrando and Malvone, 2022).

3 PRELIMINARIES

In this section we show the syntax and semantics for ATL^* based on (Alur et al., 2002) by using interpreted systems as models. Hereafter we assume sets $Ag = \{1, \dots, m\}$ of indices for agents and AP of atomic propositions. Given a set U , \bar{U} denotes its complement. We denote the length of a tuple v of elements as $|v|$, and its i th element either as v_i . Then, let $last(v) = v_{|v|}$ be the last element in v . For $i \leq |v|$, let $v_{\geq i}$ be the suffix $v_i, \dots, v_{|v|}$ of v starting at v_i and $v_{< i}$ the (finite) prefix v_1, \dots, v_i of v starting at v_1 .

3.1 Interpreted Systems

We follow the presentation of interpreted systems as given by (Fagin et al., 1995). We will use them as a semantics for ATL^* as originally put forward by (Lomuscio and Raimondi, 2006), rather than concurrent game structures. Nonetheless, the two accounts are closely related (Goranko and Jamroga, 2004).

Definition 1 (Agent). Given a set Ag of indices for agents, an *agent* is a tuple $i = \langle L_i, Act_i, P_i, t_i \rangle$ such that

- L_i is the finite set of *local states*;
- Act_i is the finite set of *individual actions*;
- $P_i : L_i \rightarrow (2^{Act_i} \setminus \emptyset)$ is the *protocol function*;
- $t_i : L_i \times ACT \rightarrow L_i$ is the *local transition function*, where $ACT = Act_1 \times \dots \times Act_{|Ag|}$ is the set of *joint actions*, such that for every $l \in L_i$, $a \in ACT$, $t_i(l, a)$ is defined iff $a_i \in P_i(l)$.

By Def. 1 an agent i is situated in some local state $l \in L_i$, which represents the information she has about

$(M, s) \models q$	iff	$\Pi(s, q) = \text{tt}$
$(M, s) \models \neg\phi$	iff	$(M, s) \not\models \phi$
$(M, s) \models \phi \wedge \phi'$	iff	$(M, s) \models \phi$ and $(M, s) \models \phi'$
$(M, s) \models \langle\langle \Gamma \rangle\rangle \psi$	iff	for some joint strategy F_Γ , for all paths $p \in \text{out}(s, F_\Gamma)$, $(M, p) \models \psi$
$(M, p) \models \phi$	iff	$(M, p_1) \models \phi$
$(M, p) \models \neg\psi$	iff	$(M, p) \not\models \psi$
$(M, p) \models \psi \wedge \psi'$	iff	$(M, p) \models \psi$ and $(M, p) \models \psi'$
$(M, p) \models X\psi$	iff	$(M, p_{\geq 2}) \models \psi$
$(M, p) \models \psi U \psi'$	iff	for some $k \geq 1$, $(M, p_{\geq k}) \models \psi'$, and for all j , $1 \leq j < k$ implies $(M, p_{\geq j}) \models \psi$

Figure 1: Semantics of ATL*.

the current state of the system. At any state she can perform the actions in Act_i according to protocol P_i . A joint action brings about a change in the state of the agent, according to the local transition function t_i . Hereafter, with an abuse of notation, we identify an agent index i with the corresponding agent.

Given set Ag of agents, a *global state* $s \in \mathcal{G}$ is a tuple $\langle l_1, \dots, l_{|Ag|} \rangle$ of local states, one for each agent in Ag . Notice that an agent's protocol and transition function depend only on her local state, which might contain strictly less information than the global state. In this sense agents have *imperfect information* about the system. A history $h \in \mathcal{G}^+$ is a finite (non-empty) sequence of global states.

For every agent $i \in Ag$, we define an *indistinguishability relation* \sim_i between global states based on the identity of local states, that is, $s \sim_i s'$ iff $s_i = s'_i$ (Fagin et al., 1995). This indistinguishability relation is extended to histories in a synchronous, pointwise way, that is, histories $h, h' \in \mathcal{G}^+$ are *indistinguishable* for agent $i \in Ag$, or $h \sim_i h'$, iff (i) $|h| = |h'|$ and (ii) for every $j \leq |h|$, $h_j \sim_i h'_j$.

Definition 2 (IS). An *interpreted system* is a tuple $M = \langle Ag, s_0, T, \Pi \rangle$, where

- Ag is the set of agents;
- $s_0 \in \mathcal{G}$ is the (global) initial state;
- $T : \mathcal{G} \times ACT \rightarrow \mathcal{G}$ is the global transition function such that $s' = T(s, a)$ iff for every $i \in Ag$, $s'_i = t_i(s_i, a)$;
- $\Pi : \mathcal{G} \times AP \rightarrow \{\text{tt}, \text{ff}\}$ is the labelling function.

Intuitively, an interpreted system describes the interactions of a group Ag of agents, starting from the initial state s_0 , according to the transition function T . Notice that T is defined on state s for joint action a iff $a_i \in P_i(s_i)$ for every $i \in Ag$.

3.2 ATL

We make use of the Alternating-time Temporal Logic ATL^* (Alur et al., 2002) to reason about the strategic abilities of agents in interpreted systems.

Definition 3 (ATL*). State (ϕ) and path (ψ) formulas in ATL^* are defined as follows:

$$\begin{aligned} \phi &::= q \mid \neg\phi \mid \phi \wedge \phi' \mid \langle\langle \Gamma \rangle\rangle \psi \\ \psi &::= \phi \mid \neg\psi \mid \psi \wedge \psi' \mid X\psi \mid (\psi U \psi') \end{aligned}$$

where $q \in AP$ and $\Gamma \subseteq Ag$. 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 goal ψ ’. The meaning of *LTL* operators ‘next’ X and ‘until’ U is standard (Baier and Katoen, 2008). Operators ‘unavoidable’ $\llbracket \Gamma \rrbracket$, ‘eventually’ F , and ‘always’ G can be introduced as usual.

Formulas in the *ATL* fragment of ATL^* are obtained from Def. 3 by restricting path formulas ψ as follows:

$$\psi ::= X\phi \mid (\phi U \phi) \mid (\phi R \phi)$$

where ϕ is a state formula and R is the release operator¹.

Since the behaviour of agents in interpreted systems depends only on their local state, we assume agents employ *uniform strategies* (Jamroga and van der Hoek, 2004). That is, they perform the same action whenever they have the same information. This is formalised as follows.

Definition 4 (Uniform Strategy). A *uniform strategy* for agent $i \in Ag$ is a function $f_i : \mathcal{G}^+ \rightarrow Act_i$ such that for all histories $h, h' \in \mathcal{G}^+$, (i) $f_i(h) \in P_i(\text{last}(h)_i)$; and (ii) $h \sim_i h'$ implies $f_i(h) = f_i(h')$.

By Def. 4 any strategy for agent i has to return actions that are enabled for i . Also, whenever two histories are indistinguishable for agent i , then the same action is returned.

Given an IS M , a *path* p is an infinite sequence $s_1 s_2 \dots$ of global states. For a set $F_\Gamma = \{f_i \mid i \in \Gamma\}$ of

¹Notice that the release operator R can be defined in ATL^* as the dual of until U (indeed, it does not appear in the syntax of Def. 3), while it must be assumed as a primitive operator in *ATL*. We refer to (Laroussinie et al., 2008) for more details on this point.

strategies, one for each agent in coalition Γ , a path p is F_Γ -compatible iff for every $j > 0$, $p_{j+1} = T(p_j, a)$ for some joint action $a \in ACT$ such that for every $i \in \Gamma$, $a_i = f_i(p_1, \dots, p_j)$. Finally, let $out(s, F_\Gamma)$ be the set of all F_Γ -compatible paths starting with some s' such that $s' \sim_i s$ for some agent $i \in \Gamma$.

We can now assign a meaning to ATL^* formulas on interpreted systems.

Definition 5 (Satisfaction). The satisfaction relation \models for an IS M , state s , path p , and ATL^* formula ϕ is defined as in Figure 1.

We say that formula ϕ is *true* in an IS M , or $M \models \phi$, iff $(M, s_0) \models \phi$. Furthermore, in Def. 5 we use $\not\models$ to represent that it is not the case that \models .

We now state the model checking problem.

Definition 6 (Model Checking). The *model checking (MC) problem* concerns determining whether, given an IS M , ATL^* formula ϕ , truth value $v \in \{tt, ff\}$, it is the case that $(M \models \phi) = v$.

In the following section, we exemplify the formal machinery introduced so far with an example.

4 TRAIN GATE CONTROLLER SCENARIO

We consider a revised version of the Train Gate Controller by (Alur et al., 2002; Belardinelli et al., 2019b; Belardinelli and Malvone, 2020) in which there are two trains and a controller. The aim of the two trains is to pass a gate. To do this, they need to coordinate with the controller. The trains are initially placed outside the gate and to ask to go in the gate they need to do a request (action req). If the controller accepts the request (actions ac_1 and ac_2 , respectively), the train has the grant to pass through the gate. Note that, to perform the physical action of passing through the gate, the train has to select the action in . Then, it stays in the gate until it does the action out . What we want to show in this game is the fact that the trains need the accordance of the controller to achieve their objectives (*i.e.* to pass the gate).

More formally, this game can be represented as the IS $M = \langle Ag, s_0, T, \Pi \rangle$, such that:

- $Ag = \{Train_1, Train_2, Controller\}$;

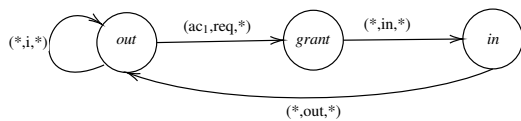


Figure 2: Local model for train 1.

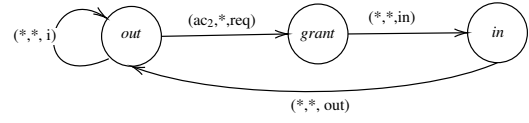


Figure 3: Local model for train 2.

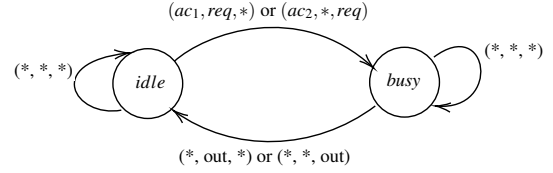


Figure 4: The local model for the controller.

- $Act_{Train_1} = Act_{Train_2} = \{req, in, out, i\}$, where by action req they do a request, by action in they go in the gate, by action out they go outside the gate, and by action i they do nothing. $Act_{Controller} = \{ac_1, ac_2, i\}$, where by action ac_j the Controller gives the access to train $j \in \{1, 2\}$, and by action i it does nothing.

The local model for $Train_1$ is given in Figure 2, the local model for $Train_2$ is given in Figure 3, and the local model of the Controller is given in Figure 4. The global initial state, the transition function, and the labelling function are given in Figure 5. In particular, each global state is represented as a rectangle where the tuple (l_c, l_{l_1}, l_{l_2}) includes the Controller's local state (l_c), the Train 1's local state (l_{l_1}), and the Train 2's local state (l_{l_2}). Furthermore, by the tuple of local states, we can consider as atomic propositions true in each state the names of the local states and, in accordance to them, define the labelling function. Notice that, in the figures, we denote any available action with the symbol $*$.

The property the Train 1 has a winning strategy to achieve the gate can be represented as follows:

$$\phi_1 = \langle\langle Train_1 \rangle\rangle F in_1$$

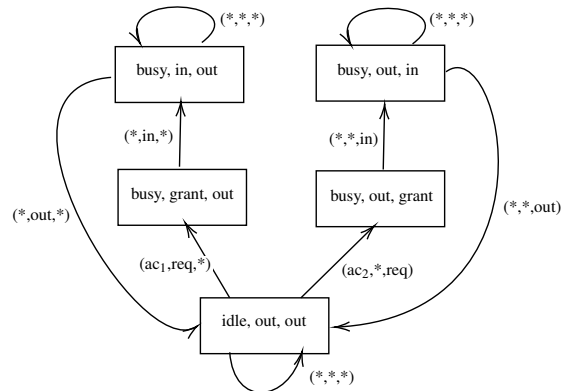


Figure 5: The interpreted systems IS, where $s_0 = (idle, out, out)$.

We observe that φ_1 is false since to make the property true the Train 1 needs the agreement of the Controller. By consequence, the property that can be satisfied is the following:

$$\varphi_1 = \langle\langle Train_1, Controller \rangle\rangle F in_1$$

5 THE APPROACH

Once an interpreted system M has been defined, we can unleash our approach. Starting from such game, we can verify for which coalitions of agents an ATL* formula φ is verified in M . Specifically, differently from standard ATL*, we do not want to explicitly state each Γ coalition in φ ; instead, we want to automatically generate such coalitions. Naturally, not all coalitions are always of interest; this of course depends on the domain of use. Thus, even though a coalition makes a formula φ satisfied by a model M , it does not necessarily mean such coalition is a good one (*i.e.*, a usable one).

γ Variables. To automatically generate the Γ coalitions in an ATL* formula φ , it is first necessary to have a way to uniquely identify each coalition inside the formula. In more detail, given an ATL* formula φ , we can annotate each strategic operator with a corresponding Γ variable. Just to make an example. Let us assume the ATL* formula φ is as follows: $\langle\langle a, b \rangle\rangle FG \langle\langle a, c \rangle\rangle Xp$; with $\{a, b\}$, and $\{a, c\}$ two coalitions, and p an atomic proposition. The formula becomes $\langle\langle \Gamma_1 \rangle\rangle FG \langle\langle \Gamma_2 \rangle\rangle Xp$, where Γ_1 and Γ_2 are two variables, which will be replaced by the automatically generated coalitions. Note that, in both strategic operators we may add the same Γ variable. In such a case, we would enforce to use the same coalition in the two strategic operators of φ .

In the following, we report the kind of guidelines we want to enforce over the coalitions. Such guidelines guide the coalitions' generation, so that all coalitions proposed by our approach both make φ satisfied in M and respect all the guidelines.

5.1 Guidelines

Two types of guidelines could be of interest in our investigation: the number of agents and how to group of agents.

Number of Agents in Coalition. The first group of guidelines concerns the size of the coalitions to generate. In more detail, it is possible to enforce the minimum (resp. maximum) number of agents per

coalition. This is very important, because it relates to possible real-world limitations. For instance, there might be scenarios where coalitions with less than n and more than m agents are not reasonable, because to create a coalition of less than n and more than m agents is too expensive for its gain. For this reason, a *min* (resp. *max*) constraint can be specified to rule out all Γ coalitions such that $min \leq |\Gamma|$ (resp. $|\Gamma| \leq max$). Considering our running example in Section 4, possible guidelines over the size of the coalitions could be *min* : 2 and *max* : 3. With such guidelines, we would enforce the generation of coalitions with at least two agents, but at most three. This could be guided by the fact that we know that no agent in isolation can achieve its own goals in the running example; while more than three agents would just be a waste of resources (from the viewpoint of the interactions that would be needed inside a coalition of agents).

Agents in the Same/Different Coalition. Another relevant group of guidelines concerns which agents can (or not) be in the same coalition. Again, this finds its motivation in real-world applications, where it is common to have limitations on how some agents can be grouped. For instance, considering that the agents are commonly situated in an environment, it may be possible that some of them are close (or not) to each other. For this reason, there might be interest in not having in the same coalition agents that are far from each other (for technological and practical reasons), while there might be interest in having in the same coalition agents that are local to each other. For this reason, a $[a \rightarrow \leftarrow b]$ (resp. $[a \leftarrow \rightarrow b]$) constraint can be specified to keep all Γ coalitions s.t. $a \in \Gamma \iff b \in \Gamma$ (resp. $a \in \Gamma \implies b \notin \Gamma$ and $b \in \Gamma \implies a \notin \Gamma$); where a and b can be any agent in Ag . By considering our running example, a possible constraint could be $[Train_1 \rightarrow \leftarrow Controller]$, where we enforce $Train_1$ and $Controller$ to be in coalition. For similar reasons, we may add the constraint $[Train_1 \leftarrow \rightarrow Train_2]$ and enforce the two trains to not be in coalition.

5.2 Verification

In the previous sections, we mainly focused on how the ATL* formulas can be modified, and how such modification can be guided by the user. Here, we move forward and present how our approach uses the pre-processing steps to perform the actual formal verification on MAS. Specifically, this is obtained through two algorithms. Let us explore them in detail.

Algorithm 1. It reports the steps required to generate a set of valid coalitions, *i.e.*, coalitions that re-

Algorithm 1: GenCoalitions(Ag, min, max, T, S).

```

1:  $\Gamma_{valid} = \emptyset$ 
2: for  $k \in [min, max]$  do
3:   for  $\Gamma \in \Gamma_k^{Ag}$  do
4:     if  $\exists [a_1 \rightarrow \leftarrow a_2] \in T : \{a_1, a_2\} \not\subseteq \Gamma \wedge \{a_1, a_2\} \cap \Gamma \neq \emptyset$  then continue
5:     if  $\exists [a_1 \leftarrow \rightarrow a_2] \in S : \{a_1, a_2\} \subseteq \Gamma$  then continue
6:     Add  $\Gamma$  to  $\Gamma_{valid}$ 
7: return  $\Gamma_{valid}$ 

```

Algorithm 2: MCMAS_{co}(M, ϕ, min, max, T, S).

```

1:  $Ag = GetAgents(M)$ 
2:  $\Gamma_{good} = \emptyset$ 
3:  $\Gamma_{valid} = GenCoalitions(Ag, min, max, T, S)$ 
4: for  $\Gamma \in \Gamma_{valid}$  do
5:   if  $M \models \phi^\Gamma$  then
6:     Add  $\Gamma$  to  $\Gamma_{good}$ 
7: return  $\Gamma_{good}$ 

```

spect the user's guidelines. Algorithm 1 takes in input the set of agents Ag , and the user's guidelines, such as the minimum/maximum number of agents to be in the coalitions, and the set of agents that have to (resp., cannot) stay in the same coalition T (resp., S). At line 1, the set of valid coalitions is initialised to the empty set. Then, at line 2, a value k is selected for any integer value between min and max (both included). After that, the algorithm loops over all possible values of k (lines 3-6); with k denoting the current size of the considered coalitions. Naturally, there are multiple k coalitions that can be formed over a set Ag of agents. In more detail, they correspond to all possible combinations of k agents taken from the set Ag ; this is expressed by the set Γ_k^{Ag} . For each of these coalitions, the algorithm checks whether the guidelines are followed or not. First, it checks if all agents that are required to be together in the coalition are as such (line 4). If for at least one couple $[a_1 \rightarrow \leftarrow a_2]$, we find only one agent in the coalition, then we skip to the next possible coalition to evaluate. In the same way, the algorithm checks for the agents that are not meant to be together (lines 5). This again is achieved by checking whether for some couple both the agents are in the coalition. If that is the case, then the algorithm moves on to the next coalition to evaluate. At the end of the algorithm, the set Γ_{valid} contains all coalitions respecting the user's guidelines.

Algorithm 2. It performs the actual verification considering all valid agents' coalitions. Algorithm 2 takes in input the model M , the *ATL* formula to verify ϕ , and the user's guidelines. At line 1, the set of

agents is extracted from M . These are the agents involved in the model. At line 2, the set of good coalitions is initialised to the empty set. By the end of the algorithm, such set will contain the coalitions that respect the user's guidelines and make ϕ satisfied in M . At line 3, Algorithm 1 is called. In this step, all valid coalitions respecting the user's guidelines are returned. After that, the algorithm loops over each of such valid coalitions. For each of them, the model checking is performed. In here, with ϕ^Γ we denote ϕ where the coalition has been replaced with the currently selected one (*i.e.*, Γ). If the model checking returns true, *i.e.*, model M satisfies formula ϕ^Γ , then Γ is added to the set of good coalitions Γ_{good} . This verification step is applied on each valid coalition. At the end of the algorithm, the set Γ_{good} is returned. Note that, in Algorithm 2, we only show the case with one strategic operator in ϕ , that is, only one Γ coalition is replaced in ϕ . We decided to do so in order to improve the readability of the procedure. However, in case multiple Γ coalitions are used, the same reasoning is followed, where for each one of them a set of valid coalitions is generated (using Algorithm 1). Then, instead of performing model checking only once (Algorithm 2, line 5), the algorithm would perform the latter for every possible permutation. For instance, if we had two coalitions Γ^1 and Γ^2 in ϕ (like in the example mentioned earlier in the paper), then we would consider all possible permutations of Γ_{valid}^1 and Γ_{valid}^2 .

Table 1: Number of good coalitions generated in our experiments. 1st column reports number of trains. 2nd column, no guidelines are given. 3rd to 5th column minimum number of agents per coalition is required. 6th to 8th column maximum number of agents per coalition is required. 9th to 12th columns guidelines on which agents can stay (or not) in coalition with.

N	-	≥ 2	≥ 3	≥ 4	≤ 2	≤ 3	≤ 4	$\exists!i.[T_i \rightarrow \leftarrow C]$	$\forall i.[T_i \rightarrow \leftarrow C]$	$\exists!(i, j).[T_i \leftrightarrow T_j]$	$\forall(i, j).[T_i \leftrightarrow T_j]$
2	3	3	1	0	2	3	3	2	1	2	2
4	15	15	11	5	4	10	14	8	1	11	4
6	63	63	57	42	6	21	41	32	1	47	6
8	255	255	247	219	8	36	92	128	1	191	8
10	1023	1023	1013	968	10	55	175	512	1	767	10

6 IMPLEMENTATION

A prototype of our approach has been implemented in Python². The prototype gets in input an interpreted system M , specified in terms of an ISPL file (the formalism supported by the MCMAS model checker), an ATL formula to verify ϕ , and generates all coalitions of agents which make $M \models \phi$. To understand the tool, first, we need to describe its pillar components.

The model checker we use is MCMAS (Lomuscio et al., 2015), which is the *de facto* standard model checker of strategic properties on MAS. MCMAS expects in input an interpreted system specified as an ISPL file. In such a file, the interpreted system is defined along with the formal property of interest to verify. From the viewpoint of a MCMAS user, our tool can be seen as an extension of MCMAS that allows the user to, not only perform the verification of ATL properties as usual, but to extract which coalitions of agents make such properties verified in the model.

Since MCMAS expects a fully instantiated ATL formula, in order to extract which coalitions of agents are good candidates, our tool performs a pre-processing step. In such step, as described previously in the paper, all coalitions which follow the user’s guidelines are generated (Algorithm 1) and tested on MCMAS (Algorithm 2). In each run, MCMAS returns the boolean result corresponding to the satisfaction of the ATL formula over the interpreted system. The coalitions for which MCMAS returns a positive verdict are then presented as output to the user.

The generation of all agents’ coalitions has been implemented in Python, as well as its enforcement over the ISPL file. In fact, for each coalition following the user’s guidelines, our tool updates the ISPL in the following way. Considering Algorithm 2, this step is implicitly performed in line 5, where the model checking is performed. However, at the implementation level, the actual verification through MCMAS requires to explicitly modify the ISPL file w.r.t. the Γ coalition of interest (*i.e.*, each coalition generated by Algorithm 1). To achieve this technical step, first,

the tool searches all occurrences of Γ coalitions in the ISPL file. This can be done by looking for the groups keyword (which is the one used in MCMAS to define the agents belonging to each coalition used in the ATL formula). After that, the tool replaces each coalition with a coalition following the user’s guidelines. Naturally, in case of multiple Γ coalitions in the ATL formula, all possible permutations of valid coalitions are considered. Once the ISPL file has been properly modified with valid coalitions, MCMAS is called to perform the actual verification.

7 EXPERIMENTS

We tested our tool over the train-gate controller scenario, on a machine with the following specifications: Intel(R) Core(TM) i7-7700HQ CPU @ 2.80GHz, 4 cores 8 threads, 16 GB RAM DDR4. We carried out various experiments on our running example. But, we have not only considered the case with two trains. Instead, we experimented with larger number of trains as well, to better evaluate our tool’s performance. Table 1 and Table 2 report the results we obtained.

Let us start with Table 1. It contains the number of coalitions we found through experiments. In more detail, the table is so structured. The first column reports the number of trains used in the experiments (from 2 to 10 trains). Then, the rest of the columns correspond to the results we get w.r.t. some specific guidelines. Going from left to right. First, we find the case where no guidelines have been passed to the tool. In such case, the tool reports all good coalitions, without any filter. This would correspond to a scenario where we would not have any sort of resource limitation and to group agents. Then, we have three different scenarios where we set the minimum number of agents per coalition (*i.e.*, we pass the *min* guideline). We do so for *min* equals to 2, 3, and 4. That is, we request only coalitions containing at least 2, 3, and 4 agents, respectively. Here, we can note how with *min* : 2, the number of coalitions does not change w.r.t. the case with no guidelines. This is due to the fact that, as

²<https://github.com/AngeloFerrando/mcmas-multi-coalitions>

Table 2: Execution time (in seconds) to generate the set of good coalitions in our experiments. 1st column reports number of trains. 2nd column, no guidelines are given. 3rd to 5th column minimum number of agents per coalition is required. 6th to 8th column maximum number of agents per coalition is required. 9th to 12th columns guidelines on which agents can stay (or not) in coalition with.

N	-	≥ 2	≥ 3	≥ 4	≤ 2	≤ 3	≤ 4	$\exists i.[T_i \rightarrow \leftarrow C]$	$\forall i.[T_i \rightarrow \leftarrow C]$	$\exists!(i, j).[T_i \leftarrow \rightarrow T_j]$	$\forall(i, j).[T_i \leftarrow \rightarrow T_j]$
2	0,06	0,03	0,01	0,00018	0,05	0,05	0,05	0,03	0,01	0,04	0,04
4	0,28	0,23	0,15	0,06	0,13	0,22	0,36	0,13	0,02	0,2	0,09
6	8,87	8,33	6,88	4,62	1,99	4,34	6,9	4,45	0,08	6,68	0,89
8	53,41	51,11	47,4	38,88	4,5	12,49	24,69	25,18	0,1	37,99	1,77
10	382,18	380,66	369,09	340,04	12,03	40,58	99,93	186,9	0,19	304,06	3,94

expected, no coalitions with less than 2 agents can satisfy the property of interest; which we remind being $\varphi = \langle\langle \Gamma \rangle\rangle Fin$. Instead, the other two cases have a fewer number of coalitions. This does not come as a surprise, since we are requesting only larger coalitions (we filter out all coalitions with 2 and 3 agents, respectively). After that, we find similar cases, where instead the *max* guideline is used. First, by enforcing the maximum number of agents in each coalition to be 2, then 3, and finally 4. W.r.t. the previous cases, here we can note how the choice of limiting the maximum number of agents in the coalitions is much more effective in reducing the number of good coalitions proposed. This again is reasonable, because we are filtering out the larger coalitions. Finally, we find the last four columns, which are focused on guidelines on which agents can stay with whom in the coalitions. First, we find the case where we request one single train to be in coalition with the controller. Note that, we do not decide such train *a priori*; it can be any of the available trains. In such case, the number of good coalitions is reduced, but not too much. This is due to the fact that requesting only one train to be in coalition with the controller is not a strong guideline (indeed, other trains can be in coalition as well). Then, the next case consists in requesting all trains to be in coalition with the controller. In this case, we obtain only one good coalition (no matter the number of trains). Since we are requesting all agents to be in coalition, this result is in line with the expectations. In the second to last column, we find a case where we request two trains not to be in coalition. As before, we are not interested in which trains, as long as only two are required not to be in coalition. As expected, this guideline does not affect much the number of good coalitions generated. Indeed, asking to not having just two trains in coalition does not filter out many viable alternatives. Last column presents the same scenario, but where all trains are requested to not be in the same coalition. So, each train cannot collaborate with any other train. This produces a number of coalitions equivalent to the number of trains used in the experiments. This again does not come as a surprise, since the only possible good coalitions are the

ones with one train and the controller (no other trains involved).

Moving on with Table 2, we find the same kind of experiments of Table 1. Nonetheless, instead of reporting the number of good coalitions generated, Table 2 reports the execution time required to extract such coalitions. The execution time comprises both the generation of the valid coalitions, and their verification through MCMAS. The columns are the same as in Table 1, but we can observe how much time the tool required to extract the coalitions. Naturally, we can observe that stronger are the guidelines, less is the execution time (since less are the valid coalitions that need to be verified in MCMAS). One important aspect to point out is that our experiments required less than 1 minute when considering the scenarios with at most 8 trains and less than 6 minutes (or so) for 10 trains. This is encouraging, since our approach handles even scenarios where the resulting model is far from being trivial (or small).

8 CONCLUSIONS AND FUTURE WORK

In this paper, we have presented an extension of MCMAS in which the users can characterise the coalitions in the strategy quantifiers. To do this, we have considered coalitions as variables of the problem. In particular, we have shown how to give the power to a user to handle two main features: the number of agents involved in the coalitions and how to create coalitions by considering who have to play together and who have to play against. This work is a first stone to develop a more generalised tool for verifying multi-agent systems.

As future work, we want to analyse the theoretical foundations of our practical idea. So, the first aim is to study how to extend the syntax and semantics of ATL to handle coalition variables. To do this, we can follow some ideas on graded modalities such as in (Malvone et al., 2018; Aminof et al., 2018). Furthermore, we want to study additional features to make the verification as useful and friendly as possible for

general users.

REFERENCES

- Alur, R., Henzinger, T., and Kupferman, O. (2002). Alternating-time temporal logic. *J. ACM*, 49(5):672–713.
- Aminof, B., Malvone, V., Murano, A., and Rubin, S. (2018). Graded modalities in strategy logic. *Inf. Comput.*, 261:634–649.
- Baier, C. and Katoen, J. P. (2008). *Principles of Model Checking (Representation and Mind Series)*. MIT press.
- Belardinelli, F., Jamroga, W., Kurpiewski, D., Malvone, V., and Murano, A. (2019a). Strategy logic with simple goals: Tractable reasoning about strategies. In Kraus, S., editor, *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJ-CAI 2019, Macao, China, August 10-16, 2019*, pages 88–94. ijcai.org.
- Belardinelli, F., Lomuscio, A., and Malvone, V. (2018). Approximating perfect recall when model checking strategic abilities. In *KR18*.
- Belardinelli, F., Lomuscio, A., and Malvone, V. (2019b). An abstraction-based method for verifying strategic properties in multi-agent systems with imperfect information. In *Proceedings of AAAI*.
- Belardinelli, F., Lomuscio, A., Malvone, V., and Yu, E. (2022). Approximating perfect recall when model checking strategic abilities: Theory and applications. *J. Artif. Intell. Res.*, 73:897–932.
- Belardinelli, F. and Malvone, V. (2020). A three-valued approach to strategic abilities under imperfect information. In Calvanese, D., Erdem, E., and Thielscher, M., editors, *Proceedings of the 17th International Conference on Principles of Knowledge Representation and Reasoning, KR 2020, Rhodes, Greece, September 12-18, 2020*, pages 89–98.
- Bloem, R., Chatterjee, K., Jacobs, S., and Könighofer, R. (2015). Assume-guarantee synthesis for concurrent reactive programs with partial information. In *TACAS*, pages 517–532.
- Dima, C. and Tiplea, F. (2011). Model-checking ATL under Imperfect Information and PerfectRecall Semantics is Undecidable. Technical report, arXiv.
- Fagin, R., Halpern, J. Y., Moses, Y., and Vardi, M. Y. (1995). *Reasoning about Knowledge*. MIT Press.
- Ferrando, A. and Malvone, V. (2021a). Strategy RV: A tool to approximate ATL model checking under imperfect information and perfect recall. In Dignum, F., Lomuscio, A., Endriss, U., and Nowé, A., editors, *AAMAS '21: 20th International Conference on Autonomous Agents and Multiagent Systems, Virtual Event, United Kingdom, May 3-7, 2021*, pages 1764–1766. ACM.
- Ferrando, A. and Malvone, V. (2021b). Towards the verification of strategic properties in multi-agent systems with imperfect information. *CoRR*, abs/2112.13621.
- Ferrando, A. and Malvone, V. (2022). Towards the combination of model checking and runtime verification on multi-agent systems. In Dignum, F., Mathieu, P., Corchado, J. M., and de la Prieta, F., editors, *Advances in Practical Applications of Agents, Multi-Agent Systems, and Complex Systems Simulation. The PAAMS Collection - 20th International Conference, PAAMS 2022, L'Aquila, Italy, July 13-15, 2022, Proceedings*, volume 13616 of *Lecture Notes in Computer Science*, pages 140–152. Springer.
- Goranko, V. and Jamroga, W. (2004). Comparing semantics for logics of multi-agent systems. *Synthese*, 139(2):241–280.
- Jamroga, W. and van der Hoek, W. (2004). Agents that know how to play. *Fund. Inf.*, 62:1–35.
- Kupferman, O. and Vardi, M. Y. (1997). Module checking revisited. In *CAV'97*, pages 36–47. Springer.
- Laroussinie, F., Markey, N., and Oreiby, G. (2008). On the expressiveness and complexity of ATL. *Logical Methods in Computer Science*, 4(2:7).
- Lomuscio, A., Qu, H., and Raimondi, F. (2015). MCMAS: A model checker for the verification of multi-agent systems. *Software Tools for Technology Transfer*.
- Lomuscio, A. and Raimondi, F. (2006). Model checking knowledge, strategies, and games in multi-agent systems. In *AAMAS 2006*, pages 161–168. ACM Press.
- Malvone, V., Mogavero, F., Murano, A., and Sorrentino, L. (2018). Reasoning about graded strategy quantifiers. *Inf. Comput.*, 259(3):390–411.
- Mogavero, F., Murano, A., Perelli, G., and Vardi, M. (2014). Reasoning about strategies: On the model-checking problem. *ACM Trans. Comp. Log.*, 15(4):34:1–34:47.
- Mogavero, F., Murano, A., Perelli, G., and Vardi, M. Y. (2017). Reasoning about strategies: on the satisfiability problem. *Log. Methods Comput. Sci.*, 13(1).
- Pnueli, A. and Rosner, R. (1990). Distributed reactive systems are hard to synthesize. In *FOCS*, pages 746–757.
- Reif, J. H. (1984). The complexity of two-player games of incomplete information. *J. Comput. Syst. Sci.*, 29(2):274–301.