We present Strategy RV, a tool that allows to approximate the verification of Alternating-time Temporal Logic under imperfect information and perfect recall, which is known to be undecidable, by using Runtime Verification. The tool uses an interface to enter the game model and the specifications and to provide results. We test Strategy RV in a variant of the Curiosity rover scenario and provide some experimental results.

KEYWORDS
ATL Model Checking; Imperfect Information; Perfect Recall Strategies; Runtime Verification

1 INTRODUCTION
A well-known formalism for reasoning about strategic behaviours in Multi-agent Systems (MAS) is Alternating-time Temporal Logic (ATL) [1]. An important feature of ATL is the computational complexity of its model checking, which is PTIME-complete under perfect information. However, MAS typically exhibit imperfect information and the model checking against ATL specifications under imperfect information and perfect recall is undecidable [8]. Given the importance of the imperfect information setting, even partial solutions to the problem can be useful. Previous approaches have either focused on an approximation to perfect information [4, 5] or developed notions of bounded recall [3]. The key idea of this contribution is that, by using runtime verification, MAS with imperfect information and perfect recall can be evaluated via perfect information and perfect recall (resp., imperfect information and imperfect recall) variants. This gives us an approximation procedure for ATL under imperfect information and perfect recall. Here, the sense of approximation is different from the one presented in previous papers [3–5]. In fact, while in the other works the procedures respond only in certain conditions, our procedure replies in all situations but the output is the verification of a strategic part (which in the worst case can be null, that is we can not strategically verify anything) and the remaining temporal part.


Algorithm 1: Our verification procedure

Data: a model M, a property \( \varphi \) and a variable choice
Result: the verification result

1. \( c_{IR} = \{\} \);
2. \( c_{IR} = \{\} \);
3. if choice = 0 then
   4. \( c_{IR} = \text{FindSubModelsWithPerfectInfo}(M, \varphi) \);
   5. end
4. else if choice = 1 then
   6. \( c_{IR} = \text{FindSubModelsWithImperfectRecall}(M, \varphi) \);
   7. end
8. else
   9. \( c_{IR} = \text{FindSubModelsWithPerfectInfo}(M, \varphi) \);
   10. \( c_{IR} = \text{FindSubModelsWithImperfectRecall}(M, \varphi) \);
11. end
12. return GenerateAndRunMonitors(M, \varphi, c_{IR} \cup c_{IR}());

As usual in the verification process, we denote imperfect recall with \( r \) perfect recall with \( R \), imperfect information with \( i \), and perfect information with \( I \).
The latter is implemented using LamaConv [14], a Java library that translates temporal logic expressions into equivalent automata and generates monitors out of these automata. For generating monitors, LamaConv uses the algorithm presented in [2]. Finally, the result of the runtime verification process is reported to the user.

3 THE TOOL
The algorithms presented previously have been implemented in Java\(^2\). The resulting tool implementing Algorithm 1 allows to extract all sub-models with perfect information (FindSubModelsWithPerfectInfo) and/or imperfect recall (FindSubModelsWithImperfectRecall) that satisfy a strategic objective from a model given in input. The extracted sub-models are then used by the tool to generate and execute the corresponding monitors. In more detail, the tool expects a model and a specification in input formatted as a Json file. This file is then parsed and an internal representation of the model is generated. In Fig. 1, we show the GUI of our tool to support this phase. On the left, the user inputs the Json model, and on the right, its graphical representation is visualised. After that, depending on the user’s choice, the proper procedure is called to extract the sub-models of interest (checkbox in Fig. 1). In both cases the verification of a sub-model against a sub-formula is achieved translating the sub-model into its equivalent ISPL (Interpreted Systems Programming Language) program, which then is verified by the model checker MCMAS\(^3\)[12]. The sub-models that satisfy this verification step are reported to the user (see Fig. 2). By clicking on a sub-model inside the list, the user can visualise the sub-model (on the right) and its corresponding verified formula. The entire manipulation, from parsing the model formatted in Json, to translating the latter to its equivalent ISPL program, has been performed by extending an existent Java library [6]; the rest of the tool is novel. Upon selection of a sub-model, the user can test a trace of events against the corresponding runtime monitor (GenerateAndRunMonitors is called). The latter is implemented using LamaConv [14], a Java library that translates temporal logic expressions into equivalent automata and generates monitors out of these automata. For generating monitors, LamaConv uses the algorithm presented in [2]. Finally, the result of the runtime verification process is reported to the user.

4 CURIOSITY ROVER SCENARIO
The Curiosity rover is one of the most complex systems successfully deployed in a planetary exploration mission to date. Differently from the original [13], in our setting the rover is equipped with decision making capabilities, which make it autonomous and without the need of human teleoperations. We simulate an inspection mission, where the Curiosity patrols a topological map of the surface of Mars. We begin with the deployment of the Curiosity and a start-up period where it initialises all three of its control modules. After the agent receives confirmation that the modules are ready, the mission may start. In one of the missions analysed, the rover starts behind the ship. So, its aim is to move from its position, makes a picture of a sample rock and then returns to the initial position. More in detail, it can decide to move left or right. Then, the rover can make the photo. At this point, the rover has to repeat the same moving action chosen the step before (left or right) to conclude the mission. The mission and the start-up period are modelled in Fig. 1. An important property to check should be if there exists a strategy for the rover such that sooner or later it can finish the mission. This property can be written in ATL and checked in the context of perfect recall strategies (see Fig. 2).

5 EXPERIMENTS
We tested our tool over some variants of the Curiosity rover 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 report the results obtained applying both FindSubModelsWithPerfectInfo and FindSubModelsWithImperfectRecall to the scenario presented in Section 4. The tool found 6 sub-models with perfect information in 0.6 [sec] and 452 sub-models with imperfect recall in 6.8 [sec]. The monitors generation for all the sub-models took 2.1 [sec] and 62.4 [sec], respectively. The tool has been applied to other scenarios as well, where it has been tested with different combinations of models and properties. The obtained results are in the same order of magnitude, and empirically confirmed our expectations. The extraction procedures applied to models of similar size took averagely less than 10 [sec], while the monitors generation required averagely ~60 [sec]. Note that, the latter was only a stress test, in fact our tool allows to generate and execute a monitor at a time to improve performance.

6 CONCLUSIONS
As remarked in the introduction one of the key issues in employing logics for strategic reasoning in the context of MAS is that their model checking problem is undecidable under perfect recall and imperfect information. However, this is one of the most natural setups in real applications. So, finding appropriate approximations remains an open problem. In this paper we presented Strategy RV, a tool that partially overcomes this difficulty by extracting sub-models with perfect information and/or imperfect recall that satisfy a strategic objective; it then uses runtime verification to check the remaining temporal objectives and to reach one of the sub-models so generated. We try to answer what is currently not possible to answer with an innovative technique that requires the use of two verification techniques already considered in the temporal context [7, 9–11] but which it has never been used in the strategic context.

---
\(^2\)The resulting tool can be found at https://github.com/AngeloFerrando/StrategyRV
\(^3\)https://vas.doc.ic.ac.uk/software/mcmas/
REFERENCES