Theory and Practice of Quantitative ATL

Angelo Ferrando[1][0000−0002−8711−4670], Giulia Luongo[2], Vadim Malvone[3][0000−0001−6138−4229], and Aniello Murano[2][0000−0003−4876−3448]

[1] University of Modena and Reggio Emilia, Italy
angelo.ferrando@unimore.it
[2] University of Naples Federico II, Italy
name.surname@unina.it
[3] LTCI, Télécom Paris, Institut Polytechnique de Paris, France
vadim.malvone@telecom-paris.fr

Abstract. In multi-agent system design and reasoning, strategy logics and formal verification play pivotal roles. Numerous logic formalisms have been introduced alongside the implementation of formal verification tools. Recently, also spurred by applications in neuro-symbolic AI, there has been a growing interest in modelling and reasoning about quantitative aspects of multi-agent systems as well. This paper introduces a quantitative strategic logic called $\mathsf{ATL}[\mathcal{F}]$, which extends the well known Alternating-time Temporal Logic with fuzzy functions. We have developed an algorithm to model check a multi-agent system with respect to an $\mathsf{ATL}[\mathcal{F}]$ formula, and implemented it within the VITAMIN tool. The paper also provides execution examples to show how the tool behaves and scales in practice.

1 Introduction

Multi-Agent Systems (MAS) represent a fundamental aspect in artificial intelligence, aiming to model complex systems of rational agents. Over the past two decades, significant efforts have been devoted to synthesis and verification methods for MAS. In the field of strategic reasoning, various formalisms and logics have been studied [3,13,42], considering both collaboration and antagonism among agents [9]. In this context, the Alternating-time Temporal Logic (ATL), introduced by Alur, Henzinger, and Kupferman more than two decades ago [3], is probably the most popular one. ATL formulas allow to express that a set of agents has a strategy to achieve a desired temporal goal, regardless of the strategies of the opponent agents. Since the introduction of ATL, the theoretical study of strategic logics has consistently been complemented by the development of tools designed to facilitate the practical application of formal aspects of strategic reasoning [20,21,32,37,40]. Upon examining these tools, it becomes evident that most efforts have been directed towards the qualitative aspects, namely verifying whether a formula is true or false. However, in the realm of multi-agent strategic reasoning, quantitative aspects play a crucial role. This is especially pertinent in today's context, where the integration of logics and

strategic reasoning in neuro-symbolic AI prominently features rewards as quantitative functions [39]. Consequently, it is imperative for logical frameworks to also encompass these quantitative aspects. Motivated by this perspective, classical temporal and strategic logics have been extended to incorporate quantitative aspects. Among these extensions, $\mathsf{LTL}[\mathcal{F}]$ [2] and $\mathsf{SL}[\mathcal{F}]$ [14] stand out, where, for the first time, functions are employed instead of atomic propositions to express quantitative aspects alongside the well known Linear-time Temporal logic (LTL) [44] and Strategy Logic (SL) [42], respectively.

Our Contribution. In this work, we introduce a quantitative extension of ATL with fuzzy functions ($\mathsf{ATL}[\mathcal{F}]$), an algorithm for the related verification question, and its implementation in a tool. Regarding the latter, we have chosen to build on VITAMIN [28], a very recently released tool for strategic reasoning that is garnering particular interest in the community. Our contribution is twofold: on the one hand, we introduce $\mathsf{ATL}[\mathcal{F}]$ along with its syntax, semantics, and model checking algorithm, and on the other hand, we implement for the first time a tool to deal with a quantitative extension of ATL by means of fuzzy functions. It is worth noting that $\mathsf{ATL}[\mathcal{F}]$ includes ATL as a special case. Consequently, this work is also the first one to deal with the implementation of ATL in VITAMIN. Besides defining the algorithm for model checking $\mathsf{ATL}[\mathcal{F}]$ and having implemented it, along the paper we have tested our solution in practice. However, as our tool is the first one able to deal with quantitative aspects for ATL, we have been unable to make a proper comparison with other tools.

1.1 Related Work

Algorithms and tools for model checking strategic abilities [3,42] have been developed for over 20 years [4,19,21,24,30,32,37,40]. Among the others, *MCMAS* [40] is recognized as one of the most widely used tools for the strategic verification of MAS, primarily due to being one of the earliest tools developed, which served as a foundational proof-of-concept for researchers. In order to speed up the verification of some specific goals, the tool *STV* has been developed [36], which treats the specification target in a predetermined way. Both MCMAS and STV, lack modularity and usability and require a strong background in formal methods, making them challenging for non-expert users to employ them. To overcome these limitations, the tool VITAMIN [28] has been released very recently. Notably, MCMAS and STV have been extended in various directions, including imperfect information [20], but never alongside quantitative aspects; this is the case for the tool VITAMIN as well.

From a theoretical point of view, the quantitative dimension has been explored in various logics, such as $\mathsf{LTL}[\mathcal{F}]$ [2] and $\mathsf{SL}[\mathcal{F}]$ [13]. Other quantitative extensions of ATL and SL have been explored in the context of model checking. These include timed [16,31], multi-valued [33,10], weighted [8,17,38,45], resources [43,18], natural strategies [34,35,11], and probabilistic [12,25,7,22]. Quantitative extensions of LTL have also been proposed, such as averaging [15] and discounting [1,26,41].

The concept of quantitativeness, which involves the possibility of having a value that is not simply 0 or 1, but rather a range between these, corresponds to aiming for the "best" outcome achievable, and certainly has also connections with best effort strategies [5,6].

2 Weighted Alternating-time Temporal Logic

In this section, we introduce $\mathsf{ATL}[\mathcal{F}]$, the quantitative extension of ATL with fuzzy functions, detailing its syntax and semantics. Before delving into the presentation of $\mathsf{ATL}[\mathcal{F}]$, we introduce some notation that will be used throughout the paper. Given a set U, $\overline{U}$ denotes its complement. We denote the length of a tuple v as $|v|$, its j-th element as v_j, and its last element $v_{|v|}$ as $last(v)$. For $j \leq |v|$, let $v_{\geq j}$ be the suffix $v_j, ..., v_{|v|}$ of v starting from v_j and $v_{\leq j}$ the prefix $v_1, ..., v_j$ of v.

We start by introducing Weighted Concurrent Game Structures, which will be used to interpret $\mathsf{ATL}[\mathcal{F}]$ formulas and thus to define the model checking problem for $\mathsf{ATL}[\mathcal{F}]$.

2.1 Weighted Concurrent Game Structures

A Concurrent Game Structure (CGS) [3] is a mathematical model used to represent and analyse MAS where agents (or players) can interact with each other and their environment. Unlike Kripke structures [23], which are used to model closed systems, CGSs are particularly useful for modeling open systems that involve interaction between multiple players. In a CGS, the system is represented as a graph with nodes and edges, where the nodes represent the states of the system and the edges represent the possible transitions between states. Each state can have labels that represent the properties of the system in that particular state, while each transition is associated with a set of actions that can be performed by the players. These actions are typically non-deterministic, meaning that the players can choose from a set of possible actions, and the outcome of the game depends on the choices made by all players.

Since in this work we focus on the quantitative verification of MAS, other than CGSs, we are interested in analysing their extended version, called Weighted Concurrent Game Structures (wCGS) [13]. A wCGS models the concurrent computation where agents simultaneously choose their actions. In a wCGS, atomic propositions describe features of the game. Therefore, for each state s, and for each atomic proposition p, a weight to p in s is assigned.

Definition 1. *A wCGS is a tuple* $\mathcal{G} = (\mathrm{Ag}, \mathrm{Ap}, \{\mathrm{Act}_i\}_{i\in\mathrm{Ag}}, S, s_I, \ell, d, o)$ *where:*

- Ag *is a nonempty finite set of agents.*
- Ap *is a nonempty finite set of atomic propositions (atoms).*
- *For every* $i \in \mathrm{Ag}$, Act_i *is a nonempty finite set of* actions. *Let* $Act = \bigcup_{i\in\mathrm{Ag}} Act_i$ *be the set of all actions, and* $ACT = \prod_{i\in\mathrm{Ag}} Act_i$ *the set of all joint actions.*

- S *is a finite set of states.*
- $s_I \in S$ *is an initial state.*
- $\ell : S \times \mathrm{Ap} \to [0,1]$ *is a weight function.*
- $d : \mathrm{Ag} \times S \to 2^{\mathrm{Act}}$ *is an availability function that defines a non-empty set of actions available to agents at each state.*
- o *is a transition function which assigns the outcome state* $s' = o(s, \boldsymbol{c})$ *to each state* s *and tuple of actions* $\boldsymbol{c} \in \prod_{a \in \mathrm{Ag}} d(a, s)$ *that can be executed by the agents in* s*.*

In the rest of the paper, when a joint action $\boldsymbol{c}$ and a coalition of agents A are given, we denote $\boldsymbol{c}_A$ as the projection of $\boldsymbol{c}$ where only the actions of agents in A are considered; while we denote $\boldsymbol{c}_{\mathrm{Ag}\setminus A}$ as the projection of $\boldsymbol{c}$ where only the actions of agents not in A are considered.

Example 1. We present a model inspired by the Drone Battle scenario in [13]. In this model, there are two agents within a two-dimensional space: the carrier (c) and the villain (v). The carrier's objective is to deliver an artifact to the rescue point (represented by the grey area in Figure 1) while maintaining a certain distance from the villain. We simplify the model from [13], by restricting the space to a range between 0 and 1, where both agents can only move in increments of $0.\overline{3}$. Consequently, coordinates along the x and y axes are 0, $0.\overline{3}$, $0.\overline{6}$, and 1. For the sake of illustration, we restrict agents to a few actions: the carrier can move **up (u)**, **right (r)**, and **left (l)**, while the villain can move **down (d)**, **right (r)**, and **left (l)**. The wCGS in Figure 2 illustrates all agents' moves. We have a total of 21 states describing the possible configurations. Transitions between states occur based on the actions chosen by the two agents. Each state has two atomic propositions: **dist**, representing the distance between the carrier and the villain, and **safe**, indicating whether the carrier is located at the rescue point. When the carrier is at the rescue point, the atomic proposition evaluates to 1 and in the model diagram, it is coloured in grey.

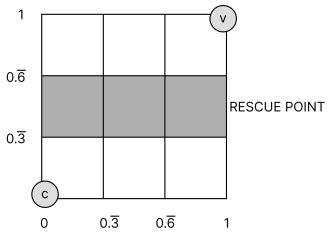

Fig. 1: Drone Battle scenario.

2.2 ATL[$\mathcal{F}$]

We now introduce ATL[$\mathcal{F}$], an extension of ATL with fuzzy functions, which allows for the representation of weighted atomic propositions. By means of this extension, it is possible to reason strategically about quantitative aspects of MAS within the framework of strategic logics.

Syntax. We start by providing the syntax of ATL[$\mathcal{F}$]. We build our logic on ATL, as it is defined in [3].

Definition 2. *Formulas φ in $\mathsf{ATL}[\mathcal{F}]$ are defined as follows:*

$$\varphi ::= p \mid f[\varphi, ..., \varphi] \mid \langle\langle \Gamma \rangle\rangle \mathbf{X}\varphi \mid \langle\langle \Gamma \rangle\rangle \mathbf{G}\varphi \mid \langle\langle \Gamma \rangle\rangle \varphi \mathbf{U} \varphi$$

where $p \in \mathrm{Ap}$, $\Gamma \in 2^{\mathrm{Ag}}$, and $f \in \mathcal{F}$, where $\mathcal{F} \subseteq \{f : [0,1]^m \to [0,1] \mid m \in \mathbb{N}\}$ represents a set of computable functions.

As for ATL, $\mathsf{ATL}[\mathcal{F}]$ makes use of the strategic operator $\langle\langle \cdot \rangle\rangle$, which can predicate over a temporal logic formula having one single temporal operator at the time. Given a coalition of agents Γ, $\langle\langle \Gamma \rangle\rangle \varphi$ represents the maximum value of φ the agents in Γ can ensure, regardless of how the other players act. Furthermore, $\mathsf{ATL}[\mathcal{F}]$ utilises the standard temporal operators $\mathbf{X}$, $\mathbf{G}$, and $\mathbf{U}$, representing "next", "globally", and "until", respectively. Additionally, the meaning of $f[\varphi_1, ..., \varphi_2]$ depends on the function $f \in \mathcal{F}$.

For the sake of simplicity, in this paper we focus on a specific set of $\mathcal{F}$ functions, which includes the $min\{x, y\}$, $max\{x, y\}$, and $1 - x$ functions. These functions serve as the standard quantitative equivalents of the $\wedge$, $\vee$, and $\neg$ operators, commonly referred to as the Zadeh operators in fuzzy logics. Intuitively, the value of the formula $\varphi_1 \vee \varphi_2$ is the maximum value of the values of the two subformulas φ_1 and φ_2; analogously, $\varphi_1 \wedge \varphi_2$ takes the minimal value of the values of the two subformulas φ_1 and φ_2; finally the value of $\neg\varphi$ is 1 minus the value of φ. Thus, the implication $\varphi_1 \to \varphi_2$ takes the maximum value between that of φ_2 and 1 minus the value of φ_1.

Observation. In a Boolean setting, we assume that the values of the atomic propositions are in $\{0, 1\}$ (0 represents $\bot$ whereas 1 represents $\top$), and so are the output values of functions in $\mathcal{F}$. In such a setting, we can observe that the min, max, and $1 - x$ functions take the exact same meaning of $\wedge$, $\vee$, and $\neg$.

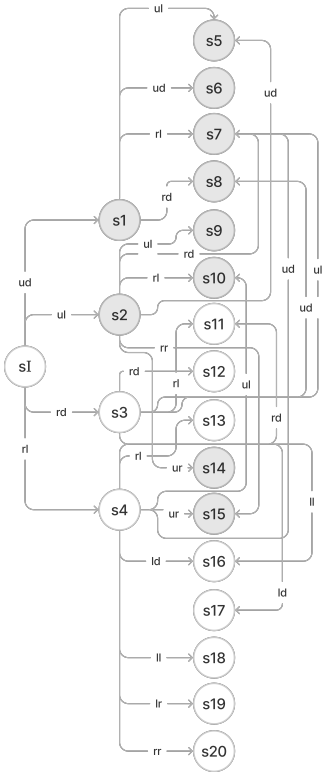

Fig. 2: Drone Battle model. Note that, all the states that have not a successor contains an implicit loop.

Semantics. $\mathsf{ATL}[\mathcal{F}]$ formulas are interpreted on wCGS, where atomic propositions can take values in the range of [0, 1].

In order to present $\mathsf{ATL}[\mathcal{F}]$ semantics, we need to introduce some notions. A *history* $h \in S^+$ is a finite (non-empty) sequence of states in a wCGS. A *perfect recall strategy* is a conditional plan that assigns an action for every history. The formal definition follows.

Definition 3. *A perfect recall strategy for agent* $i \in \mathrm{Ag}$ *is a function* $\sigma_i : S^+ \to \mathrm{Act}_i$ *such that for each history* $h \in S^+$*, we have that* $\sigma_i(h) \in d(i, last(h))$.

According to Definition 3, any strategy for agent i must return actions that are enabled for i. Note that, *memoryless* (imperfect recall) strategies can be obtained by considering S as the domain of σ_i, i.e., $\sigma_i : S \to \mathrm{Act}_i$.

In a wCGS $\mathcal{G}$, a *path* π represents an infinite sequence of states. The set of paths over S is denoted by S^ω.

For a joint strategy σ_Γ, consisting of one strategy for each agent in coalition Γ, a path π is σ_Γ*-compatible* if, for every $j \geq 1$, $\pi_{j+1} = o(\pi_j, \boldsymbol{c})$ for some joint action $\boldsymbol{c}$ such that for every $i \in \Gamma$, $c_i = \sigma_i(\pi_{\leq j})$, and for every $i \in \overline{\Gamma}$, $c_i \in d(i, \pi_j)$.

The set of all σ_Γ*-compatible* paths from s is denoted by $\mathrm{out}(s, \sigma_\Gamma)$. While the set of all possible *joint strategies* for a coalition Γ is denoted by Σ_Γ.

Definition 4. *For a given* $\mathsf{ATL}[\mathcal{F}]$ *formula* φ*, a wCGS* $\mathcal{G}$*, and a path* π*, the satisfaction value of* φ *on* π *in* $\mathcal{G}$ *is denoted* $[\![\varphi]\!]^{\mathcal{G}}(\pi)$ *and defined recursively as:*

$$
\begin{aligned}
[\![p]\!]^{\mathcal{G}}(\pi) &= \ell(\pi_1, p) \\
[\![f[\varphi_1, ..., \varphi_n]]\!]^{\mathcal{G}}(\pi) &= f[[\![\varphi_1]\!]^{\mathcal{G}}(\pi), ..., [\![\varphi_n]\!]^{\mathcal{G}}(\pi)] \\
[\![\langle\!\langle \Gamma \rangle\!\rangle \mathbf{X}\varphi]\!]^{\mathcal{G}}(\pi) &= \max_{\sigma_\Gamma \in \Sigma_\Gamma} \Big(\min_{\pi' \in \mathrm{out}(\pi_1, \sigma_\Gamma)} ([\![\varphi]\!]^{\mathcal{G}}(\pi'_{\geq 2})) \Big) \\
[\![\langle\!\langle \Gamma \rangle\!\rangle \mathbf{G}\varphi]\!]^{\mathcal{G}}(\pi) &= \max_{\sigma_\Gamma \in \Sigma_\Gamma} \Big(\min_{\pi' \in \mathrm{out}(\pi_1, \sigma_\Gamma)} (\min_j([\![\varphi]\!]^{\mathcal{G}}(\pi'_{\geq j}))) \Big) \\
[\![\langle\!\langle \Gamma \rangle\!\rangle \varphi_1 \mathbf{U} \varphi_2]\!]^{\mathcal{G}}(\pi) &= \max_{\sigma_\Gamma \in \Sigma_\Gamma} \Big(\min_{\pi' \in \mathrm{out}(\pi_1, \sigma_\Gamma)} (\max_j(\min_j([\![\varphi_2]\!]^{\mathcal{G}}(\pi'_{\geq j}), \min_{i<j}([\![\varphi_1]\!]^{\mathcal{G}}(\pi'_{\geq i}))))) \Big)
\end{aligned}
$$

Example 2. Consider again the Drone Battle scenario as reported in Example 1. Consider now the following $\mathsf{ATL}[\mathcal{F}]$ formula:

$$\varphi_{\text{rescue}} = \langle\!\langle c \rangle\!\rangle (\text{dist} \geq 0.5)\ \mathbf{U}\ \text{safe}^4$$

This formula means that there exists a strategy for the *carrier* such that she can reach a safe state keeping always the *villain* at a distance of at least 0.5.

3 $\mathsf{ATL}[\mathcal{F}]$ Model Checking

We now present the model checking algorithm for $\mathsf{ATL}[\mathcal{F}]$. The algorithm is reported in Algorithm 1. This algorithm takes inspiration from the one for ATL [3], modified opportunely to handle the quantitative semantics of $\mathsf{ATL}[\mathcal{F}]$. Specifically, with respect to the ATL algorithm, two main aspects need to be fully revisited: the preimage construction and, the generation and handling of states. In Algorithm 2, the preimage function, called $Pre[\mathcal{F}]$, is extended to evaluate the strategies in a quantitative way. Concerning instead the states, as we see in the following, Algorithm 1 is built upon set of states where at each state a quantitative value is associated (denoting the corresponding current quantitative satisfaction over the $\mathsf{ATL}[\mathcal{F}]$ formula under analysis).

[4] The subformula $(\text{dist} \geq 0.5)$ is syntactic sugar for $(\geq(\text{dist}, 0.5))$, where $\geq \in \mathcal{F}$.

However, some modifications are required. First of all, a distinct version of the preimage function, that we call $Pre[\mathcal{F}]$, is added as the primary modification of the standard ATL model checking algorithm. Furthermore, the way of generating the states satisfying an atomic proposition is also different.

Algorithm 1 ATL$[\mathcal{F}]$ model checking

for each φ' **in** $Sub(\varphi)$ **do**
  case $\varphi' = p : [\varphi'] \coloneqq Reg[\mathcal{F}](p)$
  case $\varphi' = f[\theta_1, ..., \theta_n] : [\varphi'] \coloneqq f[[\theta_1], ..., [\theta_n]]$
  case $\varphi' = \langle\langle\Gamma\rangle\rangle \mathbf{X}\theta : [\varphi'] \coloneqq Pre[\mathcal{F}](\Gamma, [\theta])$
  case $\varphi' = \langle\langle\Gamma\rangle\rangle \mathbf{G}\theta :$
    $\rho \coloneqq \{\langle s, 1\rangle \mid s \in S\}$
    $\tau \coloneqq [\theta]$
    while $\rho \not\sqsubseteq \tau$ **do**
      $\rho \coloneqq \tau$
      $\tau \coloneqq Pre[\mathcal{F}](\Gamma, \rho) \cap [\theta]$
    $[\varphi'] \coloneqq \rho$
  case $\varphi' = \langle\langle\Gamma\rangle\rangle \theta_1 \mathbf{U} \theta_2 :$
    $\rho \coloneqq \{\langle s, 0\rangle \mid s \in S\}$
    $\tau \coloneqq [\theta_2]$
    while $\tau \not\sqsubseteq \rho$ **do**
      $\rho \coloneqq \rho \cup \tau$
      $\tau \coloneqq Pre[\mathcal{F}](\Gamma, \rho) \cap [\theta_1]$
    $[\varphi'] \coloneqq \rho$
end for
return $[\varphi]$

Before proceeding with the definition of the new $Pre[\mathcal{F}]$ algorithm (along with its auxiliary procedures), we wish to linger on the definition of the $Reg[\mathcal{F}]$ function. Specifically, since ATL$[\mathcal{F}]$ formulas are verified on wCGS, the atomic propositions are quantified in the states of the model, the $Reg[\mathcal{F}]$ function returns a set of tuples.

Definition 5. *Given an atomic proposition $p \in \mathrm{Ap}$, we define $Reg[\mathcal{F}](p)$ as the set $\{\langle s, v\rangle \mid s \in S \wedge v = \ell(s, p)\}$. That is, the set containing tuples, where each tuple consists of a state s in the wCGS, along with its associated value of p in s.*

Note that, because of $Reg[\mathcal{F}]$, Algorithm 1 does not work on sets of states, but on sets of tuples, where at each state is associated a value. Thus, the set operations in the algorithm need further explanation. Specifically, assuming two set of tuples S_1 and S_2 as previously defined, we have that $S_1 \subseteq S_2$, if and only if, for all $s \in S$, s.t. $\langle s, v_1\rangle \in S_1$ and $\langle s, v_2\rangle \in S_2$, we have that $v_1 \leq v_2$. Furthermore, for the same reason we extend the notion of intersection, that is, given two sets of tuples, we have that $S_1 \cap S_2 = \{\langle s, v\rangle \mid \langle s, v_1\rangle \in S_1 \wedge \langle s, v_2\rangle \in S_2 \wedge v = min(v_1, v_2)\}$. Note that, the union is obtained in the same way, but selecting the maximum value amongst v_1 and v_2 (*i.e.*, $v = max(v_1, v_2)$).

Observation. Differently from the standard ATL model checking algorithm, in the ATL$[\mathcal{F}]$ algorithm, all states are taken into account at all times. That is, $\langle s, v\rangle \in Pre[\mathcal{F}]$ if and only if $s \in S$.

Let us now provide a detailed explanation of the algorithms that extend the standard model checking of ATL to achieve the model checking of ATL$[\mathcal{F}]$.

In Algorithm 2, the $Pre[\mathcal{F}]$ function is reported. Such a function – as in the standard Pre function in ATL model checking – expects a set of agents Γ and a set of tuples (state-value) ρ in input. Algorithm 2 first initialises a set

Algorithm 2 $Pre[\mathcal{F}](\Gamma, \rho)$

1: $res = \{\langle s, 0\rangle \mid s \in S\}$
2: **for each** $s \in S$ **do**
3: $\quad maxStrategyValue = evaluateMaxStrategy(\Gamma, s, \rho)$
4: $\quad res = res \cup \{\langle s, maxStrategyValue\rangle\}$
5: **end for**
6: **return** res

containing the final result of $Pre[\mathcal{F}]$ to a set of tuples where to each state in S is assigned value 0 (line 1). After that, the algorithm iterates on all the states s in S (lines 2–5), where for each $s \in S$, the maximum value for the strategy for the agents in Γ is computed (line 3). This computation is performed in Algorithm 3, discussed in the following paragraphs. However, at high level, the value returned by Algorithm 3 corresponds to the maximum value the coalition Γ of agents can obtain, amongst the minimum values the opponent agents $\text{Ag} \setminus \Gamma$ can enforce. This follows the semantics of ATL$[\mathcal{F}]$, given in Section 2.2. Such value, along with state s, is then added to the result set res (line 4), exploiting our redefined union operator (see before). This preserves the fact that Algorithm 1 manipulates sets of tuples rather than sets of states. The algorithm then terminates by returning the res set (line 6).

Moving on, Algorithm 3 is tasked with the goal of evaluating all possible outcome trees. To better understand, let us first define what an outcome tree is.

Definition 6. *Given a state $s \in S$, a coalition of agents $\Gamma \subseteq \text{Ag}$, and a projected joint action $\boldsymbol{c}_\Gamma$, we have that a tree t_s for $\boldsymbol{c}_\Gamma$ is an outcome tree, if and only if, $\boldsymbol{c}_\Gamma$ is the root of t_s and for all $\boldsymbol{c}' \in \prod_{a \in \text{Ag}} d(a, s)$ s.t. $\boldsymbol{c}'_\Gamma = \boldsymbol{c}_\Gamma$, we have a leaf node in t_s that is children of the root node $\boldsymbol{c}_\Gamma$ and contains the tuple $\langle \boldsymbol{c}'_{\text{Ag}\setminus\Gamma}, s'\rangle$, with $s' \in S$ and $s' \in o(s, \boldsymbol{c}')$.*

Algorithm 3 $evaluateMaxStrategy(\Gamma, s, \rho)$

1: $minValuesTree = \emptyset$
2: $trees = createNextMoveTrees(\Gamma, s)$
3: **for each** $node(\boldsymbol{c}_\Gamma) \in trees$ **do**
4: $\quad children = getChildren(node(\boldsymbol{c}_\Gamma))$
5: $\quad valuesTree = \emptyset$
6: $\quad$ **for each** $\langle\langle \boldsymbol{c}_{\text{Ag}\setminus\Gamma}, s'\rangle \in children$ **do**
7: $\quad\quad$ get $\langle s', value\rangle$ from ρ
8: $\quad\quad valuesTree = valuesTree \cup value$
9: $\quad$ **end for**
10: $\quad minimum = min(valuesTree)$
11: $\quad minValuesTree = minValuesTree \cup minimum$
12: **end for**
13: **return** $max(minValuesTree)$

Algorithm 3 expects in input the set of agents in coalition, the state that has been currently evaluated, and the set of tuples of states – along with their current

maximum value associated – resulting from previous recursive calls of the $Pre[\mathcal{F}]$ function.

Algorithm 3 begins by initialising an empty set, which will serve as a collector of values associated with the found outcome trees (line 1). Then, Algorithm 4 is invoked (line 2), and its result is stored in *trees*. This set contains all the possible trees, according to Definition 6, where the root is a joint action selected by the coalition Γ, and the children are all the possible joint actions that can be selected by the opposing agents (that is, $\text{Ag} \setminus \Gamma$). Note that, as it is presented in Algorithm 4, each opponents' action is attached to a state s'; such a state denotes the next state to be visited in case the opponents' action is selected.

After extracting these trees, Algorithm 3 iterates over each of the obtained trees (lines 3–12). For each tree, all the child nodes are extracted (line 4) and evaluated (lines 6–9). These are tuples comprising the joint action that can be selected by the opposing agents, along with the associated landing state (*i.e.*, the next state to visit if such action is selected). In this step, the algorithm updates a set of values by adding, for each child of the selected tree, the value associated with the state in ρ. That is, it retrieves the current maximum value associated with s' in ρ. Afterwards, the minimum value amongst the evaluated values is determined (line 10) and added to the other values resulting from the alternative trees (line 11). The algorithm concludes by selecting the maximum value amongst the minimum values (line 13), representing the fact that when the coalition of agents Γ can choose amongst different options, they will opt for the one that produces the highest outcome.

Finally, the last auxiliary algorithm to define is Algorithm 4, where the action trees are produced to be used in Algorithm 3. Algorithm 4 expects in input the set of agents belonging to the coalition and the state of the wCGS currently evaluated. It starts by initialising the set of found trees to the empty set (line 1). Then, it loops over all the possible joint actions that can be performed in s. This can be extracted through the $d(a, s)$ function of the wCGS that denotes which actions an agent a can perform in a state s (lines 2–11). After that, for each joint action so selected ($\boldsymbol{c}$), the algorithm creates a node n (line 3) containing the opposing joint action ($\boldsymbol{c}_{\text{Ag}\setminus\Gamma}$) along with the state that can be reached by performing the whole joint action ($o(s, \boldsymbol{c})$). Then, the algorithm checks whether this is the first time a tree for the coalition action $\boldsymbol{c}_\Gamma$ is encountered. If that is the case (lines 4–7), then a tree is initialised with $\boldsymbol{c}_\Gamma$ as root (line 5), n is added as unique child of $\boldsymbol{c}_\Gamma$ (line 6), and the resulting tree is added along all the other trees found in the execution of Algorithm 4 (line 7). Otherwise (lines 8–10), since the tree is already present in the set of trees, n is simply added as a new child of $\boldsymbol{c}_\Gamma$ (line 9). Once all the possible joint actions in s are evaluated, the algorithm concludes by returning the generated set of trees (line 12).

We conclude this section by providing the complexity result of our procedure.

Theorem 1. *Given an* $\mathsf{ATL}[\mathcal{F}]$ *formula* φ *and a wCGS* $\mathcal{G}$*, Algorithm 1 terminates in polynomial time with respect to the size of* $\mathcal{G}$ *and the length of* φ*.*

Proof. To prove the termination of Algorithm 1 we first prove the termination of the auxiliary algorithms it uses. Algorithm 2 loops over the states of $\mathcal{G}$ and for

Algorithm 4 $createNextMoveTrees(\Gamma, s)$

```
1: trees = ∅
2: for each c ∈ ∏_{a∈Ag} d(a, s) do
3:     n = node(⟨c_{Ag\Γ}, o(s, c)⟩)
4:     if root(c_Γ) ∉ trees then
5:         tree = root(c_Γ)
6:         add n as child of tree
7:         trees = trees ∪ tree
8:     else
9:         add n as child of root(c_Γ)
10:    end if
11: end for
12: return trees
```

each state it calls Algorithm 3, which then calls Algorithm 4 to synthesise the action trees needed for the evaluation. The latter trivially terminates in polynomial time with respect to the number of joint actions available in the current state. Consequently, this makes Algorithm 3 also polynomial with respect to the number of joint actions available in the state under analysis (since it iterates over the resulting trees). Thus, going back to Algorithm 2, we can conclude it terminates in polynomial time with respect to the size of $\mathcal{G}$ (since for each state it analyses all transitions available in that state). Now that we have concluded the analysis of all auxiliary algorithms, we can resume the analysis of Algorithm 1; specifically, it loops over the subformulas φ' of φ, thus, to prove Algorithm 1 is polynomial, each case of Algorithm 1 has to be at most polynomial.

- Case $\varphi' = p$: Algorithm 1 calls $Reg[\mathcal{F}]$ and the latter is trivially polynomial on the size of $\mathcal{G}$.
- Case $\varphi' = f[\theta_1, ..., \theta_n]$: Since each $[\theta_i]$ for $1 \leq i \leq n$ was computed in a previous iteration and we assume f to be at most polynomial[5], then the complexity of this case follows.
- Case $\varphi' = \langle\langle\Gamma\rangle\rangle \mathbf{X}\theta$: The preimage function (Algorithm 2) is called. We proved the latter is polynomial with respect to the size of $\mathcal{G}$, so we may conclude.
- Case $\varphi' = \langle\langle\Gamma\rangle\rangle \mathbf{G}\theta$: Similarly to the previous case also here the algorithm exploits the preimage function; however, differently from before, the latter is called a polynomial number of times with respect to the number of states in $\mathcal{G}$. Thus, this case may conclude in polynomial time w.r.t. the size of $\mathcal{G}$.
- Case $\varphi' = \langle\langle\Gamma\rangle\rangle \theta_1 \mathbf{U} \theta_2$: The algorithm concludes in polynomial time as in the previous case.

4 Implementation

In this section, we present the implementation (source code available at `https://anonymous.4open.science/r/ATLF-B4CC`) of the model checking algorithm and its integration with the VITAMIN tool [27,28,29] as a verification component to handle the verification of $\mathsf{ATL}[\mathcal{F}]$ formulas.

[5] Without this assumption the time complexity of the algorithm may change.

The VITAMIN tool operates on Streamlit[6], an open-source framework used for creating web applications. The tool takes as input a file representing the model, a formula, and applies the corresponding model checking algorithm for the chosen temporal logic. wCGSs are represented as graphs and implemented via adjacency matrices, as specified in the input file.

Model Parsing: Extracting Information from the Input File. Before applying the model checking algorithm, we need to obtain the model to which it will be applied to. A function for reading the input file has been implemented in VITAMIN.

The input file contains all the information regarding the model. It must follow a specific structure:

- `Transition` matrix: Represents the wCGS. Each row corresponds to a source state, and each column corresponds to a destination state. Transitions are represented by the matrix content: 0 indicates no transition, while other values indicate the associated tuple of actions.
- `Name_State`: Represents the states of the model. Their order matches the order in the adjacency matrix.
- `Initial_State`: Represents the initial state of the model.
- `Atomic_Propositions`: Represents atoms with truth values in each state. Only names and order are indicated here, crucial for interpreting values in the corresponding matrix.
- `Labelling` matrix: Represents the labelling function, indicating the value of each atomic proposition for each state. Values range between 0 and 1. States are in rows, atomic propositions in columns.
- `Number_of_Agents`: Crucial for understanding the wCGS configuration and the total number of moves required for any transition.

Formula Parsing. The formula parser is developed in Python using the ply package[7]. The output generated by the parser is an Abstract Syntax Tree (AST). The parser file includes all the logical operators used in $\mathsf{ATL}[\mathcal{F}]$, along with the correct syntax for their usage. This enables the parser to identify formulas with incorrect syntax and divide correct formulas into smaller sub-formulas. The actual model checking algorithm is then applied on the so extracted sub-formulas as presented in Algorithm 1.

5 Experiments

We tested our tool on the drones example presented in Example 1 on a machine with the following specifications: Intel(R) Core(TM) i7-7700HQ CPU @ 2.80GHz, 4 cores 8 threads, 16 GB RAM DDR4. Specifically, we verified that the $\mathsf{ATL}[\mathcal{F}]$ formula $\langle\langle c\rangle\rangle(dist \geq 0.5)\ \mathbf{U}$ safe is satisfied in the wCGS of Figure 2. The tool completed the verification within 0.005 seconds. We remind the reader

[6] https://streamlit.io/
[7] https://github.com/dabeaz/ply

that the implementation of the ATL[$\mathcal{F}$] model checking algorithm in VITAMIN only supports $min\{x, y\}$, $max\{x, y\}$, and $1 - x$ functions in $\mathcal{F}$. Consequently, we cannot represent the ≥ 0.5 comparison directly inside the formula. Nonetheless, considering the semantics of the **U** operator, it is enough to rewrite the formula as $\langle\langle c \rangle\rangle dist$ **U** safe and then to check whether the resulting value associated to the initial state is, or not, equal to or greater than 0.5. Note that, the rewritten formula is still an ATL[$\mathcal{F}$] formula and requires to be verified through the ATL[$\mathcal{F}$] model checking algorithm presented in this paper. The only difference with the original formula is that the comparison is not natively supported in the ATL[$\mathcal{F}$] implementation, but it is performed on the value resulting from the verification.

Synthetic models. To further experiment on our tool, we carried out additional tests on randomly generated wCGS models. Figure 3 presents the experimental results obtained by varying the system model, while keeping fixed an ATL[$\mathcal{F}$] formula[8]. In these synthetic experiments, the models differ from the ones presented in the Drone Battle scenario; in fact, they are automatically generated. Each model has a specific size, determined by the sum of the states and transitions within it. The distribution of the atomic propositions among the states is randomly generated for each model. As observed in Figure 3, the results exhibit the expected polynomial behaviour relative to the size of the model under analysis. It is important to note that, for each reported model size, over 10,000 models have been randomly generated and verified against the strategic formula. The plot depicts the average execution time. We also want to emphasise that the size of the models used in these experiments is not negligible. In fact, the largest models subjected to verification contain more than 1,000 states and over 100,000 transitions. Nonetheless, despite their size, our algorithm manages to complete the verification process in less than 40 seconds.

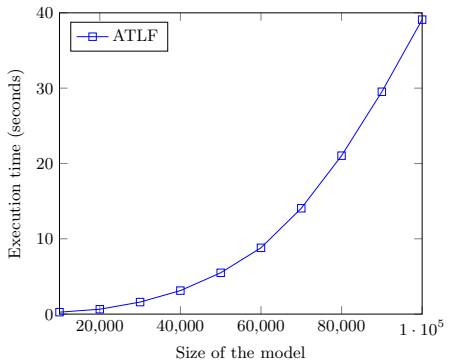

Fig. 3: Experimental results obtained on randomly generated wCGS.

[8] Such a formula corresponds to a liveness property where we check whether a specific atom in the model can be reached and with which value.

6 Conclusions and Future Work

In this work, we have presented $\mathsf{ATL}[\mathcal{F}]$ and its instantiation in the VITAMIN framework, thus creating the first tool for quantitative strategic reasoning. We addressed $\mathsf{ATL}[\mathcal{F}]$ at both theoretical and practical levels by first presenting its syntax, semantics, and model checking algorithm, and finally, by demonstrating its implementation as a VITAMIN component. We thoroughly analysed the resulting algorithms for model checking $\mathsf{ATL}[\mathcal{F}]$ formulas on wCGS models and conducted experiments to evaluate our approach.

This work opens up a new research direction that we expect to have a significant impact on the MAS formal verification community. Furthermore, the tool is highly dynamic and can be easily extended to a wide range of examples.

Acknowledgements

This research has been supported by the PNRR MUR project PE0000013-FAIR and the PRIN 2020 project RIPER.

References

1. Almagor, S., Boker, U., Kupferman, O.: Discounting in LTL. In: TACAS. Lecture Notes in Computer Science, vol. 8413, pp. 424–439. Springer (2014)
2. Almagor, S., Boker, U., Kupferman, O.: Formally reasoning about quality. Journal of the ACM (JACM) **63**(3), 1–56 (2016)
3. Alur, R., Henzinger, T.A., Kupferman, O.: Alternating-time temporal logic. Journal of the ACM (JACM) **49**(5), 672–713 (2002)
4. Alur, R., Henzinger, T.A., Mang, F.Y.C., Qadeer, S., Rajamani, S.K., Tasiran, S.: MOCHA: modularity in model checking. In: Hu, A.J., Vardi, M.Y. (eds.) Computer Aided Verification, 10th International Conference, CAV '98, Vancouver, BC, Canada, June 28 - July 2, 1998, Proceedings. Lecture Notes in Computer Science, vol. 1427, pp. 521–525. Springer (1998)
5. Aminof, B., Giacomo, G.D., Lomuscio, A., Murano, A., Rubin, S.: Synthesizing best-effort strategies under multiple environment specifications. In: Bienvenu, M., Lakemeyer, G., Erdem, E. (eds.) Proceedings of the 18th International Conference on Principles of Knowledge Representation and Reasoning, KR 2021, Online event, November 3-12, 2021. pp. 42–51 (2021)
6. Aminof, B., Giacomo, G.D., Rubin, S.: Best-effort synthesis: Doing your best is not harder than giving up. In: Zhou, Z. (ed.) Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI 2021, Virtual Event / Montreal, Canada, 19-27 August 2021. pp. 1766–1772. ijcai.org (2021)
7. Aminof, B., Kwiatkowska, M., Maubert, B., Murano, A., Rubin, S.: Probabilistic strategy logic. In: Kraus, S. (ed.) Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI 2019, Macao, China, August 10-16, 2019. pp. 32–38. ijcai.org (2019)
8. Aminof, B., Malvone, V., Murano, A., Rubin, S.: Graded strategy logic: Reasoning about uniqueness of nash equilibria. In: Jonker, C.M., Marsella, S., Thangarajah, J., Tuyls, K. (eds.) Proceedings of the 2016 International Conference on Autonomous Agents & Multiagent Systems, Singapore, May 9-13, 2016. pp. 698–706. ACM (2016)

9. Baier, C., Katoen, J.P.: Principles of model checking. MIT press (2008)
10. Belardinelli, F., Ferrando, A., Malvone, V.: An abstraction-refinement framework for verifying strategic properties in multi-agent systems with imperfect information. Artif. Intell. **316**, 103847 (2023)
11. Belardinelli, F., Jamroga, W., Malvone, V., Mittelmann, M., Murano, A., Perrussel, L.: Reasoning about human-friendly strategies in repeated keyword auctions. In: Faliszewski, P., Mascardi, V., Pelachaud, C., Taylor, M.E. (eds.) 21st International Conference on Autonomous Agents and Multiagent Systems, AAMAS 2022, Auckland, New Zealand, May 9-13, 2022. pp. 62–71. International Foundation for Autonomous Agents and Multiagent Systems (IFAAMAS) (2022)
12. Belardinelli, F., Jamroga, W., Mittelmann, M., Murano, A.: Verification of stochastic multi-agent systems with forgetful strategies. In: Dastani, M., Sichman, J.S., Alechina, N., Dignum, V. (eds.) Proceedings of the 23rd International Conference on Autonomous Agents and Multiagent Systems, AAMAS 2024, Auckland, New Zealand, May 6-10, 2024. pp. 160–169. International Foundation for Autonomous Agents and Multiagent Systems / ACM (2024)
13. Bouyer, P., Kupferman, O., Markey, N., Maubert, B., Murano, A., Perelli, G.: Reasoning about quality and fuzziness of strategic behaviors. ACM Transactions on Computational Logic **24**(3), 1–38 (2023)
14. Bouyer, P., Kupferman, O., Markey, N., Maubert, B., Murano, A., Perelli, G.: Reasoning about quality and fuzziness of strategic behaviors. ACM Trans. Comput. Log. **24**(3), 21:1–21:38 (2023)
15. Bouyer, P., Markey, N., Matteplackel, R.M.: Averaging in LTL. In: Baldan, P., Gorla, D. (eds.) CONCUR 2014 - Concurrency Theory - 25th International Conference, CONCUR 2014, Rome, Italy, September 2-5, 2014. Proceedings. Lecture Notes in Computer Science, vol. 8704, pp. 266–280. Springer (2014)
16. Brihaye, T., Laroussinie, F., Markey, N., Oreiby, G.: Timed concurrent game structures. In: Proc. of the 18th International Conference on Concurrency Theory, CONCUR 2007. Lecture Notes in Computer Science, vol. 4703, pp. 445–459. Springer (2007)
17. Bulling, N., Goranko, V.: Combining quantitative and qualitative reasoning in concurrent multi-player games. Auton. Agents Multi Agent Syst. **36**(1), 2 (2022)
18. Catta, D., Ferrando, A., Malvone, V.: Resource action-based bounded atl: a new logic for mas to express a cost over the actions. In: Arisaka, R., Anguix, V.S., Stein, S., Aydogan, R., van der Torre, L., Ito, T. (eds.) PRIMA 2024: Principles and Practice of Multi-Agent Systems - 25th International Conference, Kyoto, Japan, November 18-24, 2024, Proceedings. Lecture Notes in Computer Science, vol. to appear. Springer (2024)
19. Cermák, P., Lomuscio, A., Mogavero, F., Murano, A.: MCMAS-SLK: A model checker for the verification of strategy logic specifications. In: Biere, A., Bloem, R. (eds.) Computer Aided Verification - 26th International Conference, CAV 2014, Held as Part of the Vienna Summer of Logic, VSL 2014, Vienna, Austria, July 18-22, 2014. Proceedings. Lecture Notes in Computer Science, vol. 8559, pp. 525–532. Springer (2014)
20. Cermák, P., Lomuscio, A., Mogavero, F., Murano, A.: Practical verification of multi-agent systems against SLK specifications. Inf. Comput. **261**, 588–614 (2018)
21. Cermák, P., Lomuscio, A., Murano, A.: Verifying and synthesising multi-agent systems against one-goal strategy logic specifications. In: Bonet, B., Koenig, S. (eds.) Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence, January 25-30, 2015, Austin, Texas, USA. pp. 2038–2044. AAAI Press (2015)

22. Chatterjee, K., de Alfaro, L., Faella, M., Legay, A.: Qualitative logics and equivalences for probabilistic systems. Log. Methods Comput. Sci. **5**(2) (2009)
23. Chellas, B.F.: Modal Logic - An Introduction. Cambridge University Press (1980)
24. Chen, T., Forejt, V., Kwiatkowska, M.Z., Parker, D., Simaitis, A.: Prism-games: A model checker for stochastic multi-player games. In: Piterman, N., Smolka, S.A. (eds.) Tools and Algorithms for the Construction and Analysis of Systems - 19th International Conference, TACAS 2013, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2013, Rome, Italy, March 16-24, 2013. Proceedings. Lecture Notes in Computer Science, vol. 7795, pp. 185–191. Springer (2013)
25. Chen, T., Lu, J.: Probabilistic alternating-time temporal logic and model checking algorithm. In: Lei, J. (ed.) Fourth International Conference on Fuzzy Systems and Knowledge Discovery, FSKD 2007, 24-27 August 2007, Haikou, Hainan, China, Proceedings, Volume 2. pp. 35–39. IEEE Computer Society (2007)
26. De Alfaro, L., Faella, M., Henzinger, T.A., Majumdar, R., Stoelinga, M.: Model checking discounted temporal properties. Theoretical Computer Science **345**(1), 139–170 (2005)
27. Ferrando, A., Malvone, V.: Hands-on VITAMIN: A compositional tool for model checking of multi-agent systems. In: Alderighi, M., Baldoni, M., Baroglio, C., Micalizio, R., Tedeschi, S. (eds.) Proceedings of the 25th Workshop "From Objects to Agents", Bard (Aosta), Italy, July 8-10, 2024. CEUR Workshop Proceedings, vol. 3735, pp. 148–160. CEUR-WS.org (2024)
28. Ferrando, A., Malvone, V.: VITAMIN: A compositional framework for model checking of multi-agent systems. CoRR **abs/2403.02170** (2024)
29. Ferrando, A., Malvone, V.: Vitamin: A tool for model checking of mas. In: Collier, R., Ricci, A., Nallur, V. (eds.) Multi-Agent Systems - 21st European Conference, EUMAS 2024, Dublin, Ireland, August 26-28, 2024, Proceedings. Lecture Notes in Computer Science, vol. to appear. Springer (2024)
30. Gutierrez, J., Najib, M., Perelli, G., Wooldridge, M.J.: EVE: A tool for temporal equilibrium analysis. In: Lahiri, S.K., Wang, C. (eds.) Automated Technology for Verification and Analysis - 16th International Symposium, ATVA 2018, Los Angeles, CA, USA, October 7-10, 2018, Proceedings. Lecture Notes in Computer Science, vol. 11138, pp. 551–557. Springer (2018)
31. Henzinger, T.A., Prabhu, V.S.: Timed alternating-time temporal logic. In: Proc. of the 4th International Conference on Formal Modeling and Analysis of Timed Systems, FORMATS 2006. Lecture Notes in Computer Science, vol. 4202, pp. 1–17. Springer (2006)
32. Huang, X., van der Meyden, R.: Symbolic model checking epistemic strategy logic. In: Brodley, C.E., Stone, P. (eds.) Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence, July 27 -31, 2014, Québec City, Québec, Canada. pp. 1426–1432. AAAI Press (2014)
33. Jamroga, W., Konikowska, B., Kurpiewski, D., Penczek, W.: Multi-valued verification of strategic ability. Fundam. Informaticae **175**(1-4), 207–251 (2020)
34. Jamroga, W., Malvone, V., Murano, A.: Natural strategic ability. Artif. Intell. **277** (2019). https://doi.org/10.1016/J.ARTINT.2019.103170, https://doi.org/10.1016/j.artint.2019.103170
35. Jamroga, W., Malvone, V., Murano, A.: Natural strategic ability under imperfect information. In: Elkind, E., Veloso, M., Agmon, N., Taylor, M.E. (eds.) Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent

Systems, AAMAS '19, Montreal, QC, Canada, May 13-17, 2019. pp. 962–970. International Foundation for Autonomous Agents and Multiagent Systems (2019), http://dl.acm.org/citation.cfm?id=3331791

36. Kurpiewski, D., Jamroga, W., Knapik, M.: STV: model checking for strategies under imperfect information. In: Elkind, E., Veloso, M., Agmon, N., Taylor, M.E. (eds.) Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent Systems, AAMAS '19, Montreal, QC, Canada, May 13-17, 2019. pp. 2372–2374. International Foundation for Autonomous Agents and Multiagent Systems (2019)
37. Kurpiewski, D., Pazderski, W., Jamroga, W., Kim, Y.: Stv+reductions: Towards practical verification of strategic ability using model reductions. In: Dignum, F., Lomuscio, A., Endriss, U., Nowé, A. (eds.) AAMAS '21: 20th International Conference on Autonomous Agents and Multiagent Systems, Virtual Event, United Kingdom, May 3-7, 2021. pp. 1770–1772. ACM (2021)
38. Laroussinie, F., Markey, N., Oreiby, G.: Model-checking timed. In: Proc. of the 4th International Conference on Formal Modeling and Analysis of Timed Systems, FORMATS 2006. Lecture Notes in Computer Science, vol. 4202, pp. 245–259. Springer (2006). https://doi.org/10.1007/11867340_18
39. Li, X., Vasile, C.I., Belta, C.: Reinforcement learning with temporal logic rewards. In: 2017 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2017, Vancouver, BC, Canada, September 24-28, 2017. pp. 3834–3839. IEEE (2017). https://doi.org/10.1109/IROS.2017.8206234, https://doi.org/10.1109/IROS.2017.8206234
40. Lomuscio, A., Qu, H., Raimondi, F.: MCMAS: an open-source model checker for the verification of multi-agent systems. Int. J. Softw. Tools Technol. Transf. **19**(1), 9–30 (2017). https://doi.org/10.1007/S10009-015-0378-X, https://doi.org/10.1007/s10009-015-0378-x
41. Mittelmann, M., Murano, A., Perrussel, L.: Discounting in strategy logic. In: Proceedings of the Thirty-Second International Joint Conference on Artificial Intelligence, IJCAI 2023, 19th-25th August 2023, Macao, SAR, China. pp. 225–233. ijcai.org (2023)
42. Mogavero, F., Murano, A., Perelli, G., Vardi, M.Y.: Reasoning about strategies: On the model-checking problem. ACM Trans. Comput. Log. **15**(4), 34:1–34:47 (2014)
43. Nguyen, H.N., Alechina, N., Logan, B., Rakib, A.: Alternating-time temporal logic with resource bounds. J. Log. Comput. **28**(4), 631–663 (2018). https://doi.org/10.1093/LOGCOM/EXV034, https://doi.org/10.1093/logcom/exv034
44. Pnueli, A.: The temporal logic of programs. In: Proc. 18th Annual Symposium on Foundations of Computer Science (FOCS). pp. 46–57. IEEE Computer Society (1977). https://doi.org/10.1109/SFCS.1977.32
45. Vester, S.: On the complexity of model-checking branching and alternating-time temporal logics in one-counter systemss. In: Finkbeiner, B., Pu, G., Zhang, L. (eds.) Automated Technology for Verification and Analysis. pp. 361–377. Springer International Publishing, Cham (2015)