Playing Quantitative Games Against an Authority: On the Module Checking Problem

Wojciech Jamroga
SnT, University of Luxembourg, Luxembourg
ICS, Polish Academy of Sciences, Poland

Munyque Mittelmann
University of Naples Federico II
Naples, Italy

Aniello Murano
University of Naples Federico II
Naples, Italy

Giuseppe Perelli
Sapienza University of Rome
Rome, Italy

ABSTRACT

Module checking is a decision problem to formalize the verification of systems that must adapt their behavior to the input they receive from the environment, also viewed as an *authority*. So far, module checking has been only considered in the Boolean setting, which does not capture the different levels of quality inherent to complex systems (e.g., systems dealing with quantitative utilities or sensor inputs). In this paper, we address this issue by proposing quantitative module checking. We study the problem in the quantitative and multi-agent setting, which enables the verification of different levels of satisfaction in relation to a specification. We consider specifications given in Quantitative Alternating-time Temporal logics and investigate their complexity and expressivity.

KEYWORDS

Logics for Strategic Reasoning, Module Checking, Quantitative Verification

ACM Reference Format:
Wojciech Jamroga, Munyque Mittelmann, Aniello Murano, and Giuseppe Perelli. 2024. Playing Quantitative Games Against an Authority: On the Module Checking Problem . In *Proc. of the 23rd International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2024), Auckland, New Zealand, May 6 – 10, 2024*, IFAAMAS, 9 pages.

1 INTRODUCTION

Model checking is a well-established formal-method technique for the automated analysis of systems that can be modeled by state-transition models [19]. This verification method consists of checking the system for global correctness in relation to a logical specification. Early use of model checking mainly considered the verification of finite-state *closed systems*, whose behavior is completely determined by the state of the system. In this setting, models are usually given as labeled-state transition graphs equipped with some internal degree of non-determinism while system properties are specified in temporal logics such as the linear-time temporal logic LTL [47], the branching-time temporal logics CTL and CTL* [25], and the alternating-time temporal logics ATL and ATL* [5].

Proc. of the 23rd International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2024), N. Alechina, V. Dignum, M. Dastani, J.S. Sichman (eds.), May 6 – 10, 2024, Auckland, New Zealand. © 2024 International Foundation for Autonomous Agents and Multiagent Systems (www.ifaamas.org).

In the last three decades, model-checking techniques have been extended to the analysis of *open systems*, that is, reactive systems that interact with their environment and whose behavior depends on this interaction. Although temporal logics are suitable for describing the interactions of those systems, the model-checking algorithms used for the verification of closed systems are not appropriate for the verification of open systems. More precisely, open systems should be checked with respect to arbitrary environments and should take into account uncertainty regarding the environment. One of the first approaches introduced to model check finite-state open systems is *module checking* [36]. A *module* is a state-transition model with states partitioned into those controlled by the system and those controlled by the environment. Correctness in module checking means that the desired property must hold with respect to all possible interactions between the system and the environment. An additional source of nondeterminism is brought by the environment: the computation, from a state, can continue with any subset of its possible successor states. As for model checking, the correctness of a system is a matter of Boolean satisfaction: either it satisfies the specification or it does not. With module checking, however, this is a harder problem to deal with, as it requires to consider an infinite number of trees, one for each possible behavior of the environment.

After its initial proposal [36, 40], the module checking problem was extended to the setting where the environment has imperfect information about the state of the system [37]. An extension of the problem has been also used to reason about three-valued abstractions in [21, 29]. Previous work investigated module-checking through a tableaux-based approach [8] and also studied the module-checking problem for bounded pushdown modules [46]. Later, pushdown modules were considered to deal with infinite-state open systems both for perfect [13] and imperfect information settings [6, 11]. All these extensions have considered Boolean module checking, which is often inadequate when considering complex systems that interact with a physical environment. Those systems may deal with quantitative aspects and measurements (such as temperature, prices, and distances). In this paper, we address this issue by proposing quantitative module checking. We generalize the problem to the quantitative and multi-agent setting, which enables the verification of different *levels* of satisfaction of a specification.

Module checking for Multi-Agent Systems (MAS) captures the situation in which the system, composed of interacting agents, plays against an environment (or an *authority*) whose behavior may inhibit access to certain paths of the computation tree. In other

words, the environment dictates and restricts the possibilities for the MAS. As an analogy, imagine a group of children (the MAS), in which each child has their own goal (playing a video game, eating candies, ...). The children's mother (the environment) supervises them and regulates what they can achieve, e.g., by sending them to sleep early or hiding the TV's remote control. The quantitative dimension expresses *how much* the system satisfies their goals or *how much* the authority can limit them from achieving their goals. Going back to the analogy, this can represent how much time the children could enjoy the video game, or how much they enjoyed their snacks. More realistic examples can include a group of agents trying to communicate on a social network while being subjected to moderation policies [28], networks of autonomous vehicles dealing with transportation policies [7], and software bidding agents who may be imposed restrictions on online auctions [23].

Related work. Concerning the specification language, module checking was first investigated with respect to CTL/CTL* specifications [8, 36, 37] and μ-calculus [26]. More recent approaches have considered module checking of MAS with specifications in ATL and ATL* [12, 33, 34]. In these approaches, the system is modeled by a multi-agent finite-state concurrent game, whose transitions are determined by the actions made simultaneously and independently by all the agents. Module checking with ATL allows us to express the strategic abilities of agents to achieve certain goals, when interacting with an external environment. In this paper we consider specifications given in quantitative ATL and ATL* which are inspired by LTL$[\mathcal{F}]$ [4], a multi-valued logic that augments LTL with quality operators. Other quantitative extensions of ATL have also been investigated in the context of model checking, such as timed ATL [14, 31], multi-valued ATL [32], and weighted versions of ATL [17, 42, 49]. SL$[\mathcal{F}]$ [10] was recently introduced as a quantitative extension of Strategy Logic (SL) and it subsumes both SL [18, 45] and LTL$[\mathcal{F}]$. Model checking a SL$[\mathcal{F}]$-formula φ is k-Exptime-complete in the number k of alternation on the quantifications in φ [10], but it does not subsume module checking ATL, because of the existence of nondeterminism in the latter.

With module checking, we capture how the environment restricts the executions of the game. This is related to normative systems [1], which define constraints (in terms of obligations and permissions) on the behavior of agents. In some approaches, norms correspond to labelling 'violating' aspects of the game, such as states [20, 22, 44], transitions [2], and paths [16]. The synthesis of norms with system objectives specified in temporal logics has also been studied, for instance with objectives in LTL [15] and ATL* [3].

Contribution. This is the first work to consider module checking in the quantitative setting. We define the problem of quantitative module checking and investigate its complexity in relation to specifications given in ATL$^*[\mathcal{F}]$ and ATL$[\mathcal{F}]$, the weighted variants of ATL* and ATL, resp. We also study the model checking problem for both languages. Table 1 sums up the complexity results. For the upper bounds, we adopt an automata-theoretic approach. Precisely, to solve the module-checking question we build a tree automaton[1] and check for its emptiness.

[1]We will use standard parity and Büchi tree automata. The interested reader can refer to [39] for an introduction to these automata.

	Model checking	Module checking
ATL$[\mathcal{F}]$	Ptime-complete	Exptime-complete
ATL$^*[\mathcal{F}]$	2Exptime-complete	3Exptime-complete

Table 1: Summary of the complexity results

We study the expressive power of ATL$^*[\mathcal{F}]$ module checking showing that it allows us to capture and verify properties that cannot be captured by decision problems based on the existing variants of alternating-time logics. Precisely, we show that ATL$^*[\mathcal{F}]$ module checking is neither subsumed by ATL* module checking nor by ATL$^*[\mathcal{F}]$ model checking.

2 QUANTITATIVE ATL AND ATL*

For the remainder of the paper, we fix a finite set of atomic propositions AP and a finite set of agents Ag, except when stated otherwise. We also let $\mathcal{F} \subseteq \{f\colon [0,1]^m \to [0,1] \mid m \in \mathbb{N}\}$ be a set of functions over $[0,1]$ of possibly different arities, that will parameterize the logics we consider. With slight abuse of notation, we denote by $f \in \mathcal{F}$ both the function and the corresponding functor. It will be clear from the context to which the one symbol corresponds. We assume all functions in $\mathcal{F}$ are computable in polynomial time. This is enough to capture classic functions, such as the ones representing disjunction, negation, average, minimum, etc. We write $\boldsymbol{c}$ for a tuple of objects $(c_a)_{a\in \mathrm{Ag}}$, one for each agent, and call it a *profile*. Given a profile $\boldsymbol{c}$ and $a \in \mathrm{Ag}$, we let c_a be agent a's component.

We begin by introducing the Quantitative Alternating-time Temporal logics ATL$^*[\mathcal{F}]$ and ATL$[\mathcal{F}]$.

Definition 1. *The syntax of* ATL$^*[\mathcal{F}]$ *is defined by the grammar*

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

where $p \in \mathrm{AP}$, $A \in 2^{\mathrm{Ag}}$, *and* $f \in \mathcal{F}$.

The intuitive reading of the operators is as follows: $\langle\langle A\rangle\rangle\varphi$ means that there exists a strategy for the coalition A such that, no matter how the other players act, φ holds; **X**, **U**, and **R** are the usual temporal operators "next", "until". and "release". The meaning of $f[\varphi_1, ..., \varphi_n]$ depends on the function f.

We define usual Boolean operators as functions and assume they are always present in $\mathcal{F}$. Precisely, we have: $\top := 1$, $\bot := 0$, $\varphi \vee \varphi' := \max(\varphi, \varphi')$, $\varphi \wedge \varphi' := \min(\varphi, \varphi')$, and $\neg\varphi := 1 - \varphi$, respectively. We also make use of the usual syntactic sugar $\mathbf{F}\varphi := \top \mathbf{U} \varphi$ and $\mathbf{G}\varphi := \neg\mathbf{F}\neg\varphi$ for temporal operators.

An ATL$^*[\mathcal{F}]$ formula of the form $\langle\langle A\rangle\rangle\varphi$ is also called a state formula, or sentence. An important syntactic restriction of ATL$^*[\mathcal{F}]$, namely ATL$[\mathcal{F}]$, is defined as follows.

Definition 2. *The syntax of* ATL$[\mathcal{F}]$ *is defined by the grammar*

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

where $p \in \mathrm{AP}$, $A \in 2^{\mathrm{Ag}}$, *and* $f \in \mathcal{F}$.

The language ATL$[\mathcal{F}]$ allows us to use only a single temporal operator in the scope of the strategy quantifiers. Such restriction is still expressive enough to represent meaningful properties of games, as well as having more convenient computational complexities than ATL$^*[\mathcal{F}]$ (see Theorems 4.1 and 4.2).

ATL* is defined over Concurrent Game Structures, which are a model of concurrent computation where agents simultaneously choose their actions. In such structures, propositions have Boolean labels, representing whether they are true or false at a given state. Differently from ATL*, the semantics of ATL*[$\mathcal{F}$] are based on Weighted Concurrent Game Structures (wCGS), which are also considered for SL[$\mathcal{F}$]. In a wCGS, atomic propositions describe features of the game and are assigned a weight in each state.

Definition 3 (wCGS). *A* weighted concurrent game structure *(wCGS) is a tuple* $\mathcal{G} = (\mathrm{Ac}, V, d, o, \ell)$ *where* Ac *is a finite set of* actions, *V is a finite set of* states, *and* $\ell : V \times \mathrm{AP} \to [0,1]$ *is a* weight function. *The* availability function $d : \mathrm{Ag} \times V \to 2^{\mathrm{Ac}}$ *defines a non-empty set of actions available to agents at each state and the (deterministic)* transition function *o assigns the outcome state $v' = o(v, \boldsymbol{c})$ to each state v and tuple of actions $\boldsymbol{c} \in \prod_{a\in\mathrm{Ag}} d(a, v)$ that can be executed by the agents in v. A pointed wCGS is a pair $(\mathcal{G}, v_\iota)$ where $v_\iota \in V$ is a special state designed as initial.*

In the following examples, we assume that $\mathcal{F}$ contains the function $\leq : (x, y) = 1$ if $x \leq y$ and $\leq : (x, y) = 0$ otherwise; the function $<$, and the function $=$ (defined similarly, with $<$ and $=$ instead of $\leq$, resp.). We use the infix notations $x \leq y$, $x < y$, and $x = y$ in the formulas for readability. We also assume that $\mathcal{F}$ contains the $|\mathrm{Ag}|$-ary sum function $\sum : x_1, ..., x_n \mapsto \min(1, \max(0, \sum_k x_n))$.

Example 1 (Weighted voting game). *In a weighted voting game, each player is given a numeric weight and agents can potentially benefit by cooperating and forming coalitions [24]. In its standard formulation, a coalition takes the value 1 if the sum of the weights of its components exceeds a particular threshold, and the value 0 otherwise. We consider the case in which the weighting is dynamic, in the sense that agents' have different weights in different states of the game. We omit the formalization of the wCGS and refer to [43] for similar constructions. For each agent a, we let the atomic proposition w_a denote her weight in each state.*

We can express the winning condition for a coalition $A \in 2^{\mathrm{Ag}}$ and the threshold $\epsilon \in [0,1]$, with the following ATL[$\mathcal{F}$] *formula: $\langle\langle A\rangle\rangle \mathbf{F}(\sum_{a\in A} w_a > \epsilon)$. That is, the coalition can exceed the threshold ϵ in the future. We can also consider the goal in which agents try to maximize their combined weight, rather than simply achieving a threshold. The* ATL[$\mathcal{F}$] *formula $\langle\langle A\rangle\rangle \mathbf{F}(\sum_{a\in A} w_a)$ captures the value the agents can achieve in the future by combining their weights.*

The maximum value that can be achieved in the future among all coalitions is captured by the formula $\bigvee_{A\in 2^{\mathrm{Ag}}} \langle\langle A\rangle\rangle \mathbf{F}(\sum_{a\in A} w_a)$.

Nondeterministic choices of agents in a wCGS $\mathcal{G}$ can be represented by sets of actions. At state v, agent a can select any nonempty set $\alpha \subseteq d(a, v)$. The set of successors of v after the nondeterministic choice α is the union of successor states for each action in α. In a state $v \in V$, each player a chooses an action $c_a \in d(a, v)$, and the game proceeds to state $o(v, \boldsymbol{c})$ where $\boldsymbol{c}$ is an *action profile* $(c_a)_{a\in\mathrm{Ag}}$, (that is, a tuple of actions, one for each agent).

A *path* $\pi = \pi_0\pi_1... \in V^\omega$ is an infinite sequence of states such that for every $i \geq 0$ there exists an action profile $\boldsymbol{c} \in \prod_{a\in\mathrm{Ag}} d(a, \pi_i)$ such that $o(\pi_i, \boldsymbol{c}) = \pi_{i+1}$. We write π_i for the state at index i in path π. Moreover, we write $\pi_{\geq i}$ for the suffix of π starting from index i. A *history* h is a finite prefix of a path and $\mathrm{last}(h)$ is the last state of history h. We let Hist be the set of histories.

A (perfect recall) *strategy* for agent a is a function $\sigma : \mathrm{Hist} \to \mathrm{Ac}$ such that $\sigma_a(h) \in d(a, \mathrm{last}(h))$ that maps each history to an action. We let Str_a be the set of all strategies for agent a, and $Str = \cup_{a\in\mathrm{Ag}} Str_a$. For a state v, a coalition of agents $A \in 2^{\mathrm{Ag}} \setminus \emptyset$, a strategy profile, denoted $\boldsymbol{\sigma}_A = (\sigma_a)_A \in Str_A = \prod_{a\in A} Str_a$ is a collection of strategies, one for each agent a in A. The outcome function $\mathrm{Out}(v, \boldsymbol{\sigma}_A)$ returns the set of all paths starting on state v that can occur when agents in A execute the strategy profile $\boldsymbol{\sigma}_A$.

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

- $[\![p]\!]^{\mathcal{G}}(\pi) = \ell(\pi_0, p)$
- $[\![f[\varphi_1,..., \varphi_m]]\!]^{\mathcal{G}}(\pi) = f([\![\varphi_1]\!]^{\mathcal{G}}(\pi), ..., [\![\varphi_m]\!]^{\mathcal{G}}(\pi))$
- $[\![\langle\langle A\rangle\rangle\varphi]\!]^{\mathcal{G}}(\pi) = \max_{\boldsymbol{\sigma}_A\in \mathrm{Str}_A} \min_{\pi'\in\mathrm{Out}(\pi_0,\boldsymbol{\sigma}_A)} \{[\![\varphi]\!]^{\mathcal{G}}(\pi')\}$
- $[\![\mathbf{X}\varphi]\!]^{\mathcal{G}}(\pi) = [\![\varphi]\!]^{\mathcal{G}}(\pi_{\geq 1})$
- $[\![\varphi_1\mathbf{U}\varphi_2]\!]^{\mathcal{G}}(\pi) = \sup_{i\geq 0}\{\min\left([\![\varphi_2]\!]^{\mathcal{G}}(\pi_{\geq i}), \min_{0\leq j<i}[\![\varphi_1]\!]^{\mathcal{G}}(\pi_{\geq j})\right)\}$
- $[\![\varphi_1\mathbf{R}\varphi_2]\!]^{\mathcal{G}}(\pi) = 1 - \sup_{i\geq 0}\{\min\left(1 - [\![\varphi_2]\!]^{\mathcal{G}}(\pi_{\geq i}), \min_{0\leq j<i}(1 - [\![\varphi_1]\!]^{\mathcal{G}}(\pi_{\geq j}))\right)\}$

where sup *denotes the supremum.*

$\langle\langle A\rangle\rangle\varphi$ is the best satisfaction value of φ that the agents in A can ensure, no matter how the other agents behave. $\varphi_1\mathbf{U}\varphi_2$ maximizes, over all positions along the play, the minimum between the value of φ_2 at that position and the minimal value of φ_1 before this position. The intuition of the remaining operators is defined similarly.

Note that, for a sentence $\langle\langle A\rangle\rangle\varphi$, the satisfaction value does not depend on the entire path π but only on its initial state π_0. Thus, we also write $[\![\varphi]\!]^{\mathcal{G}}(v)$ to denote the satisfaction value of a sentence w.r.t. the pointed wCGS $(\mathcal{G}, v)$. Finally, we write $[\![\varphi]\!]^{\mathcal{G}}$ to denote the function mapping each node v to its satisfaction value $[\![\varphi]\!]^{\mathcal{G}}(v)$.

3 QUANTITATIVE MODULE CHECKING

We now introduce weighted modules and the problem of module checking for ATL[$\mathcal{F}$] and ATL*[$\mathcal{F}$].

3.1 Modules

In its simplest formulation, a module is a labeled-state transition system with two kinds of agents, the system and the environment. The set of states is partitioned into those owned by the system, and those owned by the (nondeterministic) environment [36]. In the multi-agent setting [34], there are multiple agents (in place of the system player) that share the control with the environment. Here, we extend the definition from [34] to the weighted setting, in which atomic propositions have values in $[0, 1]$.

Definition 5 (Modules). *A* multi-agent weighted module *(or simply a* module*) is a pointed wCGS $(\mathcal{G}, v_\iota)$ that contains a special agent called "the environment" ($env \in \mathrm{Ag}$) and whose states are partitioned into those* owned *by the environment (i.e., $|d(a, v)| = 1$ for all $a \neq env$) and those where the environment is* passive *(i.e., $|d(env, v)| = 1$).*

In other words, a module is a special pointed wCGS where the executions are controlled in turn by either the environment or the rest of agents. Note that such alternation in the control is not strict, meaning that either the environment or the agents can have

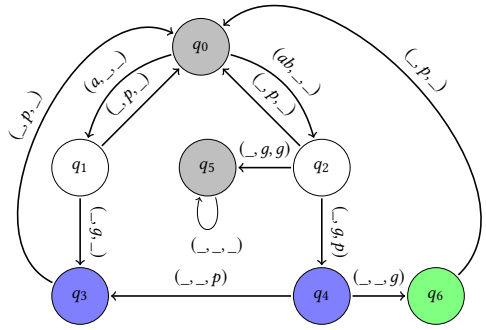

Figure 1: Weighted voting game $\mathcal{G}_{wv}$. The symbol "_" in an action profile represents any arbitrary action of the respecting agent. States q_0 and q_5 (in grey) are controlled by the environment. Agents have their weights equal 0 in all states except states q_3, q_4 (in blue) and q_6 (in green). In state q_3, $w_{\mathbf{Ann}} = 0.5$ and $w_{\mathbf{Bob}} = 0$. In state q_6, $w_{\mathbf{Ann}} = 0.5$ and $w_{\mathbf{Bob}} = 0.5$.

multiple controlling round in a row. From now on, we consider only multi-agent weighted modules. Moreover, whenever it is clear from the context or not otherwise stated, we denote a module $(\mathcal{G}, v_\iota)$ simply by $\mathcal{G}$, assuming the initial state has been fixed.

Example 2 (Weighted voting game cont.). *Let us present a module $(\mathcal{G}_{wv}, q_0)$ for the weighted voting game (Example 1). Figure 1 presents the states and transitions in the module. The module includes two agents, Ann and Bob. The atomic proposition control denotes whether a state is owned by the environment (that is, the transition on that state does not depend on agents' actions). The environment owns states q_0 and q_5, while the agents own the remaining ones. Each agent's goal is to maximize her own weight. The weights are equal to 0 in all states except q_3, q_4, and q_6. In states q_3 and q_4, $w_{Ann} = 0.5$ and $w_{Bob} = 0$. In state q_6, $w_{Ann} = 0.5$ and $w_{Bob} = 0.5$.*

The environment is an authority able to prevent Bob from participating. In q_0, the environment has two actions available, denoted a and ab. When it plays a, only Ann can perform actions in the next states. Conversely, if the environment plays ab, the game turns into a situation in which both agents can perform actions. Agents have two possible actions: either to pass (denoted p) or to gain (denoted g). In states not owned by the agents, they can only pass. By playing g, they gain resources for themselves, which has the effect of obtaining the weight 0.5 for themselves in the next state. If agents try to gain resources at the same time, the game goes to state q_5 in which the environment prevents both from playing.

3.2 Module Checking

Given finite sets D of directions, AP of atomic propositions, and $\mathcal{V} \subseteq [0,1]$ of possible values, an $(\mathrm{AP}, \mathcal{V})$-labeled D-tree, (or tree for short when the parameters are understood), is a pair $t = (T, \ell)$ where $T \subseteq D^+$ is closed under non-empty prefixes, all *nodes* $u \in T$ start with the same direction r, called the *root*, and have at least one *child* $u \cdot d \in T$, and $\ell : T \to \mathcal{V}^{\mathrm{AP}}$ is a *weight function*. A *branch* $\lambda = u_0 u_1 \ldots$ is an infinite sequence of nodes such that for all $i \geq 0$, we have that u_{i+1} is a child of u_i. We let $\mathrm{Br}(u)$ be the set of branches that start in node u. We say that a tree $t = (T, \ell)$ is *Boolean in p*, written $\mathbb{B}^+ tp$, if for all $u \in T$ we have $\ell(u)(p) \in \{0, 1\}$.

For a wCGS $\mathcal{G} = (\mathrm{Ac}, V, d, o, \ell)$, the *unwinding* of $\mathcal{G}$ is the module $\mathsf{Unw}(\mathcal{G}) = (\mathrm{Ac}, H, d', o', \ell')$, pointed to the root ε, where $H \subseteq V^*$ is the (prefix-closed) set of histories in $\mathcal{G}$, $d'(h) = d(\mathrm{last}(h))$ for each $h \in H$, $o'(h, \boldsymbol{c}) = h \cdot o(\mathrm{last}(h), c)$, for every history h and profile of actions $\boldsymbol{c}$, and $\ell'(h) = \ell(\mathrm{last}(h))$. Let $\mathrm{Val}_{\mathcal{G}} = \{\ell(v, p) \mid (v, p) \in V \times \mathrm{AP}\}$. Note that $\mathsf{Unw}(\mathcal{G})$ defines the unique $(\mathrm{AP}, \mathrm{Val}_{\mathcal{G}})$-labeled V-tree, where the successors of a node v are all possible successors in $\mathcal{G}$ and the weight function corresponds to the one in $\mathcal{G}$. Sometimes, we use $\mathsf{Unw}(\mathcal{G})$ to also denote the corresponding $(\mathrm{AP}, \mathrm{Val}_{\mathcal{G}})$-labelled tree. For a module $(\mathcal{G}, v_\iota)$, the set of (environment) strategy trees of $\mathcal{G}$, denoted $\mathrm{exec}(\mathcal{G})$ is obtained from $\mathsf{Unw}(\mathcal{G})$ by pruning some environment transitions (i.e., transitions from environment states). Formally, $\mathcal{T} = (\mathrm{Ac}, T, d', o, \ell') \in \mathrm{exec}(\mathcal{G})$ if $T \subseteq H$ is a prefix-closed subset of histories such that, for every $h \in T$, it holds:

- if $\mathrm{last}(h)$ is a state not own by the environment, then $o'(h, \boldsymbol{c}) \in T$ for every available profile of actions $\boldsymbol{c}$;
- if $\mathrm{last}(h)$ is a state own by the environment, available profile of agent's actions $\boldsymbol{c}$ such that $o'(h, \boldsymbol{c}) \in T$.

Every $(\mathrm{AP}, \mathrm{Val}_{\mathcal{G}})$-labeled tree $\mathcal{T} \in \mathrm{exec}(\mathcal{G})$ corresponds to a subtree of $\mathsf{Unw}(\mathcal{G})$ where only children nodes (and subsequent successors) of the environment nodes are pruned.

Definition 6. *Given an $\mathsf{ATL}^*[\mathcal{F}]$ formula φ, a pointed wCGS $(\mathcal{G}, v)$, and predicate $P \subseteq [0,1]$,* Model checking $\mathsf{ATL}^*[\mathcal{F}]$ *consists in deciding whether the satisfaction value of φ in $\mathcal{G}$ is in P, i.e., $[\![\varphi]\!]^{\mathcal{G}}(v) \in P$. In this case, we say that $(\mathcal{G}, v)$ P-*satisfies *φ (written: $(\mathcal{G}, v) \models_P \varphi$).*

Definition 7. *Given an $\mathsf{ATL}^*[\mathcal{F}]$ formula φ and a module $(\mathcal{G}, v)$, we define the* reactive semantics *of φ as the set of truth values for φ in all strategy trees of the module, i.e., $[\![\varphi]\!]^{\mathcal{G}}_r(v) = \{[\![\varphi]\!]^{\mathcal{T}}(v) \mid \mathcal{T} \in \mathrm{exec}(\mathcal{G})\}$* [2]*. Module checking $\mathsf{ATL}^*[\mathcal{F}]$ consists in deciding whether $[\![\varphi]\!]^{\mathcal{G}}_r(v) \subseteq P$, for a given predicate $P \subseteq [0,1]$. In that case, we say that $(\mathcal{G}, v)$* reactively *P-satisfies φ (written: $(\mathcal{G}, v) \models^r_P \varphi$).*

In the case that atomic propositions only take values in $[0,1]$ and the functions in $\mathcal{F}$ are restricted to the ones representing disjunction and negation, the $\mathsf{ATL}^*[\mathcal{F}]$ and $\mathsf{ATL}[\mathcal{F}]$ correspond to ATL^* and ATL, resp. Similarly, the fragment of $\mathsf{ATL}^*[\mathcal{F}]$ with only temporal operators and the functions for disjunction and negation corresponds to Fuzzy Linear-time Temporal Logic [27, 41].

Since $\mathsf{ATL}[\mathcal{F}]$ and $\mathsf{ATL}^*[\mathcal{F}]$ can be seen as fragments of $\mathsf{SL}[\mathcal{F}]$ [10], their model-checking problems are subsumed by $\mathsf{SL}[\mathcal{F}]$ model-checking. On the other hand, the module checking problem involves nondeterminism of the environment, which has not been considered for $\mathsf{SL}[\mathcal{F}]$ and is not captured by its model checking. Since $\mathsf{SL}[\mathcal{F}]$ is deterministic, the system always executes the same strategies from a given history. An example of nondeterminism for the environment is the situation in which may rain tomorrow, which would cause the parks to be closed. In this case, the execution trees in which the agents enjoy the day at the park could be pruned, depending on the behavior of the environment. This setting is captured by module checking, but not by $\mathsf{SL}[\mathcal{F}]$.

[2] We use "r" to denote that the semantics is reactive, in opposition to the satisfaction value of model-checking (for which we write simply $[\![.]\!](.)$).

Example 3 (Weighted voting game cont.). *Let us resume the module $(\mathcal{G}_{wv}, q_0)$ of Example 2. Let A be the coalition of Ann and Bob. It is not the case that $\mathcal{G}_{wv}$ reactively $\{0.5, 1\}$-satisfies $\langle\langle Bob\rangle\rangle \mathbf{F} w_{Bob}$ nor $\{1\}$-satisfies $\langle\langle A\rangle\rangle \mathbf{F} \sum_{a\in A}(w_a)$. Instead, $\mathcal{G}_{wv}$ reactively $\{0.5, 1\}$-satisfies $\langle\langle A\rangle\rangle \mathbf{F} \sum_{a\in AB}(w_a)$, because Ann can always obtain the required weight. Note that no agent by itself can bring back the control from the system, that is it is not the case that $\mathcal{G}_{wv}$ reactively $\{1\}$-satisfies $\langle\langle Ann\rangle\rangle \mathbf{F}\neg control \vee \langle\langle Bob\rangle\rangle \mathbf{F}\neg control$. If they cooperate, they can avoid the state in which the environment prevents them from playing, that is, $\mathcal{G}_{wv}$ reactively $\{1\}$-satisfies $\langle\langle A\rangle\rangle \mathbf{F}\neg control$.*

4 COMPLEXITY

To solve module checking, we adopt an automata-theoretic approach, whose overall complexity varies according to whether we are using an ATL$^*[\mathcal{F}]$ or an ATL$[\mathcal{F}]$ specification.

We first recall some basic concepts about tree automata (see [30], for a survey). We start with alternating Parity tree automata (APT), which are formally defined as tuples $\mathcal{A} = \langle Q, q_0, D, \Sigma, \delta, \alpha\rangle$ where Q, D, and Σ are non-empty finite sets of states, directions, and input letters, $q_0 \in Q$ is an initial state, $\alpha = (F_1, ..., F_k)$ is a sequence of subsets of Q where $F_1 \subseteq ... \subseteq F_k = Q$ is a parity acceptance condition and k the index of the automaton, and $\delta : Q \times \Sigma \to B^+(D \times Q)$ is an alternating transition function that maps each pair of states and input symbols into a negation-free Boolean formula on the set of propositions of the form (d, q), where d is a direction and q a state. Note that an APT, while visiting a node of the input tree, can send several copies of itself to the same successor. A run of an APT $\mathcal{A}$ on a Σ-labeled D-tree $t = (T, \ell)$ is a $(Q \times T)$-labeled $\mathbb{N}$-tree $R = (Tr, r)$ such that (i) $r(\varepsilon) = (q_0, \varepsilon)$ and (ii) for all $y \in Tr$ with $r(y) = (q, x)$, there exists a set $S \subseteq D \times Q$ with $S \models \delta(q, \ell(x))$ such that, for all $(d, q') \in S$, there is an index $i \in \mathbb{N}$ for which it holds that $r(y \cdot i) = (q', x \cdot d)$. The run R is accepting if, for every branch λ, the least index $1 \le i \le k$ such that at least one state of F_i occurs infinitely often in λ is even. A tree t is accepted by $\mathcal{A}$ if there is an accepting run of $\mathcal{A}$ on it. By $\mathcal{L}(\mathcal{A})$ we denote the language accepted by the automaton $\mathcal{A}$, i.e., the set of all trees that $\mathcal{A}$ accepts. $\mathcal{A}$ is said empty if $\mathcal{L}(\mathcal{A}) = \emptyset$. The emptiness problem for $\mathcal{A}$ is to decide whether $\mathcal{L}(\mathcal{A}) = \emptyset$. A nondeterministic Parity tree automaton (NPT) is a special case of an APT in which, when its transition relation is rewritten in disjunctive normal form, each conjunction in the transition function δ has exactly one move (d, q) associated with each direction d. A nondeterministic Büchi tree automaton (NBT) is an NPT with $\alpha = (F_1 = \emptyset, F_2, F_3 = Q)$. Note that for an NBT, the acceptance condition reduces to having a state in F_2 occur infinitely often on every branch. For this reason, an NBT can also be denoted as $\mathcal{A} = \langle Q, q_0, D, \Sigma, \delta, F\rangle$, with $F = F_2$ being the set that must occur infinitely often for acceptance.

We can now proceed with the solution of the module-checking problem. For simplicity, we show the procedure for ATL$^*[\mathcal{F}]$ formulas φ of the form $\langle\langle A\rangle\rangle\psi$ with ψ containing no strategy quantifiers. Such a procedure can be easily lifted to address any ATL$^*[\mathcal{F}]$ and ATL$[\mathcal{F}]$ formulas in the usual way ([5, 10]) [3]. Consider a pointed $\mathcal{G}$ $(\mathcal{G}, v_\iota)$ and a coalition A. Observe that every strategy profile σ_A defines an (AP, V)-labeled V-tree T_{σ_A, v_ι} such that, if $u, u \cdot v \in T_{\sigma_A, v_\iota}$, then $v = o(\mathrm{last}(u), \boldsymbol{c})$, for some $\boldsymbol{c}$ consistent with $\sigma_A(u)$. In other words, the tree T_{σ_A, v_ι} collects all possible outcomes in $\mathcal{G}$ that are compatible with the execution of σ_A. We have the following.

Lemma 1. *For a pointed wCGS $(\mathcal{G}, v_\iota)$ and a coalition A, there exists an NBT $\mathcal{A}^A_{\mathcal{G}, v_\iota}$ such that*

$$\mathcal{L}(\mathcal{A}^A_{\mathcal{G}, v_\iota}) = \{T_{\sigma_A, v_\iota} \mid \text{for some strategy profile } \sigma_A\}.$$

Proof sketch. The NBT $\mathcal{A}^A_{\mathcal{G}, v_\iota} = \langle V, v_\iota, D, (\mathrm{AP}, \mathcal{V}), \delta, V\rangle$ is defined over the set of states V, the set $\mathrm{AP}, \mathcal{V}$ of atomic proposition evaluations as alphabet, the initial state v_ι, and every state as final. Let $C = \prod_{a\in A} d(a, v)$ and $D = \prod_{a\in \mathrm{Ag}\setminus A} d(a, v)$, the latter being also the set of directions of the automaton. The transition function δ is defined such that, for every state $v \in V$, we have that

$$\delta(v, \ell(v)) = \bigvee_{c_A \in C} \bigwedge_{c_{\mathrm{Ag}\setminus A} \in D} (o(v, c_A \cup c_{\mathrm{Ag}\setminus A}), c_{\mathrm{Ag}\setminus A}).$$

This means that, if the input corresponds to the labeling $\ell(v)$ of the current state v, the successors correspond to a possible choice c_A of A and, for each of them, the automaton branches to each direction $c_{\mathrm{Ag}\setminus A}$ that corresponds to the choice of $\mathrm{Ag}\setminus A$. Otherwise, if the input is not the labeling of the current state, the automaton sends to the empty set, meaning that we reach a dead end, and it is not possible to generate any accepting run from there onward. The statement straightforwardly follows from the construction. Moreover, the size of $\mathcal{A}_{\mathcal{G}, v_\iota}$ is linear in the size of the underlying wCGS $\mathcal{G}$. □

Also, we can define an automaton accepting trees with only branches that P-satisfy a given LTL$[\mathcal{F}]$ formula ψ.

Proposition 1. *Let $\mathcal{V} \subseteq [0, 1]$ be a finite set of values such that $\{0, 1\} \subseteq \mathcal{V}$, and let D be a finite set of directions. For every formula $\psi \in$ LTL$[\mathcal{F}]$ and predicate $P \subseteq [0, 1]$, there exists an APT $\mathcal{A}^{\mathcal{V},P}_\psi$ with doubly-exponentially many states w.r.t. the size of ψ and exponentially many colors w.r.t. the size of ψ, such that for every $(\mathrm{AP}, \mathcal{V})$-labelled D-tree t, $\mathcal{A}^{\mathcal{V},P}_\psi$ accepts t if and only if every branch λ of t P-satisfies ψ. If the LTL$[\mathcal{F}]$ formula ψ has* no nesting of temporal operators, *$\mathcal{A}^{\mathcal{V},P}_\psi$ has an exponential number of states w.r.t. the size of ψ and 2 colors.*

Proof sketch. For every formula $\psi \in$ LTL$[\mathcal{F}]$, we can write the formula $\forall\psi$, which can be regarded as a Booleanly-quantified CTL* (BQCTL$^*[\mathcal{F}]$) formula, introduced in [10], of nesting depth 1. Such formula is P-satisfied on a $(\mathrm{AP}, \mathcal{V})$-labelled D-tree t iff every branch λ of t P-satisfied ψ. This means that the language of $\mathcal{A}^{\mathcal{V},P}_\psi$ must be exactly made by such labelled trees.

The proof of Proposition 4 in [10] shows how to construct such an automaton. Note that having nesting depth 1 ensures that $\mathcal{A}^{\mathcal{V},P}_\psi$ is of the requested size.

In case ψ has no nesting operators, the BQCTL$^*[\mathcal{F}]$ formula $\forall\psi$ belongs to the fragment BQCTL$[\mathcal{F}]$, obtained from extending CTL instead of CTL* to the quantitative setting. Observe that in the proof of Proposition 4 in [10] the approach is to construct, for each value $v \in P \cap \mathcal{V}$, an automaton accepting trees whose satisfaction value is exactly v. Thus, the union of these automata accepts labeled trees whose satisfaction value belongs to $P \cap \mathcal{V}$. As the temporal part

[3] For an arbitrary ATL$^*[\mathcal{F}]$ or ATL$[\mathcal{F}]$ formula, the procedure consists of recursively solving the innermost formula with strategic operators and replacing it by a proposition whose weight is the satisfaction value of such formula.

encapsulates the standard automata approach, the construction for a CTL-like formula builds an automaton with polynomially many states and 2 colors. However, as we have to build the union of $|P \cap \mathcal{V}|$ of them, and the latter can be exponential in the size of ψ, we obtain that the overall construction is of exponentially many states and 2 colors. □

For a given subset A of agents and a state v, consider the function

$$\text{Post}(A, v) = \{o(v, \boldsymbol{c_A}) \mid \boldsymbol{c_A} \in \prod_{a \in A} d(a, v)\}.$$

Intuitively, the function Post returns all possible subsets of states that can be *enforced* by coalition A from v, one for each action profile available to them.

The function BestX computes the *Best Next response*, according to a given evaluation function $\text{eval} : V \to [0, 1]$, defined as

$$\text{BestX}(A, v, \text{eval}) = \max_{S \in \text{Post}(A,v)} \min_{v' \in S} \{\text{eval}(v')\}$$

Algorithm 1 modelCheck($\mathcal{G}, \varphi$)

Input: wCGS $\mathcal{G} = (\text{Ac}, V, d, o, \ell)$; ATL[$\mathcal{F}$] formula φ.
Output: a function $\text{eval}(\varphi) : V \to [0, 1]$ such that $\text{eval}(\varphi)(v) = [\![\varphi]\!]^{\mathcal{G}}(v)$, for each $v \in V$.

1: **case** $\varphi = p \in \text{AP}$
2: **for** $v \in V$ **do**
3: $\text{eval}(\varphi)(v) := \ell(v, p)$
4: **case** $\varphi = f[\varphi_1, ..., \varphi_n]$
5: **for** $v \in V$ **do**
6: $\text{eval}(\varphi)(v) := f(\text{eval}(\varphi_1)(v), ..., \text{eval}(\varphi_n)(v))$
7: **case** $\varphi = \langle\langle A \rangle\rangle \mathbf{X}\psi$
8: **for** $v \in V$ **do**
9: $\text{eval}(\varphi)(v) := \text{BestX}(A, v, \text{eval}(\psi))$
10: **case** $\varphi = \langle\langle A \rangle\rangle \psi_1 \mathbf{U} \psi_2$
11: $\text{eval}(\varphi) := \text{eval}(\psi_2)$
12: **while** $\text{eval}(\varphi)$ changes **do**
13: **for** $v \in V$ **do**
14: **if** $\text{eval}(\varphi)(v) < \min\{\text{eval}(\psi_1)(v), \text{BestX}(A, v, \text{eval}(\varphi))\}$ **then** $\text{eval}(\varphi)(v) := \min\{\text{eval}(\psi_1)(v), \text{BestX}(A, v, \text{eval}(\varphi))\}$
15: **case** $\varphi = \langle\langle A \rangle\rangle \psi_1 \mathbf{R} \psi_2$
16: $\text{eval}(\varphi) := 1 - \text{eval}(\psi_2)$
17: **while** $\text{eval}(\varphi)$ changes **do**
18: **for** $v \in V$ **do**
19: **if** $\text{eval}(\varphi)(v) < 1 - \min\{\text{eval}(\psi_1)(v), \text{BestX}(A, v, \text{eval}(1 - \varphi))\}$ **then** $\text{eval}(\varphi)(v) := 1 - \min\{\text{eval}(1 - \psi_1)(v), \text{BestX}(A, v, \text{eval}(\varphi))\}$
return $\text{eval}(\varphi)$

Lemma 2. *For a given* ATL[$\mathcal{F}$] *formula* φ *and a wCGS* $\mathcal{G}$*, Algorithm 1 returns a function* $\text{eval}(\varphi)$ *such that*

$$\text{eval}(\varphi)(v) = [\![\varphi]\!]^{\mathcal{G}}(v) \text{ for each } v \in V$$

Proof. The proof proceeds by structural induction on the formula φ. The only nontrivial case is for $\varphi = \langle\langle A \rangle\rangle \psi_1 \mathbf{U} \psi_2$.

Since this case involves the activation of a while-loop, we first prove *termination*.

Let $\text{eval}_0(\varphi), \text{eval}_1(\varphi), ..., \text{eval}_k(\varphi), ...$ be the sequence of functions eval computed at each iteration of the while-loop. First observe that $\text{eval}_k(\varphi)(v)$ ranges in a finite set of values in $[0, 1]$. Moreover, because of lines 14 and 15 of the while-loop, we have that $\text{eval}_k(\varphi)(v) \leq \text{eval}_{k+1}(\varphi)(v)$ for each $v \in V$ and $k \in \mathbb{N}$. This means that $\text{eval}_k(\varphi)(v)$ is an non-decreasing sequence of reals, ranging in a finite set, which implies that there exists $f \in \mathbb{N}$ such that $\text{eval}_f(\varphi)(v) = \text{eval}_{f+1}(\varphi)(v)$, for each $v \in V$ and that the termination condition of the while-loop is always met.

It remains to prove that $\text{eval}_f(\varphi)(v) = [\![\varphi]\!]^{\mathcal{G}}(v)$ for each $v \in V$.

First, we prove, by induction, that $\text{eval}_k(\varphi)(v) \leq [\![\varphi]\!]^{\mathcal{G}}(v)$, for each $k \leq f$. As base case, first observe that

$$\langle\langle A \rangle\rangle \psi_1 \mathbf{U} \psi_2 \equiv \max\{\psi_2, \min\{\psi_1, \langle\langle A \rangle\rangle \mathbf{X} \langle\langle A \rangle\rangle \psi_1 \mathbf{U} \psi_2\}\}.$$

From which we obtain the following.

$$\begin{aligned}
\text{eval}_0(\varphi)(v) &= \text{eval}_0(\psi_2)(v) && (1)\\
&= [\![\psi_2]\!]^{\mathcal{G}}(v) && (2)\\
&\leq [\![\langle\langle A \rangle\rangle \psi_1 \mathbf{U} \psi_2]\!]^{\mathcal{G}}(v) && (3)
\end{aligned}$$

Where equality 2 holds by structural induction, while inequality 3 because of the equivalence mentioned above.

For the induction case, assume that $\text{eval}_k(\varphi)(v) \leq [\![\varphi]\!]^{\mathcal{G}}(v)$ for each $v \in V$. We have the following.

$$\begin{aligned}
\text{eval}_{k+1}(\varphi)(v) &\leq \min\{\text{eval}(\psi_1)(v), \text{BestX}(A, v, \text{eval}_k(\varphi))\} && (4)\\
&\leq \min\{\text{eval}(\psi_1)(v), \text{BestX}(A, v, [\![\varphi]\!]^{\mathcal{G}})\} && (5)\\
&= \min\{\text{eval}(\psi_1)(v), [\![\langle\langle A \rangle\rangle \mathbf{X}\varphi]\!]^{\mathcal{G}}(v)\} && (6)\\
&= \min\{[\![\psi_1]\!]^{\mathcal{G}}(v), \langle\langle A \rangle\rangle \mathbf{X}\varphi\} && (7)\\
&\leq \max\{[\![\psi_2]\!]^{\mathcal{G}}(v), \min\{[\![\psi_1]\!]^{\mathcal{G}}(v), \langle\langle A \rangle\rangle \mathbf{X}\varphi\}\} && (8)\\
&= [\![\varphi]\!]^{\mathcal{G}}(v) && (9)
\end{aligned}$$

Where

- Inequality 4 holds from the definition of the algorithm;
- Inequality 5 holds by induction hypothesis;
- Equality 6 holds from the definition of BestX;
- Equality 7 holds from the structural induction on φ;
- Inequality 8 holds from the definition of max;
- Equality 9 holds from the formula equivalence mentioned above.

Now, assume by contradiction that $\text{eval}_f(\varphi) \neq [\![\varphi]\!]^{\mathcal{G}}$ and consider a node $v \in V$ such that $\text{eval}_f(\varphi)(v) \neq [\![\varphi]\!]^{\mathcal{G}}(v)$ and $\text{eval}_f(\varphi)(v') = [\![\varphi]\!]^{\mathcal{G}}(v')$ for every other node v' such that $\text{eval}_f(\varphi)(v) < \text{eval}_f(\varphi)(v')$.

First, observe that $[\![\psi_2]\!]^{\mathcal{G}}(v) \lneq [\![\varphi]\!]^{\mathcal{G}}(v)$. Otherwise, it would hold that $[\![\psi_2]\!]^{\mathcal{G}}(v) = [\![\varphi]\!]^{\mathcal{G}}(v) \text{eval}(\psi_2)(v) \leq \text{eval}_f(\varphi)(v)$, and, for the inequality proved above, $[\![\psi_2]\!]^{\mathcal{G}}(v) = \text{eval}_f(\varphi)(v)$, which contradicts the assumption.

Therefore, we must have

$$[\![\varphi]\!]^{\mathcal{G}}(v) = \max\{[\![\psi_2]\!]^{\mathcal{G}}(v), \min\{[\![\psi_1]\!]^{\mathcal{G}}(v), \text{BestX}(A, v, [\![\varphi]\!]^{\mathcal{G}})\}\} \tag{10}$$

$$= \min\{[\![\psi_1]\!]^{\mathcal{G}}(v), \text{BestX}(A, v, [\![\varphi]\!]^{\mathcal{G}})\} \tag{11}$$

$$= \min\{\text{eval}(\psi_1)(v), \text{BestX}(A, v, \text{eval}_f(\varphi))\} \tag{12}$$

The last equivalence holds from the fact that we are looking at successors of v according to the best responses of coalition A, which must contain only nodes v' such that $[\![\varphi]\!]^{\mathcal{G}}(v) \leq [\![\varphi]\!]^{\mathcal{G}}(v')$.

Now, from line 14 of the algorithm, we obtain that the while loop is not yet terminated, which is a contradiction. Hence $\text{eval}_f(\varphi)(v) = [\![\varphi]\!]^{\mathcal{G}}(v)$. The case $\varphi = \langle\langle A\rangle\rangle \psi_1 \mathbf{R} \psi_2$ can be proved similarly to the previous, by just dualizing the arguments. □

We are now ready to prove the following.

Theorem 4.1 (Model Checking). *Assume that the functions in $\mathcal{F}$ are computable in polynomial time, for a given $\mathsf{ATL}^*[\mathcal{F}]$ formula $\varphi = \langle\langle A\rangle\rangle\psi$, a pointed wCGS $(\mathcal{G},v)$, and a predicate $P \subseteq [0,1]$, we have that:*

(1) Checking that $(\mathcal{G}, v)$ P-satisfies φ is 2Exptime-complete.
(2) If φ is an $\mathsf{ATL}[\mathcal{F}]$ formula, it is Ptime-complete.

Proof. We prove the two items separately.

(1) Regarding $\mathsf{ATL}^*[\mathcal{F}]$, consider the product $\mathcal{A} = \mathcal{A}^{A}_{\mathcal{G},v_\iota} \otimes \mathcal{A}^{\mathcal{V},P}_{\psi}$. This is an APT accepting those (AP, V)-labeled trees that are obtained from some strategy profile σ_A for coalition A (Lemma 1) such that each branch P-satisfies ψ (Proposition 1). This proves that the model-checking problem can be decided by solving the non-emptiness problem of such automaton. Due the size of $\mathcal{A}^{\mathcal{V},P}_{\psi}$, we obtain that $\mathcal{A}$ has doubly-exponentially many states, w.r.t the size of φ and exponentially many colors w.r.t. the size of φ, which in turns implies that the emptiness problem of $\mathcal{A}$ is 2Exptime w.r.t. φ. The lower-bound follows from the model checking of ATL^*, which is obviously reduced in linear time to the model checking of $\mathsf{ATL}^*[\mathcal{F}]$.
(2) Regarding $\mathsf{ATL}[\mathcal{F}]$, the lower bound is inherited from ATL [5]. The upper bound is a consequence of Lemma 2 and by noticing that the number of cycles in the while loops (Lines 12-14, 17-19) of Algorithm 1 is, in the worst case, quadratic on the number states in V. □

For module checking, instead, we need to account also for all possible pruning of the environment. We first provide the following.

Lemma 3. *For a given weighted CGS $(\mathcal{G}, v)$, an $\mathsf{ATL}^*[\mathcal{F}]$ formula $\varphi = \langle\langle A\rangle\rangle\psi$ and a set $P \subseteq [0,1]$, one can construct an NPT $\mathcal{A}^{\varphi,P}_{\mathcal{G},v_\iota}$ accepting all the $(\text{AP}, \text{Val}_{\mathcal{G}})$-labeled subtrees t_{σ_A,v_ι} of some environment strategy tree $\mathcal{T} \in \text{exec}(\mathcal{G})$ such that $[\![\varphi]\!]^{\mathcal{T}}(v) \in P$. The automaton $\mathcal{A}^{\varphi,P}_{\mathcal{G},v_\iota}$ has triply-exponentially many state w.r.t. the size of ψ and doubly-exponentially many colors w.r.t. the size of ψ.*

Proof sketch. Consider again the APT $\mathcal{A} = \mathcal{A}^{A}_{\mathcal{G},v_\iota} \otimes \mathcal{A}^{\mathcal{V},P}_{\psi}$. From Theorem 4.1, we obtain that this recognizes the labeled subtrees t of $\mathcal{G}$ that correspond to some strategy profile σ_A and P-satisfy ψ. We need to modify such automaton in order to recognize also those labeled subtrees t of some $\mathcal{T} \in \text{exec}(\mathcal{G}, v_\iota)$. Every subtree t can be equivalently represented as $(\text{AP}, \mathcal{V} \cup \{\bot\})$-labeled complete V-tree $t_\bot$, called $\bot$-completion encoding of t in which, each node u of t is labeled the same in $t_\bot$ and each node u in $t_\bot$ that does not belong to t is labeled with the special symbol $\bot$.

In [Theorem 4][12], the authors show that this can be done at the price of an exponential blow-up. Indeed, in order to perform such extension, one has to first nondeterminize the automaton $\mathcal{A}$ and then extend it in order to recognize the $\bot$-completion encoding. The resulting automaton (which is an NPT) thus recognizes those subtrees t of some strategy tree $\mathcal{T} \in \text{exec}(\mathcal{G})$ such that $[\![\varphi]\!]^{\mathcal{T}}(v) \in P$.

Given that the automaton $\mathcal{A}$ is already of double-exponential size w.r.t. φ, the resulting construction is of triple-exponential size. □

With Lemma 3, we have a machinery in place to solve module-checking. The following theorem holds.

Theorem 4.2 (Module Checking). *Assume that the functions in $\mathcal{F}$ are computable in polynomial time, for a given $\mathsf{ATL}^*[\mathcal{F}]$ formula φ, a pointed wCGS $(\mathcal{G}, v)$, and a predicate $P \subseteq [0,1]$, we have that:*

- *Checking that $(\mathcal{G}, v)$ reactively P-satisfies φ is 3Exptime-complete.*
- *If φ is an $\mathsf{ATL}[\mathcal{F}]$ formula, it is Exptime-complete.*

Proof. Consider the complement $\overline{P} = [0,1] \setminus P$ of P. Clearly, a weighted CGS $(\mathcal{G}, v)$ **does not** reactively P-satisfy an $\mathsf{ATL}[\mathcal{F}]$ formula φ iff there exists an environment strategy tree $\mathcal{T} \in \text{exec}(\mathcal{G})$ such that $[\![\varphi]\!]^{\mathcal{T}}(v) \in \overline{P}$. Therefore, we can reduce module checking to check the non-existence of such $\mathcal{T}$. Now, from Lemma 3, we can build the NPT $\mathcal{A}^{\varphi,\overline{P}}_{\mathcal{G},v_\iota}$ that accepts exactly those $(\text{AP}, \text{Val}_{\mathcal{G}})$-labeled subtrees t_{σ_A,v_ι} of some environment strategy tree $\mathcal{T} \in \text{exec}(\mathcal{G})$ such that $[\![\varphi]\!]^{\mathcal{T}}(v) \in \overline{P}$. This implies that, $\text{Ł}(\mathcal{A}^{\varphi,\overline{P}}_{\mathcal{G},v_\iota}) = \emptyset$ if, and only if, $(\mathcal{G}, v)$ reactively P-satisfies φ.

As for the complexity, we have that checking the emptiness of an NPT is polynomial in the number of states and exponential in the number of colors [38]. This returns the following complexity for module checking.

- If φ is an $\mathsf{ATL}^*[\mathcal{F}]$ formula, the size of the automaton is double-exponential in the size of φ and has exponentially many colors, resulting in 3EXPTIME procedure for the emptiness and, subsequently, for the module checking.
- If φ is an $\mathsf{ATL}[\mathcal{F}]$ formula, the size of the automaton is exponential in the size of φ and has polynomially many colors, resulting in a EXPTIME procedure for the emptiness, subsequently, for the module checking.

Both complexities are tight, as the lower-bounds of 3EXPTIME and EXPTIME for $\mathsf{ATL}^*[\mathcal{F}]$ and $\mathsf{ATL}[\mathcal{F}]$ follow from the complexity of Module checking ATL^* and ATL, respectively [12]. □

5 EXPRESSIVITY

We will now proceed to prove the following:

(1) $\mathsf{ATL}^*[\mathcal{F}]$ module checking is not subsumed by ATL^* module checking. That is, making module checking *quantitative* extends the set of verifiable properties.
(2) $\mathsf{ATL}^*[\mathcal{F}]$ module checking is not subsumed by $\mathsf{ATL}^*[\mathcal{F}]$ model checking. That is, switching from *model* to *module*

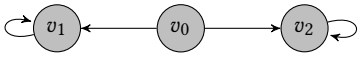

Figure 2: Structure of a module. All nodes owned by environment

checking allows us to capture quantitative properties that cannot be expressed otherwise.

The aim of this section is to show that ATL*[$\mathcal{F}$] module checking allows to capture and verify properties that cannot be captured by decision problems based on existing variants of alternating-time logics.

We first recall the concepts of expressive and distinguishing power for many-valued logics [9].

Definition 8 (Distinguishing power and expressive power). *Let $\mathcal{L}_1 = (L_1, [\![\cdot]\!]_1)$ and $\mathcal{L}_2 = (L_2, [\![\cdot]\!]_2)$ be two logical systems with syntax L_1, L_2 and real-valued semantics $[\![\cdot]\!]_1, [\![\cdot]\!]_2$ over the same class of structures $\mathcal{M}$. We say that $\mathcal{L}_2$ is* at least as distinguishing *as $\mathcal{L}_1$ (written: $\mathcal{L}_1 \preceq_d \mathcal{L}_2$) iff for every pair of structures $M, M' \in \mathcal{M}$, if there exists a formula $\varphi_1 \in L_1$ such that $[\![\varphi_1]\!]_1^M \neq [\![\varphi_1]\!]_1^{M'}$, then there is also $\varphi_2 \in L_2$ with $[\![\varphi_2]\!]_2^M \neq [\![\varphi_2]\!]_2^{M'}$. In other words, if there is a formula of $\mathcal{L}_1$ discerning M from M', then there must be also a formula of $\mathcal{L}_2$ doing the same.*

$\mathcal{L}_2$ is at least as expressive *as $\mathcal{L}_1$ (written: $\mathcal{L}_1 \preceq_e \mathcal{L}_2$) iff for every $\varphi_1 \in L_1$ there exists $\varphi_2 \in L_2$ such that, for every structure $M \in \mathcal{M}$, we have $[\![\varphi_1]\!]_1^M = [\![\varphi_2]\!]_2^M$. In other words, every formula of $\mathcal{L}_1$ has a translation in $\mathcal{L}_2$ that produces exactly the same truth values on the structures in $\mathcal{M}$.*

It is easy to see that $\mathcal{L}_1 \preceq_e \mathcal{L}_2$ implies $\mathcal{L}_1 \preceq_d \mathcal{L}_2$. Thus, by contraposition, we also get that $\mathcal{L}_1 \not\preceq_d \mathcal{L}_2$ implies $\mathcal{L}_1 \not\preceq_e \mathcal{L}_2$.

The above notions have been adapted in [9] to compare the expressivity of many-valued logics by comparing the truth values that formulas produce on the respective models. E.g., $L_2 \preceq_e L_1$ holds if every formula φ of L_2 has a counterpart in L_1 that, on each model, evaluates to exactly the same truth value as φ. Unfortunately, that approach cannot be applied in case (1), where we want to compare two logical systems with *different sets of truth values.*

5.1 Quantitative vs. Binary Module Checking

We first show that ATL*[$\mathcal{F}$] module checking is not subsumed by ATL* module checking over weighted modules. To do that, however, we need to fix how the *binary* reactive semantics of ATL* is used when the evaluation of atomic propositions is quantitative. We deal with the problem by adapting the concept of *designated truth values* $P \subseteq [0, 1]$ for which a formula is deemed satisfied [35, 48]. In line with standard practice, we assume that $1 \in P$ and $0 \notin P$.

Let $\mathcal{G}$ be a module. By $\mathcal{G}^P$, we denote $\mathcal{G}$ with the weight function changed to $\ell'(v, p) = 1$ if $\ell(v, p) \in P$, and 0 else, i.e., we "defuzzify" propositions via the designated values.

Theorem 5.1. *For every predicate $\{1\} \subseteq P \subseteq (0, 1]$, there is a pair of weighted modules $(\mathcal{G}_1, v_1)$ and $(\mathcal{G}_2, v_2)$ such that:*
(i) *for every $\varphi \in$ ATL*, we have $[\![\varphi]\!]^r_{\mathsf{ATL}^*}(\mathcal{G}_1^P, v_1) = [\![\varphi]\!]^r_{\mathsf{ATL}^*}(\mathcal{G}_2^P, v_2)$;*
(ii) *and, there exists a formula $\varphi \in$ ATL*[$\mathcal{F}$] with $[\![\varphi]\!]^r_{\mathsf{ATL}^*[\mathcal{F}]}(\mathcal{G}_1, v_1) \neq [\![\varphi]\!]^r_{\mathsf{ATL}^*[\mathcal{F}]}(\mathcal{G}_2, v_2)$.*

Proof. Fix an arbitrary P. There must be at least two different truth values $t_1 < t_2$ in P or in its complement $\overline{P}$. Consider modules $(\mathcal{G}_1, v_0)$ and $(\mathcal{G}_2, v_0)$, both with the structure depicted in Figure 2. The sole atomic proposition p evaluates to t_1 in all of $\mathcal{G}_1$, and to t_2 in all of $\mathcal{G}_2$. Clearly, $\mathcal{G}_1^P$ and $\mathcal{G}_2^P$ are the same, so they must satisfy the same properties of ATL*. On the other hand, $(\mathcal{G}_1, v_0)$ and $(\mathcal{G}_2, v_0)$ are distinguished by the reactive semantics of $\langle\!\langle\emptyset\rangle\!\rangle \mathbf{X} p$ in ATL*[$\mathcal{F}$]. □

5.2 Quantitative Module vs. Model Checking

We now prove that ATL*[$\mathcal{F}$] *module* checking is not subsumed by ATL*[$\mathcal{F}$] *model* checking.

Theorem 5.2. *There is a pair of weighted modules $(\mathcal{G}_1, v_1), (\mathcal{G}_2, v_2)$ such that:*
(i) *for every $\varphi \in$ ATL*[$\mathcal{F}$], we have $[\![\varphi]\!]_{\mathsf{ATL}^*[\mathcal{F}]}(\mathcal{G}_1, v_1) = [\![\varphi]\!]_{\mathsf{ATL}^*[\mathcal{F}]}(\mathcal{G}_2, v_2)$; and*
(ii) *there exists a formula $\varphi \in$ ATL*[$\mathcal{F}$] with $[\![\varphi]\!]^r_{\mathsf{ATL}^*[\mathcal{F}]}(\mathcal{G}_1, v_1) \neq [\![\varphi]\!]^r_{\mathsf{ATL}^*[\mathcal{F}]}(\mathcal{G}_2, v_2)$.*

Proof. It follows directly from the analogous property for binary module checking, see [Thm 4][34] and [Thm 1][33]. □

6 CONCLUSION

In this paper, we have addressed the problem of specifying and verifying MAS interacting with a nondeterministic environment and whose quality cannot be reduced to a Boolean assessment. Here, the environment represents an authority that can inhibit the MAS access to certain paths of the computation tree. The assessment of the system is quantitative and represents how much the authority can limit the agents in the MAS from achieving their goals.

As a modeling solution, we have proposed quantitative module checking in relation to specifications given in ATL*[$\mathcal{F}$] and ATL[$\mathcal{F}$], the quantitative extensions of ATL* and ATL. This allows reasoning about quality in multi-agent systems that interact with an environment. Remarkably, no other modeling of module checking studied in the literature can handle the quantitative setting, which we propose and whose solution comes at no extra cost. To maintain the same computational complexity we have used a parsimonious automata-theoretic approach, from scratch.

We studied the complexity and expressivity of module checking ATL*[$\mathcal{F}$] and ATL[$\mathcal{F}$]-specifications. In relation to expressivity, we show that ATL*[$\mathcal{F}$] module checking is not subsumed neither by ATL* module checking nor by ATL*[$\mathcal{F}$] model checking.

ACKNOWLEDGMENTS

This research has been supported by the EU Horizon 2020 Marie Skłodowska-Curie project with grant agreement No 101105549, the PNRR MUR project PE0000013-FAIR, as well as by NCBR Poland and FNR Luxembourg under the PolLux/FNR-CORE project SpaceVote (POLLUX-XI/14/SpaceVote/2023 and C22/IS/17232062/ SpaceVote). The work was partially funded also by MUR under the PRIN 2020 projects PINPOINT and RIPER. For the purpose of open access, and in fulfilment of the obligations arising from the grant agreement, the authors have applied CC BY 4.0 license to any Author Accepted Manuscript version arising from this submission.

REFERENCES

[1] T. Ågotnes, W. Van Der Hoek, J. A. Rodríguez-Aguilar, C. Sierra, and M. J. Wooldridge. 2007. On the Logic of Normative Systems.. In *IJCAI*.

[2] Thomas Ågotnes, Wiebe Van der Hoek, and Michael Wooldridge. 2010. Robust normative systems and a logic of norm compliance. *Logic Journal of IGPL* 18, 1 (2010), 4–30.

[3] Natasha Alechina, Giuseppe De Giacomo, Brian Logan, and Giuseppe Perelli. 2022. Automatic Synthesis of Dynamic Norms for Multi-Agent Systems. In *Proc. of KR 2022*. https://doi.org/10.24963/kr.2022/2

[4] Shaull Almagor, Udi Boker, and Orna Kupferman. 2016. Formally reasoning about quality. *Journal of the ACM (JACM)* 63, 3 (2016), 1–56.

[5] Rajeev Alur, Thomas A. Henzinger, and Orna Kupferman. 2002. Alternating-time temporal logic. *J. ACM* 49, 5 (2002), 672–713. https://doi.org/10.1145/585265.585270

[6] Benjamin Aminof, Axel Legay, Aniello Murano, Olivier Serre, and Moshe Y. Vardi. 2013. Pushdown module checking with imperfect information. *Inf. Comput.* 223 (2013), 1–17. https://doi.org/10.1016/j.ic.2012.11.005

[7] Saeed Asadi Bagloee, Madjid Tavana, Mohsen Asadi, and Tracey Oliver. 2016. Autonomous vehicles: challenges, opportunities, and future implications for transportation policies. *Journal of modern transportation* 24, 4 (2016), 284–303.

[8] Samik Basu, Partha S. Roop, and Roopak Sinha. 2007. Local Module Checking for CTL Specifications. *Electron. Notes Theor. Comput. Sci.* 176, 2 (2007), 125–141.

[9] Francesco Belardinelli, Wojtek Jamroga, Vadim Malvone, Munyque Mittelmann, Aniello Murano, and Laurent Perrussel. 2022. Reasoning about Human-Friendly Strategies in Repeated Keyword Auctions. In *AAMAS*. 62–71.

[10] Patricia Bouyer, Orna Kupferman, Nicholas Markey, Bastien Maubert, Aniello Murano, and Giuseppe Perelli. 2019. Reasoning about Quality and Fuzziness of Strategic Behaviours. In *IJCAI*. https://doi.org/10.24963/ijcai.2019/220

[11] Laura Bozzelli. 2011. New results on pushdown module checking with imperfect information. In *Proc. of the 2nd International Symposium on Games, Automata, Logics and Formal Verification, GandALF 2011 (EPTCS, Vol. 54)*. 162–177. https://doi.org/10.4204/EPTCS.54.12

[12] Laura Bozzelli and Aniello Murano. 2017. On the Complexity of ATL and ATL* Module Checking. In *Proc of the 8th International Symposium on Games, Automata, Logics and Formal Verification, GandALF 2017 (EPTCS, Vol. 256)*. 268–282. https://doi.org/10.4204/EPTCS.256.19

[13] Laura Bozzelli, Aniello Murano, and Adriano Peron. 2010. Pushdown module checking. *Formal Methods Syst. Des.* 36, 1 (2010), 65–95. https://doi.org/10.1007/s10703-010-0093-x

[14] Thomas Brihaye, François Laroussinie, Nicolas Markey, and Ghassan Oreiby. 2007. Timed Concurrent Game Structures. In *Proc. of CONCUR 2007*. https://doi.org/10.1007/978-3-540-74407-8_30

[15] Nils Bulling and Mehdi Dastani. 2016. Norm-based mechanism design. *Artificial Intelligence* 239 (2016), 97–142.

[16] Nils Bulling, Mehdi Dastani, and Max Knobbout. 2013. Monitoring norm violations in multi-agent systems. In *Proc. of AAMAS 2013*.

[17] Nils Bulling and Valentin Goranko. 2022. Combining quantitative and qualitative reasoning in concurrent multi-player games. *Auton. Agents Multi Agent Syst.* 36, 1 (2022), 2. https://doi.org/10.1007/s10458-021-09531-9

[18] Krishnendu Chatterjee, Thomas A. Henzinger, and Nir Piterman. 2010. Strategy Logic. *Inf. Comput.* 208, 6 (2010), 677–693. https://doi.org/10.1016/j.ic.2009.07.004

[19] Edmund M. Clarke, Thomas A. Henzinger, and Helmut Veith. 2018. *Introduction to Model Checking*. Springer International Publishing, Cham, 1–26. https://doi.org/10.1007/978-3-319-10575-8_1

[20] Mehdi Dastani, John-Jules Ch Meyer, and Davide Grossi. 2013. A logic for normative multi-agent programs. *Journal of Logic and Computation* 23, 2 (2013), 335–354.

[21] Luca de Alfaro, Patrice Godefroid, and Radha Jagadeesan. 2004. Three-Valued Abstractions of Games: Uncertainty, but with Precision. In *LICS*. IEEE Computer Society, 170–179.

[22] Louise Dennis, Nick Tinnemeier, and John-Jules Meyer. 2010. Model checking normative agent organisations. In *Int. Workshop on Computational Logic in MAS*. Springer, 64–82.

[23] Benjamin Edelman, Michael Ostrovsky, and Michael Schwarz. 2007. Internet advertising and the generalized second-price auction: Selling billions of dollars worth of keywords. *American economic review* 97, 1 (2007), 242–259.

[24] Edith Elkind, Leslie Ann Goldberg, Paul W Goldberg, and Michael Wooldridge. 2009. On the computational complexity of weighted voting games. *Annals of Mathematics and Artificial Intelligence* 56, 2 (2009), 109–131.

[25] E. Allen Emerson and Joseph Y. Halpern. 1986. "Sometimes" and "Not Never" Revisited: On Branching versus Linear Time Temporal Logic. *J. ACM* 33, 1 (jan 1986), 151–178. https://doi.org/10.1145/4904.4999

[26] Alessandro Ferrante, Aniello Murano, and Mimmo Parente. 2008. Enriched μ-Calculi Module Checking. *Log. Methods Comput. Sci.* 4, 3 (2008).

[27] Achille Frigeri, Liliana Pasquale, and Paola Spoletini. 2014. Fuzzy time in linear temporal logic. *ACM Transactions on Computational Logic (TOCL)* 15, 4 (2014), 1–22.

[28] Anna Gibson. 2019. Free speech and safe spaces: How moderation policies shape online discussion spaces. *Social Media+ Society* 5, 1 (2019), 2056305119832588.

[29] Patrice Godefroid. 2003. Reasoning about Abstract Open Systems with Generalized Module Checking. In *Proc. of EMSOFT 2003*.

[30] Erich Grädel, Wolfgang Thomas, and Thomas Wilke (Eds.). 2002. *Automata, Logics, and Infinite Games: A Guide to Current Research [outcome of a Dagstuhl seminar, February 2001]*. Springer. https://doi.org/10.1007/3-540-36387-4

[31] Thomas A. Henzinger and Vinayak S. Prabhu. 2006. Timed Alternating-Time Temporal Logic. In *Proc. of FORMATS 2006*. https://doi.org/10.1007/11867340_1

[32] Wojciech Jamroga, Beata Konikowska, Damian Kurpiewski, and Wojciech Penczek. 2020. Multi-valued Verification of Strategic Ability. *Fundam. Informaticae* 175, 1-4 (2020), 207–251. https://doi.org/10.3233/FI-2020-1955

[33] Wojciech Jamroga and Aniello Murano. 2014. On module checking and strategies. In *AAMAS*. IFAAMAS/ACM, 701–708.

[34] Wojciech Jamroga and Aniello Murano. 2015. Module Checking of Strategic Ability. In *Proc. of AAMAS 2015*. ACM, 227–235.

[35] Beata Konikowska and Wojciech Penczek. 2004. On Designated Values in Multi-valued CTL* Model Checking. *Fundamenta Informaticae* 60, 1-4 (2004), 211–224.

[36] Orna Kupferman and Moshe Y. Vardi. 1996. Module checking. In *Proc. of CAV 1996*, Rajeev Alur and Thomas A. Henzinger (Eds.).

[37] Orna Kupferman and Moshe Y. Vardi. 1997. Module Checking Revisited. In *Proc. of CAV 1997*.

[38] Orna Kupferman and Moshe Y. Vardi. 1998. Weak Alternating Automata and Tree Automata Emptiness. In *Proc. of the ACM Symposium on the Theory of Computing*, Jeffrey Scott Vitter (Ed.). https://doi.org/10.1145/276698.276748

[39] Orna Kupferman, Moshe Y. Vardi, and Pierre Wolper. 2000. An automata-theoretic approach to branching-time model checking. *J. ACM* 47, 2 (2000), 312–360. https://doi.org/10.1145/333979.333987

[40] Orna Kupferman, Moshe Y. Vardi, and Pierre Wolper. 2001. Module Checking. *Inf. Comput.* 164, 2 (2001), 322–344. https://doi.org/10.1006/inco.2000.2893

[41] Khaled Ben Lamine and Froduald Kabanza. 2000. Using fuzzy temporal logic for monitoring behavior-based mobile robots. In *Proc. of IASTED Int. Conf. on Robotics and Applications*. 116–121.

[42] François Laroussinie, Nicolas Markey, and Ghassan Oreiby. 2006. Model-Checking Timed. In *Proc. FORMATS 2006*. https://doi.org/10.1007/11867340_18

[43] Bastien Maubert, Munyque Mittelmann, Aniello Murano, and Laurent Perrussel. 2021. Strategic reasoning in automated mechanism design. In *Proc. of the KR 2021*.

[44] John-Jules Ch Meyer. 1993. Deontic logic: A concise overview. *Deontic Logic in Computer Science: Normative System Specification* (1993), 3–16.

[45] Fabio Mogavero, Aniello Murano, Giuseppe Perelli, and Moshe Y. Vardi. 2014. Reasoning About Strategies: On the Model-Checking Problem. *ACM Trans. Comput. Log.* 15, 4 (2014).

[46] Aniello Murano, Margherita Napoli, and Mimmo Parente. 2008. Program Complexity in Hierarchical Module Checking. In *In Proc. of LPAR 2008*. https://doi.org/10.1007/978-3-540-89439-1_23

[47] Amir Pnueli. 1977. The temporal logic of programs. In *Proc. of the Annual Symposium on Foundations of Computer Science*. https://doi.org/10.1109/SFCS.1977.32

[48] Yaroslav Shramko and Heinrich Wansing. 2021. Truth Values. In *The Stanford Encyclopedia of Philosophy* (Winter 2021 ed.), Edward N. Zalta (Ed.). Metaphysics Research Lab, Stanford University.

[49] Steen Vester. 2015. On the Complexity of Model-Checking Branching and Alternating-Time Temporal Logics in One-Counter Systemss. In *Automated Technology for Verification and Analysis*, Bernd Finkbeiner, Geguang Pu, and Lijun Zhang (Eds.). Springer.