Parity-energy ATL for Qualitative and Quantitative Reasoning in MAS

Dario Della Monica
Istituto Nazionale di Alta Matematica “F. Severi” (Italy)

ABSTRACT

In this paper, we introduce a new logic suitable to reason about strategic abilities of multi-agent systems where (teams of) agents are subject to qualitative (parity) and quantitative (energy) constraints and where goals are represented, as usual, by means of temporal properties. We formally define such a logic, named parity-energy-ATL (pe-ATL, for short), and we study its model checking problem, which we prove to be decidable with different complexity upper bounds, depending on different choices for the energy range.

ACM Reference Format:

1 INTRODUCTION

In recent years, game theory has been demonstrated to be very useful in open-system verification, where the game evolution emerges from the coordination of different parts viewed as autonomous and proactive agents [13, 22]. This has encouraged the development of several frameworks aimed at reasoning about strategies and their interaction [3, 9, 17, 19, 20, 23, 25, 27].

An important contribution in this field has been the development of Alternating-Time Temporal Logic (ATL, for short) by Alur, Henzinger, and Kupferman [3]. Formally, it is obtained as a generalization of the branching-time logic CTL [12], where the path quantifiers there exists “E” and for all “A” are replaced with strategic modalities of the form “⟨⟨A⟩⟩” and “[A]”, for a set A of agents. These modalities are used to express cooperation and competition among agents in order to achieve a temporal goal. Several decision problems have been investigated about ATL. In particular, the model checking problem is proved to be solvable in polynomial time [3].

ATL can efficiently express qualitative objectives, that is, specific temporal properties that are required to hold by coalitions of agents. Conversely, ATL (at least in the classical definition) cannot be used to express quantitative objectives. This is unfortunate as games dealing with quantitative aspects are recently receiving a lot of attention in the context of automated design and synthesis [5, 29]. Even more, quantitative games are central in economics, where players aim at optimizing a desired payoff.

In the context of games equipped with quantitative objectives, an important contribution is given by energy parity games [10].

These are games played on weighted graphs in which (i) states are partitioned between two players, namely player 0 and player 1, (ii) a priority is associated to each state, and (iii) an integer weight is associated to each edge. In each round, the player owning the state chooses an outgoing edge to a successor state. The play consisting of an infinite number of rounds is won by player 0 if (i) the least priority occurring infinitely often is even and (2) the sum of the weights (namely the energy) along the play remains always positive. Deciding an energy parity game amounts to checking if there is an initial energy level such that player 0 has a strategy to maintain the level of energy positive while satisfying the parity condition. This problem is known to lie in NP ∩ coNP, as it is for parity games.

In this paper we combine energy parity games and ATL specifications in a new logical formalisms, named parity-energy-ATL (pe-ATL), and we solve the related model checking question. Roughly speaking, pe-ATL allows one to check the satisfaction of a parity condition while keeping the energy level within a given range along system evolutions determined by coalitions along the ATL formula. We show that the addressed model checking question lies in P, NP, or ExpTime, depending on the type of energy range given in input.

The conceived framework can be successfully used in several contexts. In practice, it can be used in smart-city applications, which are multi-agent systems where one has to deal with both energy constraints and temporal goals [8, 26]. As another application scenario, consider a multi-agent system model for task allocation in which every odd priority represents a task request, whose allocation is represented by an even smaller priority (similar to what happens in prompt parity games [4, 15, 24]). Assume now that along the process we need to use, as well as recharge, some working energy, represented by the weights over the edges. Finally, assume that we want to check that no matter which tasks are supplied by a set of Request agents, the Allocation agents can always guarantee a correct allocation. Then, such a scenario can be easily modeled and verified using our setting. Notice that fairness requirements such as the above one can be expressed by ATL* formulas, at the price of a more expensive model checking procedure (2ExpTime-complete).

Related works. The interest in combining qualitative and quantitative reasoning in multi-agent systems has grown in the last decade. In [7], a logical framework combining agents abilities to achieve quantitative and qualitative objectives is proposed. In [5, 11], the authors propose the use of preference models that combine qualitative and quantitative objectives. More recently, in [18] a well-behaved model combining qualitative ( Büchi) and quantitative (energy) conditions in a lexicographic setting is proposed. Finally, we mention a line of work on rational verification/synthesis [2, 16, 21, 28], where agents are assumed to be rational, i.e., they act according to equilibrium notions, but have no abilities of joining in coalitions.

However, the setting we investigate is new. The logic proposed in [7] flows quickly into undecidability. The other aforementioned works adopt a game-theoretic perspective, mostly focusing on the
We fix a finite, non-empty set $N$ and $\rho$ from $A$ towards solving the model checking problem. In Section 5, we put our results together to provide a solution and a complexity analysis for the pe-ATL model checking problem, and we outline future research directions.

2 pe-ATL: QUALITATIVE AND QUANTITATIVE REASONING IN MAS

We denote by $\mathbb{N}$ the set of natural numbers, by $\mathbb{N}^0$ the set of positive numbers, i.e., $\mathbb{N}^0 = \mathbb{N} \setminus \{0\}$, and by $S^0$ the set of vectors of $n$ components ranging over $S$, for every set $S$ and $n \in \mathbb{N}^0$. According to the standard notation, for any given set $S$ we denote by $S^*_r$ (resp., $S^o_r$) the set of finite (resp., infinite) words over the alphabet $S$. The length of a word $\rho \in S^* \cup S^o$ is denoted by $|\rho|$ (we assume $|\rho| = +\infty$ for $\rho \in S^o$). Finally, for every $\rho \in S^* \cup S^o$ and $i, j \in \mathbb{N}^0$, with $i \leq j \leq |\rho|$, we write $\rho[i]$ to refer to the $i$th element of $\rho$ and $\rho[j]$ to refer to the finite sequence $\rho[i], \rho[i+1], \ldots, \rho[j-1], \rho[j]$ in $S^*$. 

2.1 Concurrent game structures and strategies

We fix a finite, non-empty set $Agt = \{a_1, \ldots, a_n\}$ of $n$ agents and a finite set $A$ of atomic propositions. A subset of $Agt$ is called team.

Definition 2.1 (CS). A concurrent game structure (CS) $G$ over $Agt$ and $A$ is a tuple $(Q, q^{\text{init}}, \pi, d, \delta)$, where:
- $Q$ is the finite set of states;
- $q^{\text{init}} \in Q$ is a distinguished state in $Q$, called initial state;
- $\pi : A \rightarrow 2^Q$ is the evaluation function, which determines the states where propositions hold true;
- $d : Q \times Agt \rightarrow 2^{\mathbb{N}^0}$ is the action function: $d(q, a)$ denotes the (number of) actions available to an agent $a \in Agt$ at a state $q \in Q$. For each $q \in Q$ we denote by $D(q)$ the set $\{1, \ldots, d(q, a_1)\} \times \ldots \times \{1, \ldots, d(q, a_n)\}$ of action profiles available at $q$. A generic action (resp., action profile) is usually denoted by $a$ (resp., $\vec{a}$);
- $\delta : Q \times \mathbb{N}^0 \rightarrow Q$ is the transition function, defined on pairs $(q, \vec{a}) \in Q \times \mathbb{N}^0$ with $\vec{a} \in D(q)$ and undefined when $\vec{a} \notin D(q)$. If $\delta(q, \vec{a}) = q'$ for some $q, q' \in Q$ and $\vec{a} \in D(q)$, we say that there is a transition from $q$ to $q'$ triggered by the action profile $\vec{a}$ (denoted by $q \xrightarrow{\vec{a}} q'$).

For every CS $G$, we denote by $Q_G$, $q^{\text{init}}_G$, $\pi_G$, $d_G$, and $\delta_G$ its components (we omit the subscript when it is clear from the context).

Let $G$ be a CS. We generalize the notion of action profile to any given team: an $A$-action profile, with $A \subseteq Agt$, is a function from $A$ to $\mathbb{N}^0$. We denote by $A_A$ the set of $A$-action profiles, and we usually use $\vec{a}_A$ to denote a generic element of $A_A$. Moreover, for every $q \in Q$ and $A \subseteq Agt$, we denote by $D_A(q)$ the set of $A$-action profiles available at $q$, that is, $D_A(q) = \{\vec{a} \in A_A \mid \vec{a}(a) \leq d(q, a)\}$ for every $a \in A$.

A computation (over a CS $G$) is an infinite sequence of pairs $(q_1, \vec{a}_1), (q_2, \vec{a}_2), \ldots$ such that $q_i \xrightarrow{\vec{a}_i} q_{i+1}$ for every $i$. If, in addition, $q_1 = q^{\text{init}}$, then the computation is initial. We denote by $C_G$ the set of all computations over $G$ (once again, we omit the subscript when there is no ambiguity). The $Q$-projection of a computation $c = (q_1, \vec{a}_1)(q_2, \vec{a}_2) \ldots \in C$, denoted by $c_Q$, is the infinite sequence of states $q_1q_2\ldots$. A history is a pair $(c, q)$, where $c = (q_1, \vec{a}_1)(q_2, \vec{a}_2) \ldots (q_k, \vec{a}_k)$ is a (possibly empty) prefix of a computation and $q \in Q$ such that $q_k \xrightarrow{\vec{a}_k} q$, unless $c$ is the empty sequence; we denote by $H$ the set of histories.

Definition 2.2 ((memoryless) strategy). A strategy (for a team $A$ over a CS $G$) is a function $F_A : \mathbb{H} \rightarrow A_A$ such that $F_A(c, q) \in D_A(q)$, for every $(c, q) \in \mathbb{H}$. A memoryless strategy is a strategy $F_A$ such that $F_A(c, q) = F_A(c', q)$ for every $c, c'$, and $q$.

Let $\vec{a}_A$ be an $A$-action profile, i.e., $\vec{a}_A \in A_A$. We define the set of its extensions as $\text{ext}(\vec{a}_A) = \{\vec{a} \in A_A \mid \vec{a}(a) = \vec{a}(a)\}$ for every $a \in A$, and, for every $q \in Q$, we let $\delta_{\vec{a}_A}(q) = \{q' \in Q \mid q' = \delta(q, \vec{a})\}$ for some $\vec{a} \in \text{ext}(\vec{a}_A) \cap D(q)$.

Definition 2.3 (outcome). The outcome of a strategy $F_A$ from a state $q \in Q$ is the set out($F_A, q$) = $\{(q_1, \vec{a}_1)(q_2, \vec{a}_2) \ldots \in C \mid q_1 = q\}$ and $\vec{a}_1 \in \text{ext}(F_A(q_1, \vec{a}_1)(q_2, \vec{a}_2) \ldots (q_{i-1}, \vec{a}_{i-1}, q_i)) \cap D(q_i)$ for all $i \geq 0$.

Let $i, g \in Q$ (elements of $i$ are called invariants while elements of $g$ are called goals). A strategy $F_A$ guarantees $i$ from $q \in Q$ to mean that, for every $c \in \text{out}(F_A, q)$, $c|_i$ only features invariants $(q'< i, \text{for every } q'< i\text{ occurring in } c|_i)$ before the first occurrence of a state in $g$; $F_A$ guarantees $i$ until $g$ from $q \in Q$ to mean that, for every $c \in \text{out}(F_A, q)$, $c|_i$ features at least one goal (state in $g$) and is such that only invariants occur in $c|_i$ before the first occurrence of a goal ($q' \in i, \text{for every } q'< i\text{ occurring in } c|_i$ before the first occurrence of a state in $g$); finally, we say that $F_A$ is $(i, g)$-friendly (with $o \in \{U, E\}$) from $q \in Q$ if:
- $\top = \square$ and $F_A$ guarantees $i$, or
- $\top = U$ and $F_A$ guarantees $i$ until $g$.

Without loss of generality, from now on we assume $i \cap g = \emptyset$.

In what follows, we refine the notions of CS and corresponding strategies to comply with qualitative (parity condition) and quantitative (energy condition) requirements.

Definition 2.4 (parity condition). A parity condition (over a CS $G$) is a function $p : Q \rightarrow \mathbb{N}$ assigning natural numbers to states in $G$.

Energy conditions are based on weight assignments. A weight assignment (over a CS $G$) is a function $w : Q \times Agt \rightarrow \mathbb{R}$ assigning a rational weight to every transition of $G$ ($w(q, \vec{a})$ is undefined whenever $\vec{a} \notin D(q)$). For every $x, y \in Q \cup \{-\infty, +\infty\}$, we denote by $[x, y]$ the set $\{z \in \mathbb{R} \mid x \leq z \leq y\}$.

Definition 2.5 (energy condition). An energy condition (over a CS $G$) is a triple $e = (\omega, E^{\text{init}}, [a, b])$, where $\omega$ is a weight assignment over $G$, $E^{\text{init}} \in [a, b]$ is the initial energy level of $e$, and $[a, b]$ is its energy bound, with $a \in Q \cup \{-\infty\}$, $b \in Q \cup \{+\infty\}$, and $a \leq b$.

Definition 2.6 (pe-CEGS). A parity-energy CS (pe-CEGS) is a triple $G = (G, p, e)$, where $G$ is a CS, and $p$ and $e$ are $\omega, E^{\text{init}}, [a, b]$ are, respectively, a parity and an energy condition over it. A position of $G$ is a pair $(q, E) \in Q \times Q; (E^{\text{init}}, E^{\text{init}})$ is the initial position of $G$.

We lift any given parity condition $p$ from the domain $Q$ to the domain $Q^o$ in the natural way, by defining $\hat{p} : Q^o \rightarrow \mathbb{N}^o$ as $\hat{p}(q_1q_2\ldots) = p(q_1)p(q_2)\ldots \in \mathbb{N}^o$ for every $q_1q_2\ldots \in Q^o$, Moreover, for $p \in Q^o$, we let infinite$_p(\rho)$ be the set of naturals that
occurs infinitely many times in $\hat{p}(\rho)$. The parity of an infinite word $\rho \in Q^{\omega}$ wrt $p$ is defined as $\min(\infty(\rho), p))$.

Let $e = (w, E^{\text{init}}[a, b])$ be an energy condition. For every computation $c = (q_0, A, \delta(q_0, \delta_2) \ldots)$, the energy-contribution of a prefix $c[1:k]$ of $e$ is $\sum_{i=1}^{k} \delta(q_i, \delta_2)$; the energy bottom of $c$ e, denoted by $e$, is defined as $\sum_{i=1}^{k} \delta(q_i, \delta_2)$; the energy peak of $c$ e is defined analogously: $\inf_{k \in \omega} e - e\delta(q_i, \delta_2)$. Finally, for any given strategy $F_A$ and $q \in Q$, the energy range of $F_A$ from $q$ wrt $e$, denoted by $e\delta(q, F_A, q)$, is the set $[\min, \max]$, where $\min = \max = \inf(e, c')$, $c' \in out(F_A, q)$.

We are now ready to define strategies that are compliant with parity- and/or energy-conditions.

**Definition 2.7** ($\langle p,e \rangle$-strategy). Let $G = (G, p, e)$ be a pe-CGS, with $e = (w, E^{\text{init}}[a, b])$ and $q \in Q$. A strategy $F_A$ is said to be:

- $p$-compliant (or a $p$-strategy) from $q$ if for every $c \in out(F_A, q)$, $c_{i_0} \in e$ has parity wrt $p$.
- $e$-compliant (or an $e$-strategy) from $q$ if its energy range from $q$ wrt $e$ is within the energy bound of, i.e., $e\delta(q, F_A, q) \subseteq [a,b]$.
- $(p,e)$-compliant (or a $(p,e)$-strategy) from $q$ if it is both $p$- and $e$-compliant from $q$.

Let $G = (G, p, e)$ be a pe-CGS with $e = (w, E^{\text{init}}[a, b])$. Before concluding this subsection, we observe that a history $(c, e)$ univocally identifies $G$ the position $(q, E)$, where $E = E^{\text{init}} + e\delta(c, e)$; therefore, we say that history $(c, E)$ leads (in $G$) to position $(q, E)$.

### 2.2 Syntax and semantics

The syntax of pe-ATL is the same as the one for ATL [3], and it is given by the following grammar:

$$\varphi ::= p \mid \neg \varphi \mid \varphi \land \varphi \mid \langle A \rangle \varphi \mid \langle A \rangle \neg \varphi \mid \langle A \rangle \varphi \langle U \rangle \varphi \mid \langle A \rangle \varphi \langle U \rangle \neg \varphi$$

where $p \in A$ and $A \subseteq A_{\text{Agnt}}$. Constants $\top$ and $\bot$, as well as other Boolean connectives and team operators (e.g., $\lor$ and $\langle A \rangle \varphi$), can be seen as abbreviations (e.g., $\langle A \rangle \varphi \langle U \rangle \varphi$ is a shorthand for $\langle A \rangle \varphi \langle U \rangle \varphi$).

Formulas of pe-ATL are interpreted wrt states of pe-CGS. Let $G = (G, p, e)$ be a pe-CGS and $q \in Q$. The truth of a pe-ATL formula $G$ over $q$ and $q$ is inductively defined by the following clauses:

- $G, q, \varphi$ if $q \in p(i)$;
- $G, q, $ if it is not the case that $G, q, \varphi$;
- $G, q, \varphi \land \varphi$ if $G, q, \varphi$ and $G, q, \varphi$;
- $G, q, \langle A \rangle \varphi$ if there exists a $(p,e)$-strategy $F_A$ from $q$ such that $G, c_{i_0}[2] = \varphi$ for every $c \in out(F_A, q)$;
- $G, q, \langle A \rangle \varphi \langle U \rangle \varphi$ if there exists a $(p,e)$-strategy $F_A$ from $q$ such that for every $c \in out(F_A, q)$ there is $c_{i_0}[1] = \varphi$ and for every $j \in N$ with $1 \leq j < i$ it holds $G, c_{i_0}[j] = \varphi$;
- $G, q, \langle A \rangle \varphi \langle U \rangle \neg \varphi$ if there exists a $(p,e)$-strategy $F_A$ from $q$ such that for every $c \in out(F_A, q)$ and every $i \in N$ it holds $G, c_{i_0}[i] = \varphi$.

Using standard notation, given a pe-ATL formula $\varphi$ and a pe-CGS $G$, we write $\varphi|_{G}$ to denote the set $\{q \mid G, q, \varphi\}$, and we omit the subscript when there is no risk of ambiguity, thus writing, e.g., $\varphi$ instead of $\varphi|_{G}$. Therefore, the clauses for the operators $\mathcal{U}$ and $\square$ can be rewritten as follows:

- $G, q, \langle A \rangle \varphi \langle U \rangle \neg \varphi$ if there exists a $([p_1[1]] \setminus [p_2[1]] \setminus [p_2[1]] \setminus [p_2[1]])$-friendly $(p,e)$-strategy $F_A$ from $q$.
- $G, q, \langle A \rangle \varphi \langle U \rangle \neg \varphi$ if there exists a $([p_1[1]] \setminus [p_2[1]] \setminus [p_2[1]] \setminus [p_2[1]])$-friendly $(p,e)$-strategy $F_A$ from $q$.

By simultaneously replacing $\models_p$ for $\models$ and $p$-strategy for $(p,e)$-strategy, or, alternatively, $\models_e$ for $\models$ and $e$-strategy for $(p,e)$-strategy, we obtain two alternative semantics, the $p$- and the $e$-semantics. We name the resulting logics p-ATL and e-ATL, respectively.

Given a pe-ATL formula $\psi$ and a pe-CGS $G$, we say that $G$ satisfies $\psi$, denoted by $G \models \psi$, if $G, q^{\text{init}} \models \psi$. Moreover, we say that $G$ satisfies $\psi$ in the $p$-semantics (resp., e-semantics), denoted by $G \models_p \psi$ (resp., $G \models_e \psi$), if, and only if, $G, q^{\text{init}} \models_p \psi$ (resp., $G, q^{\text{init}} \models_e \psi$).

### 2.3 The model checking problem

The model checking problem for pe-ATL (resp., p-ATL, e-ATL) consists in verifying, given a pe-CGS $G$ and a pe-ATL formula $\psi$, whether $G \models \psi$ (resp., $G \models_p \psi$, $G \models_e \psi$) holds. It is easy to show that the model checking problem for pe-ATL (resp., e-ATL) can be reduced to the model checking problem for p-ATL. Indeed, for a $G = (G, p, e)$, with $e = (w, E^{\text{init}}[a, b])$, let $pcgs$ and $ecgs$ be defined as:

- $pcgs(G) = (G, p, e')$, where $e' = (w', E, [a, b])$, $E = a$, and $w' = d$ for every $q \in Q$ and every $\sigma \in D(q)$;
- $ecgs(G) = (G, p, e')$, with $e' = q$ for every $q \in Q$.

The following results easily hold.

**Proposition 2.8.** For every pe-CGS $G$ and pe-ATL formula $\psi$:

- $G \models_p \psi$ if and only if $pcgs(G) \vdash \psi$,
- $G \models_e \psi$ if and only if $ecgs(G) \models \psi$.

**Lemma 2.9.** There are polynomial time reductions from the model checking problem for p-ATL to the one for pe-ATL and from the model checking problem for e-ATL to the one for pe-ATL.

Without loss of generality, we only consider integer energy conditions, i.e., energy conditions $e = (w, E^{\text{init}}[a, b])$ such that $w$ ranges over integer, $E^{\text{init}} \in \mathbb{Z}$, and $a, b \in \mathbb{Z} \cup \{-\infty, +\infty\}$. Details on how to convert a general model checking instance into an equivalent one featuring an integer energy condition are omitted for lack of space. We fix a pe-CGS $G = (G, p, e)$, where $e = (w, E^{\text{init}}[a, b])$ is an integer energy condition. Moreover, we ignore the case where $a = -\infty$ and $b = +\infty$, as it can be dealt with analogously to the one where $a = -\infty$ and $b = +\infty$, and we only consider the remaining three cases: (i) $[a, b] = [-\infty, +\infty]$ (unbounded instances), (ii) $a, b \in \mathbb{Z}$ (bounded instances), (iii) $a \in \mathbb{Z}$ and $b = +\infty$ (mixed instances).

### 3 SOLVING THE EASY CASES

The hardest case to solve when addressing the model checking problem for pe-ATL concerns formulas of the kind $\langle A \rangle \varphi|_{G}$, where $\langle A \rangle$ denotes the set of all states $\{U, \square\}$, denoting the operator, and suitable sets $i$ and $g$ denoting the semantics of $\varphi_1$ and $\varphi_2$, respectively.

In this section, we address this latter problem. More precisely, we devise a procedure to decide if there is an $(i, g, o)$-friendly $(p,e)$-strategy for a team $A$ from a state $q$, for any given $i, q, o, A$, and $g$, and where $e' = (w, E, [a, b])$ for a given $E$, that is, $e'$ is obtained from $e = (w, E^{\text{init}}[a, b])$ by replacing its initial energy level $E^{\text{init}}$ with the given one $E$.

We focus here on bounded and unbounded instances, that is, those in which either $[a, b] = [-\infty, +\infty]$ or $a = -\infty$ and $b = +\infty$. These cases are easier to deal with. In the next section, we will handle the more complex mixed instances.
Unbounded instances can be treated as particular cases of bounded ones. Indeed, an unbounded instance having energy condition \( e = (w, E^{\text{init}}, [0, \infty)) \) can be easily converted into an equivalent bounded one by replacing \( e \) with \( e' = (w', 0, [0, 0]) \), where \( w' \) assigns weight 0 to every transition. In what follows, we focus on bounded instances, unless differently specified.

The algorithm hinges on some characterizations of strategies (Lemma 3.2 and Corollary 3.3 below) in terms of memoryless and uniform ones, the latter being defined as follows.

**Definition 3.1 (uniform strategies).** A uniform strategy \( F_A \) such that \( F_A(c, q) = F_A(c', q) \) for every \( c, c', \) and \( q \) such that \( e\text{-contrib}(e, c) = e\text{-contrib}(e, c') \).

**Lemma 3.2.** If \( a \neq -\infty \) and \( b \neq +\infty \) (bounded instance), then
(a) a \((p, e)\)-strategy from \( q \) exists if and only if a uniform one exists,
(b) for every \( i, q \in Q \), an \((i, q, U)\)-friendly \((p, e)\)-strategy from \( q \) exists if and only if a uniform one exists, and
(c) for every \( i \in Q \), an \((i, U, \Box)\)-friendly \((p, e)\)-strategy from \( q \) exists if and only if a uniform one exists.

Notice also that, despite the results in Lemma 3.2 and unlike the case of \((i, q, U)\)-friendly \((p, e)\)-strategies, an \((i, q, U)\)-friendly \((p, e)\)-strategy is not necessarily uniform, as the strategy can associate different action profiles to the same position depending on which the current phase is: intuitively, a strategy is followed to satisfy the \((i, q, U)\)-friendliness and another one to meet the parity condition.

Clearly, a memoryless strategy (Definition 2.2) is also uniform. Even if the converse is not necessarily true, both kinds of strategies can be considered to be positionally: two histories \((c, q), (c', q')\) with \( e\text{-contrib}(e, c) = e\text{-contrib}(e, c') \) lead to the same position \((q, E)\) of \( G \), and thus a uniform strategy \( F_A \) forces \( A \) to behave uniformly whenever the game is in the same position.

As we have already observed, unbounded instances can be treated as special bounded ones where energy never changes. Thus, there is a bijection between states and reachable positions, from which the following corollary follows.

**Corollary 3.3.** If \([a, b] = [\infty, \infty)\) (unbounded instance), then
- a \((p, e)\)-strategy from \( q \) exists if and only if a memoryless one exists,
- for every \( i, q \in Q \), an \((i, q, U)\)-friendly \((p, e)\)-strategy from \( q \) exists if and only if a memoryless one exists, and
- for every \( i \in Q \), an \((i, U, \Box)\)-friendly \((p, e)\)-strategy from \( q \) exists if and only if a memoryless one exists.

Having these results in mind, it is not difficult to devise algorithms to look for \((i, q, U)\)-friendly \((p, e)\)-strategies (see Algorithm 1 for full details). In the following, we give an intuitive description only for the case when \( o = U \), i.e., \((i, q, U)\)-friendly \((p, e)\)-strategies.

The algorithm consists of two phases. During the first phase, it focuses on the \((i, q, U)\)-friendliness: it explores all strategies for \( A \) trying to reach a goal in \( q \) while guaranteeing invariants in \( i \). Once a goal is reached, the second phase begins, during which the algorithm focuses on the parity condition by searching for the existence of a \((p, e)\)-strategy. The step from the first phase to the second one is triggered when a goal is reached, at which point the flag variable \( igU_{\text{friendly}} \) is set to \text{TRUE} and the history is reset (lines 6-8 in Algorithm 1). Notice that when searching for \((i, q, U)\)-friendly \((p, e)\)-strategies, no goal is ever reached (\( q \) is in fact set to empty at line 2 in Algorithm 1), so the history is never reset and the search is performed in only one phase; this is coherent with the result in Lemma 3.2 (c), which states that a uniform strategy is enough to search for \((i, q, U)\)-friendly \((p, e)\)-strategies.

A strategy is discharged if during one of its outcomes the same position is reached twice without reaching a goal (in the first phase). Observe that the energy condition is checked as well, that is, the algorithm verifies that in each position \((q, E)\) the energy level \( E \) is within the allowed range \([a, b] \).

A strategy is discharged if during one of its outcomes the same position is reached twice without reaching a goal (in the first phase) or being able to guarantee the parity condition by, intuitively, reaching a cycle with even parity (in the second phase). Observe that it is safe to adopt such a termination condition thanks to Lemma 3.2.

**Complexity.** The computational complexity of the procedure is given by the number of recursive calls. In the worst case, the algorithm visits each position once in each phase (that is, before and after reaching a goal), and a recursive call is made for each such visit, thus yielding \( 2 \times [Q] \times (\infty - a + 1) \) calls. Assuming that \( a \) and \( b \) are represented in binary, the complexity of the algorithm is exponential in the size of the input.

**Theorem 3.4.** The procedure 3-strategy-bounded runs in non-deterministic exponential time in the size of the input.

The following corollary comes in handy to deal with unbounded instances, as they can be treated as bounded ones where \( a = b = 0 \).

**Corollary 3.5.** If \( a=b \), then the procedure 3-strategy-bounded runs in non-deterministic polynomial time in the size of the input.

**4 SOLVING THE DIFFICULT MIXED CASE**

In this section we deal with mixed instances, where the energy bound is bounded below and unbounded above. This case is much technical involved (the case in which the energy bound is bounded only above can be dealt with analogously and thus omitted).

We first observe in Section 4.1 that there is a natural correspondence between strategies (resp., \((p, e)\)-strategies) for a team \( A \) and a suitable class of infinite trees, named \( A \)-trees (resp., \((p, e)\)-\( A \)-trees), the latter being easier to manipulate and deal with.
4.1 Tree-based representation for strategies

Trees are particularly apt to express the possible evolutions of a multi-agent system (represented as a pe-CGS) that are consistent with a strategy adopted by a team of agents.

A node is a tuple \( N = (q, E, \tilde{a}, N_1 \ldots N_k) \), where \( q \in Q, E \in Z \) is the energy level associated with the node, \( \tilde{a} \in D(q) \cup \{\#\} \) (\( \# \) is a placeholder for an undefined action profile; it is used for root nodes), and \( N_1 \ldots N_k \) is a finite (possibly empty) sequence of nodes, representing the path to \( N \). We use \( N’ \) to refer to the set of all nodes. We denote by \( \text{state}(N) \), \( \text{e-level}(N) \), \( \text{in-action}(N) \), and \( \text{ancestors}(N) \) the first, second, third, and fourth component of \( N \), respectively, and we write \( \text{path-to}(N) \) to denote the sequence obtained by enqueuing \( N \) to the sequence of \( \text{ancestors}(N) \), i.e., \( \text{path-to}(N) = N_1 \ldots N_k N \). If \( N_1 \) and \( N_2 \) are two nodes such that \( N_1 \in \text{ancestors}(N_2) \), then we say that \( N_1 \) (resp., \( N_2 \)) is an ancestor (resp., a descendant) of \( N_2 \) (resp., \( N_1 \)); we denote by \( \text{desc}(N) \) the set of descendant of a node \( N \), i.e., \( \text{desc}(N) = \{N’ \in N’ \mid N \in \text{ancestors}(N’)\} \). For a node \( N \) we denote by \( \tilde{a} \in D(q) \cup \{\#\} \) (\( \# \) is a placeholder for an undefined action profile; it is used for root nodes), and we write \( \text{path-to}(N) \) to denote the sequence obtained by enqueuing \( N \) to the sequence of \( \text{ancestors}(N) \), i.e., \( \text{path-to}(N) = N_1 \ldots N_k N \). If \( N_1 \) and \( N_2 \) are two nodes such that \( N_1 \in \text{ancestors}(N_2) \), then we say that \( N_1 \) (resp., \( N_2 \)) is an ancestor (resp., a descendant) of \( N_2 \) (resp., \( N_1 \)); we denote by \( \text{desc}(N) \) the set of descendant of a node \( N \), i.e., \( \text{desc}(N) = \{N’ \in N’ \mid N \in \text{ancestors}(N’)\} \). In this context we also say that \( N_1 \) is the father of \( N_2 \), and we use \( \text{father}(N_2) \) to denote \( N_1 \). A root node, or simply root, is a node \( N \) for which \( \text{ancestors}(N) = \{\varepsilon\} \) (\( \varepsilon \) denotes the empty sequence). A tree \( T \) is a set of nodes that contains exactly one root, denoted by \( \text{root}(T) \), and such that for every \( N \in T \) and every ancestor \( N’ \) of \( N \) it holds: (i) \( N’ \in T \), and (ii) \( \text{ancestors}(N) \) is a proper prefix of \( \text{ancestors}(N) \). For a tree \( T \) and a node \( N \in T \), we denote by \( \text{children}(T) \) the set of children of \( N \) in \( T \), i.e., \( \text{children}(T) = \{N’ \in T \mid N’ \text{ is a child of } N\} \). A leaf of \( T \) is a node that has no children in \( T \), i.e., \( \text{children}(T) = \{\emptyset\} \); we denote by \( \text{leaves}(T) \) the set of leaves of \( T \). A branch \( B \) of \( T \) is a maximal subset of \( T \) such that every two different nodes in \( B \) are one an ancestor of the other. By the maximality requirement, if \( B \) is a branch of \( T \) and \( N \in B \) is not a leaf of \( T \), then \( |\text{children}(N)| = 1 \); a branch of \( T \) is finite if and only if it contains a leaf of \( T \). (Notice that a branch of a tree is a tree itself; moreover, since a branch of a tree is a tree itself, we may as well consider a tree as a sequence whenever we find it convenient.)

For a (finite or infinite) sequence of nodes \( N = N_1 \ldots N_k \ldots \), we denote by \( \text{state}(N) \) the sequence of states \( \text{state}(N_1) \text{state}(N_2) \ldots \), and by \( \text{state-action}(N) \) the sequence of pairs \( \text{state}(N_1), \text{in-action}(N_2) \) \( (\text{state}(N_2), \text{in-action}(N_3)) \ldots \). A node \( N \) is (i, g)-friendly (for \( i, g \subseteq Q \)) if there is \( N’ \in \text{path-to}(N) \) such that \( \text{state}(N’)=g \) and \( \text{state}(N’)=i \) for every \( N’ \in \text{ancestors}(N) \). A tree \( T \) is (i, g)-friendly if every branch \( B \) of \( T \) features at least one (i, g)-friendly node; moreover, we say that \( T \) is i- invariant if \( \text{state}(N) \in i \) for every \( N \in T \) and that \( T \) ranges within \( [a, b] \) if \( \text{e-level}(N) \in [a, b] \) for every \( N \in T \).

For \( k \) pairs of nodes \( N_1, N_1’ \ldots N_k, N_k’ \) \( \in X \), \( T \) is the tree obtained from \( T \) by replacing, for every \( j=1 \ldots k \), the sub-tree rooted in \( N_j \) with the one rooted in \( N_j’ \). Towards a formal definition, we first inductively define, for \( N_1, N_2 \) \( \in T \), the node transformation function \( \tau_{T}^{N_1 \to N_2} \) for every \( N \in T \) that is a descendant of \( N_2 \), with \( N = (q, E, \tilde{a}, \text{path-to}(N_2)) \) and \( \text{e-level}(N_1), \text{e-level}(N_2), N’ \) as the tree obtained from \( T \) by replacing, for every \( i=1 \ldots k \), the sub-tree rooted in \( N_i \) with the one rooted in \( N_i’ \).

Finally, we establish a bound for the size of a witness corresponding to a strategy; thus, the search space to search for witnesses is finite, and a decision procedure follows.
4.2 Witnesses
We introduce here the notion of witness (based on the ones of partial
witness and accumulator, see below), and we show in the next
sections that it is possible to reduce the existence of strategies (trees)
to the existence of witnesses of bounded sizes. In the reminder, let
A denote a team.

Definition 4.5 (pw). A partial witness (pw, for short) for A is an
LTS (labeled transition system) \(S = (V = V' \cup V^\text{fin}, T = T' \cup T^\text{fin} \cup T^<, \ell),\) where the set of vertices \(V \subseteq \mathcal{Q}\) (partitioned into \((V', V^\text{fin})\)), the set of transitions \(T \subseteq (T', T^\text{fin} \cup T^<)\), and the labeling function \(\ell : V \to \mathbb{Z}\), associating an energy level with each vertex, are subject to the following constraints:
- for every \(q \in V\) there is \((q, \tilde{a}, q') \in T\) if and only if \(q \in V';\)
- for every \(q \in V\), \(\ell(q) \geq 0;\)
- for every \(e \in \{<, =, >\}\) and \((q, \tilde{a}, q') \in T^<, \ell(q) + w(q, \tilde{a}) \sim \ell(q');\)
- for every \(q \in V'\) there is a unique \(\tilde{a}_V\) \(\in D_A(q)\) for which \(D(q) \cap \text{ext}(\tilde{a}_V) = \{\tilde{a} | (q, \tilde{a}, q') \in T\) for some \(q');\) we denote by \(A\text{-profile}_{\tilde{a}_V}\) such an unique A-action profile \(\tilde{a}_V\).

Unless otherwise stated, we use the following notations: the first component of a pw \(S\) (resp., \(S_i\), with \(i \in \{1, 2, 3, 4\}\)) is denoted by \(V\) (resp., \(V_i\)), its second component is denoted by \(T\) (resp., \(T_i\)), and its third component is denoted by \(\ell\) (resp., \(\ell_i\)); moreover, even if not explicitly said, \(V\) (resp., \(V_i\)) is assumed to be partitioned into \((V', V^\text{fin})\) (resp., \((V'_i, V^\text{fin}_i)\)), while \(T\) (resp., \(T_i\)) is assumed to be partitioned into \((T', T^\text{fin} \cup T^<)\) (resp., \((T'_i, T^\text{fin}_i \cup T^<_i)\)).

Let \(S\) be a pw. We denote by \(S^<\) the set \(\{q \in V | (q, \tilde{a}, q') \in T^<\) for some \(\tilde{a}\) and \(q')\); we say that \(S\) is \((i, g)\)-friendly if \(V' \subseteq V\) and \(V^\text{fin} \subseteq g\) and that \(S\) is \(i\)-invariant if \(V' \subseteq V\). Moreover, we define an \(S\)-path from \(q_1\) to \(q_r\) as a sequence \(\sigma = (q_1, \tilde{a}_1, q_2, \ldots, q_{r-1}, \tilde{a}_{r-1}, q_r)\) with \(r > 1\) and \((q_1, \tilde{a}_1, q_2) \in T\) for every \(i \in \{1, \ldots, r - 1\}\); if \((q_i, \tilde{a}_i, q_{i+1}) \in T'\cup T^<\) for every \(i \in \{1, \ldots, r - 1\}\) and \((q_i, \tilde{a}_i, q_{i+1}) \in T^<\) for at least one \(i \in \{1, \ldots, r - 1\},\) then \(\sigma\) is said to be increasing.

We use the notation \(q_1 \Rightarrow_S q_r\) to denote the existence of an \(S\)-path from \(q_1\) to \(q_r\) and we denote by \(\sigma_{\tilde{a}_V}\) the restriction of \(\sigma\) to elements of \(V, i.e., \sigma_{\tilde{a}_V} = q_1 \tilde{a}_V \ldots q_r.\) Finally, an \(S\)-cycle is an \(S\)-path \(q_1 \Rightarrow_S q_r\) with \(q_1 = q_r.\)

Definition 4.6 (accumulator). An A-accumulator is a pair \(\mathcal{A} = (S_1, S_2)\) of pw's for \(A\) such that:
- \(T^<_2 = \emptyset;\)
- every \(S_2\)-cycle is increasing,
- every \(q \in V^*_2\) occurs in some \(S_2\)-cycle,
- \(V^*'_2 \subseteq V'_1;\)
- \(V^\text{fin}_2 \subseteq V_1\) and \(\ell_2(q) \geq \ell_1(q)\) for every \(q \in V^\text{fin}_2;\)
- \(V^*_{1} \subseteq V^*'_2\) and \(\ell_1(q) \geq \ell_2(q)\) for every \(q \in V^*_{1};\)

Let \(\mathcal{A} = (S_1, S_2)\) be an \(A\)-accumulator. It is said to be acyclic if it features no \(S_1\)-cycles and \(\ell_2(q) > \ell_1(q)\) holds for every \(q \in V^\text{fin}_2\) for which there is a \(q' \in V'_2\) such that \(q' \Rightarrow_{S_2} q\) and \(q \Rightarrow_{S_1} q'\). Its parity function wrt. \(p,\) denoted by \(p^\mathcal{A}: V \to \mathbb{N},\) is defined as:
- \(p^\mathcal{A}(q) = \min\{p(q') | q \Rightarrow_{S_2} q'\}\) if \(q \in V^\text{fin}_2;\)
- \(p^\mathcal{A}(q) = \min\{\ell_2(q') | q \Rightarrow_{S_1} q'\}\) if \(q \in V^\text{fin}_1\).

\(A\) has even parity if \(p^\mathcal{A}(\sigma) = \min\{p^\mathcal{A}(q)| q \in \sigma_{\tilde{a}_V}\}\) is even for every \(S_1\)-cycle \(\sigma.\)

Definition 4.7 (witness). A \(U\)-witness for \((A, i, g)\) is a quadruple \(W = (S_1, S_2, S_3, S_4),\) where
- \((S_1, S_2, S_3)\) is an acyclic \(A\)-accumulator, with \(S_1 (i, g)\)-friendly \(p,\)
- \(S_3, S_4)\) is an \(A\)-accumulator with even parity and \(V^\text{fin}_3 = \emptyset;\)
- \(V^\text{fin}_1 \subseteq V_3\) and \(\ell_1(q) \geq \ell_2(q)\) for every \(q \in V^\text{fin}_1.

A \(\mathcal{U}\)-witness for \((A, i, g)\) is an i-invariant A-accumulator \(W = (S_1, S_2)\) with even parity and \(V^\text{fin}_1 = \emptyset.

Notice that \(g\) plays no role in the definition of \(\mathcal{U}\)-witness and could be omitted; however, we decided to keep it for the sake of a unified notation. For a witness \(W = (S_1, S_2, S_3, S_4)\), we use \(\text{init}(W)\) to refer to the set \(V_1\) of vertices of \(S_1.\)
4.4 From (p,e)-trees to witnesses

In this section, we first define a transformation from an (i,g)-
friendly (p,e)-tree T rooted in q to a U-witness WP for (A,i,g)
with q ∈ init(WP). Then, we show how to adapt such a transformation
to suitably convert an i-invariant (p,e)-tree into a □-witness.

At a very high level, we proceed as follows. First, we show how to obtain, from an (infinite) (i,g)-friendly (p,e)-tree, a finite A-tree which maintains enough significant information about the strategy represented by the original tree. From such a finite tree, we suitably choose nodes that are used as representatives for the vertices of the four partial witnesses that form the desired witness. Roughly speaking, each of these nodes define the behavior of a vertex in a pw by carrying information about its label (energy level) and outgoing transitions. A detailed outline of the transformation follows.

1. Build T′′. We obtain the finite A-tree T′′ from the infinite (p,e)-
tree T, by suitably cutting its branches. To this end, we identify the set of nodes of T whose descendant will be discharged to obtain T′′, or, in other words, the set of nodes that will be leaves in T′′, namely leavesT′. Since T is a (p,e)-tree, every branch B of T satisfies the parity condition, that is, there are along N infinitely many occurrences of a state, let us call it qB, such that p(qB) is even and p(qB) ≤ min{p(q′) | q′ occurs infinitely often along B}; moreover, since T is (i,g)-friendly, B features at least one occurrence of a state from g. Thus, for every branch B of T, there exists N ∈ B for which there are N1, N2 ∈ B, with N2 ∈ ancestors(N) and N1 ∈ path-to(N2), such that state(N1) ∈ g, state(N2) = state(N) = qB, e-level(N2) ≤ e-level(N); let N2 be the earliest node in B meeting these conditions (i.e., no N′ ∈ ancestors(N2) exists with the same properties). We define leavesT′ = {N | B is a branch of T} and T′′ = T \ desc(leaveT′). By König’s Lemma, T′′ is finite.

2. Build T′. Analogously, we obtain the auxiliary finite tree T′′′ from T′′ by suitably cutting its branches: we first define the set leavesT′ and then discharging descendents of nodes in such a set. Let leavesT′ = {N ∈ T′ | state(N) ∈ g and ∀N′ ∈ ancestors(N).state(N′) ∉ g}; we define T′ = T′′ \ desc(leaveT′).

Note that both T′ and T′′′ are A-trees. Moreover, we have that (i) T′ ⊆ T′′ ⊆ T, (ii) state(N) ∈ i for every N ∈ T′′ \ leavesT′, and (iii) state(N) ∈ g for every N ∈ leavesT′.

3. Define a linear order < over nodes in T′ \ leavesT′. We fix a linear order < over T′ \ leavesT′ such that

∀N′, N ∈ T′ \ leavesT′. (N′ ∈ ancestors(N) ⇒ N < N′).

More precisely, < can be thought of as any reverse topological ordering of nodes in T′ \ leavesT′ (since T′ \ leavesT′ is a DAG, a topological order over it exists).

4. Define two representative functions schemas. A representative
function is a function r : V → T′′, where V ⊆ Q.

Let T ⊆ T′ be a set of nodes in T′ and let < be the linear order defined above. Moreover, let

Vτ = {q ∈ Q | q = state(N) for some N ∈ T}. We define two representative functions schemas (parametric in T).

- earliestτ : Vτ → T′′ s.t. for every q ∈ Vτ earliestτ(q) is the earliest node N ∈ T (according to <) with state q; formally:

(a) state(N) = q and

(b) ∀N′ ∈ T′′. N′ < N ⇒ state(N′) ≠ stateN

- lowest-energyτ : Vτ → T′′ such that for every q ∈ Vτ,

lowest-energyτ(q) is the earliest node N ∈ T′′ (according to <) with state q and lowest energy level; formally:

(a) state(N) = q,

(b) ∀N′ ∈ T′′ with state(N′) = state(N)

- N′ < N ⇒ e-level(N′) > e-level(N) and

- N < N′ ⇒ e-level(N′) ≥ e-level(N).

5. Define function pw(·, ·). We show how a set of vertices V ⊆ Q and a representative function r : V → T′′ uniquely identify a pw. Intuitively, the resulting pw has V as set of vertices; the behavior of each vertex q ∈ V, i.e., its label (function r) and the way it evolves (relation r), is determined by r(q), its representative node in T′′.

Formally, let V = V ′ ∪ Vfin ⊆ Q and r : V → T′′ be a representative function. The pw for A induced by (V ′, Vfin, r) is (V, T, τ), where:

- T is the smallest set such that if q ∈ V ′ and N′ ∈ childrenT (r(q)), then (q, in-action(N′), state(N′)) ∈ T, and

- (q) = e-level(r(q)) for every q ∈ V.

6. Build witness WP = (S1, S2, S3, S4). We separate separately how to build (S1, S2) and (S3, S4).

6.1. Definition of S1 and S2. As a preliminary step, we build two auxiliary pw’s for A:

- SLow 1 = (V ′1, T ′1, p1,2) is the pw induced by

(V ′1 \ leavesT′, Vfin, lowest-energyτ)

- SEarly 1 = (V ′1, T ′1, p1,2) is the pw induced by

(V ′1 \ leavesT′, Vfin, earliestτ).

Intuitively, vertices of SLow 1 are also vertices of S1; in addition, vertices of S1 occurring in some S1,2-cycle also belong to S2, along with their successors in S1,2. Vertices of S2 behave in S2 as they do in S1,2; vertices of S1 not belonging to S2 behave in S1 as they do in S1,2; vertices of S1 also belonging to S2 behave in S1 as they do in S1,2, respectively. Formally, components of WP are obtained as follows.

- V ′1 = {state(N) | N ∈ T′ \ leavesT′} ⊆ i,

- Vfin = {state(N) | N ∈ leavesT′} ⊆ g,

- V ′1 = V ′2 = V ′ fin.

- (vertices occurring in S1,2-cycles and their successors in S1,2 belong to S2) V1 is the smallest set such that, for every q ∈ V ′1, if q occurs in an S1,2-cycle then (q, q ′) ∈ V1 for every (q, a, q ′) ∈ T ′1,2.

- (vertices of S2 behave in S2 as they do in S1,2) T ′1 = {(q, a, q ′) ∈ T ′1,2 | q ∈ V ′1 and q occurs in an S1,2-cycle},

- t1(q) = t1,2(q) for every q ∈ V1,

- (vertices of S1 not belonging to S2 behave in S1 as they do in S1,2) T ′1 = {(q, a, q ′) ∈ T ′1,2 | q ∈ V ′1 \ V1,2},

- t1(q) = t1,2(q) for every q ∈ V1 \ V2,

- no other transition belongs to T1.

6.2. Definition of S3 and S4. The definition of S1 and S2 is very similar to the one of S3 and S4, respectively; the only differences being (we let X = ((T′ \ T′′) \ leavesT′) \ leavesT′):
functions lowest-energy$_X$ and earliest$_X$ replace lowest-energy$_Y$ and earliest$_Y$, respectively, as representative functions, and the two auxiliary pw’s $S^{3,4}_\text{low}$ and $S^{3,4}_\text{early}$ for $A$ are defined as:

$S^{3,4}_\text{low} = (V^{3,4}_\text{low}, T^{3,4}_\text{low}, \tau)$ is the pw induced by $(V_X, \emptyset, \text{lowest-energy}_X)$

$S^{3,4}_\text{early} = (V^{3,4}_\text{early}, T^{3,4}_\text{early}, \tau)$ is the pw induced by $(V_X, \emptyset, \text{earliest}_X)$.

$V^3_3 = \emptyset$ (whereas, possibly, $V^3_1 \neq \emptyset$) and thus $V_3 = V^3_3 = V_X$.

It is possible to show the converse correspondence between witnesses and (p,e)-trees, as stated in the following lemma.

**Lemma 4.13.** If $T$ is an $(i, g)$-friendly $(p,e)$-tree rooted in $q$, then $W_T$ is a $U$-witness for $(A,i,g)$ with $q \in \text{init}(W)$.

The above procedure can be easily adapted as follows, to suitably convert $i$-invariant $(p,e)$-trees into witnesses for $(A,i,g)$. By assuming $g = Q$, the procedure yields $W_T = (S_1, S_2, S_3, S_4)$, where $A$-accumulator $(S_1, S_2, S_3)$ can be ignored (it is a trivial $A$-accumulator such that $S_1 = S_2, V_1 = \{\text{state}(\text{root}-T)\}$ and $T_1 = \emptyset$) as there is no need to search for goals; on the other hand, the $A$-accumulator $(S_1, S_3)$ is the desired $\text{witness}$.

**Lemma 4.14.** If $T$ is an $i$-invariant $(p,e)$-tree rooted in $q$, then $W_T$ is a $U$-witness for $(A,i,g)$ with $q \in \text{init}(W)$.

Finally, the next theorem immediately follows from Theorem 4.4, Corollary 4.11, Lemma 4.12, Lemma 4.13, and Lemma 4.14.

**Theorem 4.15.** An $(i, g, o)$-friendly $(p,e)$-strategy from $q$ exists if and only if there is an $o$-witness $W$ for $(A,i,g)$ with $q \in \text{init}(W)$.

Now, a simple algorithm (see Algorithm 2) to search for an $(i,g,U)$-friendly strategy non-deterministically guesses four pw’s $S_1, S_2, S_3$, and $S_4$, and then checks that conditions (a)-(c) of Definition 4.7 are satisfied. Similarly, when looking for $(i,g,\mathcal{U})$-friendly strategies the algorithm non-deterministically guesses $S_1$ and $S_2$, and then checks that they form a $\mathcal{U}$-witness (see Definition 4.7). Since such checks can be done in polynomial time, we have the following result.

**Theorem 4.16.** The procedure $\exists$-strategy-mixed runs in non-deterministic polynomial time in the size of the input.

### 5 HARVEST AND CONCLUSIONS

We are finally ready to employ the results of the previous sections into a procedure for model checking in pe-ATL, presented in Algorithm 3. The case of formulas of the kind $\langle A \rangle \psi_1$ is worked out as one might expect; the only worthwhile remark is about line 22, where, towards the valuation of a strategy, the algorithm checks that for all successors $q''$ of the state $q$ under investigation there is a $(p,e)$-strategy from $q''$ (besides checking that $q''$ satisfy formula $\psi_1$) by using the procedure $\exists$-strategy-$x$ (suitably instantiated depending on the type of instance in input) developed in the previous section to search for a $(Q,0,\mathcal{U})$-friendly $(p,e)$-strategy or, more intuitively, the algorithm checks that $q''$ satisfies formula $\langle A \rangle \psi_1$.

Formulas of kinds $\langle A \rangle \psi_1 \langle U \rangle \psi_2$ and $\langle A \rangle \langle U \rangle \psi_2$ are handled using the procedures described in the previous sections.

It is clear that complexity of the whole procedure is governed by the one of procedure $\exists$-strategy-$x$ ($x \in \{\text{bounded, mixed}\}$). Thus, we have the following theorem, which comes from Theorem 3.4, Corollary 3.5, and Theorem 4.16.

**Theorem 5.1.** The model checking problem for pe-ATL is:

- in $\text{NEXTIME}$ if $a, b \in \mathbb{B}$ (bounded instances),
- in $\text{NPTIME}$ if $a, b = [-\infty, +\infty]$ (unbounded instances),
- in $\text{NPTIME}$ if $a \in \mathbb{N}$ and $b = +\infty$ (mixed instances).

The proposed setting follows a recent and promising trend devoted to the study of systems enabling qualitative and quantitative reasoning in MAS. Before the last decade, these two aspects have been mostly kept separate, despite their interplay in many natural application scenarios (e.g., allocation systems subject to energy constraints—see Introduction). Our proposal aims at developing a logical system able to deal with these two aspects jointly.

As future work, we aim at establishing thig complexity bounds for the problems considered here, as well as considering different choices for modeling energy condition: at least another option is worth being considered, according to which energy level evolves while trying to satisfy the formula along the entire game (in our setting the energy level is reset whenever a new search for strategy by a possibly different team begins—see [6] for a comparison on the two approaches in the setting of ATL without parity condition). Moreover, we plan to extend the proposed framework to ATL^*. 

**Acknowledgements.** We thank David de Frutos for helpful comments on uniform, bounded, and memoryless strategies. Work partially supported by a 2018 CNCS project.
REFERENCES


