Model checking coalitional games in shortage resource scenarios*

Della Monica, Dario
ICE-TCS, School of Computer Science
Reykjavik University, Iceland
dariodm@ru.is

Napoli, Margherita
Dipartimento di Informatica
University of Salerno, Italy
napoli@dia.unisa.it

Parente, Mimmo
Dipartimento di Informatica
University of Salerno, Italy
parente@unisa.it

Verification of multi-agents systems (MAS) has been recently studied taking into account the need of expressing resource bounds. Several logics for specifying properties of MAS have been presented in quite a variety of scenarios with bounded resources. In this paper, we study a different formalism, called *Priced Resource-Bounded Alternating-time Temporal Logic* (PRB-ATL), whose main novelty consists in moving the notion of resources from a syntactic level (part of the formula) to a semantic one (part of the model). This allows us to track the evolution of the resource availability along the computations and provides us with a formalisms capable to model a number of real-world scenarios. Two relevant aspects are the notion of global availability of the resources on the market, that are shared by the agents, and the notion of price of resources, depending on their availability. In a previous work of ours, an initial step towards this new formalism was introduced, along with an EXPTIME algorithm for the model checking problem. In this paper we better analyze the features of the proposed formalism, also in comparison with previous approaches. The main technical contribution is the proof of the EXPTIME-hardness of the the model checking problem for PRB-ATL, based on a reduction from the acceptance problem for *Linearly-Bounded Alternating Turing Machines*. In particular, since the problem has multiple parameters, we show two *fixed-parameter* reductions.

1 Introduction

Verification of multi-agents systems (MAS) is a topic under investigation by several research groups in computer science in the last ten years ([8]). Most of the research is based on logical formalisms, maybe the most famous being the *Alternating-time Temporal Logics* (ATL) [3] and the *Coalition Logic* (CL) [15, 16], both oriented towards the description of collective behaviors and used as specification languages for open systems. These scenarios are hence naturally modeled as games. In [10] it has been shown that CL can be embedded into ATL. Recently, these two logics have been used for the verification of multi-agent systems (MAS), enhanced with resource constraints [1, 2, 5, 6, 9]. The intuitive idea is that agent actions consume and/or produce resources, thus the choice of a given action of an agent is subject to the availability of the resources. In [1], Alechina et al. introduce the logic *Resource-Bounded Coalition Logic* (RBCL), whose language extends the one of CL with explicit representation of resource bounds. In [2], the same authors propose an analogous extension for ATL, called *Resource-Bounded Alternating-time Temporal Logics* (RB-ATL), and give a model checking procedure that runs in time $O(|\varphi|^{2 \cdot r+1} \times |G|)$, where $|\varphi|$ is the length of the formula φ to be checked, $|G|$ is the size of the model

---

To appear in EPTCS.

G, and *r* is the number of resources. However, the problem of determining a lower bound to the model checking problem is left open. In [6], Bulling and Farwer introduce two *Resource-Bounded Agent Logics*, called RAL and RAL*. The former represents a generalization of Alechina et al.'s RB-ATL, the latter is an analogous extension of ATL* (analogous extensions for, respectively, CTL and CTL* were presented by the same authors in [5]). The authors study several syntactic and semantic variants of RAL and RAL* with respect to the (un)decidability of the model checking problem. In particular, while previous approaches only conceive actions *consuming* resources, they introduce the notion of actions *producing* resources. It turned out that such a new notion makes the model checking problem undecidable. Formulae of the formalisms proposed in [1, 2, 5, 6] allow one to assign an endowment of resources to the agents by means of the so-called *team operators* (borrowed from ATL). The problem is then to determine whether the agents in the *proponent* team have a strategy for the game to carry out the assigned goals with that bounded amount of resources, whatever the agents in the *opponent* team do.

In this paper we study a different formalism, called *Priced Resource-Bounded Alternating-time Temporal Logic* (PRB-ATL), introduced in [9], but in a much less mature version. The key features of this new approach toward the formalization of such complex systems can be summarized as follows.

- *Boundedness of the resources.* This is a crucial point in our formalization. In order to model boundedness of the resources, a notion of *global availability* of resources on the market (or in nature), which evolves depending on both proponent and opponent behaviors, is introduced. Such a global availability is a semantic component (it is part of the structure where the logic is interpreted) and its evolution is tracked during the executions of the system. Agents' moves are affected by the current global availability (e.g., agents cannot consume an unbounded amount of resources).

- *Resources are shared.* Resources are global, that is, they are shared by all the agents. Thus, the agents either consume or produce resources out of a shared pool of bounded capability, and acquisition (resp., release) of a resource by an agent (independently if the agent belongs to the proponent or opponent team) implies that the resource will be available in smaller (resp., greater) quantity. In this way, we can model several scenarios where shared resources are acquired at a cost that depends on that resource current availability (for example in concurrent systems where there is a competition on resources).

- *Money as a meta-resource.* In addition to public shared resources, our setting also allows one to model *private* resources, that is, resources that are possessed by agents (public resources are present in the market and will be acquired by the agents in case they need). The idea is to provide the agents with the unique private resource, *money*, that can be used to acquire (public) resources needed to perform the tasks. In this sense, money represent several resource combinations and can be considered as a meta-resource. Unlike the other resources, it is a syntactic component (money endowment is part of the formula), and is the only (meta-)resource which is private for an agent.
  At this stage, our formalization only features the possibility of assigning to agents one private resource. Nevertheless, in principle, it is possible to extend the idea to admit a vector of private resources. Furthermore, one could think of including the same resource in both the pool of public resources and in the pool of private ones. For instance, in a car race one of the players (the cars) possesses some gasoline in the tank (private resource) but he needs to acquire more gasoline at the gas station (public resource) to complete the race.

- *Resource production.* Production of resources is allowed in a quantity that is not greater than a fixed amount. Thus, we extend the model still preserving the decidability of the model checking problem. Observe that the constraint we impose still allows us to describe many interesting real-world scenarios, such as acquiring memory by a program, or leasing a car during a travel, or, in

general, any release of resources previously acquired. A similar setting has been already observed also in [6].

- *Opponent power.* First observe that we use the standard terminology which separates the role of the agents in a proponent team and those in the opponent team. This distinction is not within the game structure, but it is due to the formula under consideration. Agents of the opponent team are subject to resource availability in choosing the action to perform, in the same way as the agent of the proponent team, thus the opponent team cannot interfere with a proponent strategy performing actions which either consume or produce too much (see Example 3 in Section 3). However, it is common practice to consider opponent having maximum power, to look for robust strategy. We give unlimited economic power to the agents in the opponent team, in the sense that at each moment they have money enough to acquire the resources they need for a move, provided that the resources are available.

Actually in [9] an EXPTIME algorithm for the model checking problem was given, along with a PSPACE lower bound. The main technical contribution here is to provide an EXPTIME lower bound for the model checking problem for PRB-ATL. This result shows that the model checking problem for this logic is EXPTIME-complete. The hardness proof is obtained by means of a reduction from the acceptance problem for *Linearly-Bounded Alternating Turing Machines* (LB-ATM), known to be EXPTIME-complete [7], to the model checking problem for PRB-ATL. More precisely, let n be the number of agents, r the number of resources, and M the maximum component occurring in the initial resource availability vector, the algorithm given in [9] runs in exponential time in n, r, and the size of the representation of M (assuming that M is represented in binary). To prove here the inherent difficulty with respect to multiple input parameters, we show two reductions: one parametric in the representation of M (the digit size), that assumes constant both n and r, and another parametric in r, and assuming constant both n and the value of M.

2 Comparison with related works

In this section we compare our approach with the existing literature underlining differences and similarities respect to [2] and [6].

In the work by Alechina et al. [2], resource bounds only appear in the formulae and are applied solely to the proponent team, but they are not represented inside the model. Indeed, agents of the proponent team are endowed with new resources at the different steps of the system execution. This means that it is possible to ask whether a team can reach a goal with a given amount of resources, but it is not possible to keep trace of the evolution of the global availability of resources. Moreover, resources are private to agents of the proponent team (not shared, as in our approach) and resource consumption due to the actions of the opponent is not controlled. Here instead, we keep trace of resource global availability, whose evolution depends on both proponent and opponent moves. In this way, it is possible to avoid undesired/unrealistic computations of the system such as, for instance, computations consuming unboundedly. Let us see a very simple example. Consider the formula $\psi = \langle\langle A^{\vec{\$}}\rangle\rangle\Box p$. Its semantics is that agents in team A have together a strategy which can guarantee that p always holds, whatever agents of the opponent team do (without consuming too many resources) and provided the expense of the agents in A does not exceed $\vec{\$}$. A loop in the structure where the joint actions of agents consume resources without producing them, cannot be a model for ψ. On the contrary, consider the formula $\psi' = \langle\langle A^b\rangle\rangle\Box p$, belonging to the formalism proposed in [2], expressing a similar property, with the only difference that the agents of A use an amount of resources bounded by b. A model for ψ' must contain a loop where

the actions of agents in *A* do not consume resources, but the actions of agents in the opponent team may possibly consume resources, leading to an unlimited consumption of resources.

As a further difference, recall that in [2] actions can only consume resources. Without resource productions, the model for many formulae (for example those containing the *global* operator □) must have a loop whose actions do not consume resources (*do-nothing* actions), and a run satisfying these formulae is eventually formed by only such actions. On the contrary, by allowing resource production, we can model more complex situations when dealing with infinite games.

Finally, observe that a similarity with the cited paper is in the role of money, that could be seen as a private resource, endowed to the agents of the proponent team.

Bulling and Farwer [6] adopted an "horizontal" approach, in the sense that they explored a large number of variants of a formalism to model these complex systems. In particular, they explored the border between decidability and undecidability of the model checking problem for all such variants, and they showed how the status of a formalisms (wrt decidability of its model checking problem) is affected by (even small) changes in language, model, and semantics. Our work takes advantage of this analysis in order to propose a logic that captures several desirable properties (especially concerning the variety of natural real world scenario that is possible to express), still preserving decidability. However, our approach presents conceptual novelties that make it difficult to accomplish a direct comparisons between the formalisms presented here and the ones proposed in [6]. We are referring here to both the above mentioned idea of dealing with resources as global entities for which agents compete, and the notion of cost of resource acquisition (price of the resources) that dynamically changes depending on the global availability of that resource (thus allowing one to model the classic market law that says that getting a resource is more expensive in shortage scenario). In [6], there is no such a notion as resources are assigned to (team of) agents and proponent and opponent do not compete for their acquisition.

As regards the complexity issue, in [6], no complexity analysis (for the model checking problem) is performed, while, in [2], an upper bound is given for RB-ATL, that matches the one given in [9] for PRB-ATL. The algorithm for PRB-ATL runs in exponential time in the number *n* of agents, the number *r* of resources, and the digit size of the maximum component *M* occurring in the initial resource availability vector (assuming a binary reppresentation). Analogously, the model checking algorithm for RB-ATL runs in exponential time in *r*, in the digit size of the maximum component of resource endowment vectors *b* occuring in team operators $\langle\langle A^b \rangle\rangle$ of φ and in the number *n* of the agents (this is implicit in set of states of $|G|$). Actually, both *n* and *r* are often treated as constant [2, 3] (without this assumption, the complexity of ATL model-checking is shown to be exponential in the number of agents [11]). However, no complexity lower bound has been exhibit so far. Aim of this paper is to fill this gap, by providing an EXPTIME lower bound for PRB-ATL.

3   A logical formalization: PRB-ATL

Syntax. We start with the introduction of some notations we will use in the rest of the paper. The set of *agents* is $\mathscr{AG} = \{a_1, a_2, \ldots, a_n\}$ and a *team* is any subset of $\mathscr{AG}$. The integers *n* and *r* will be used throughout the paper to denote the number of agents and *resource types* (or simply *resources*), respectively. Let $\mathscr{M} = (\mathbb{N} \cup \{\infty\})^r$ denote the set of *global availabilities of resources on the market (or in nature)* and let $\mathscr{N} = (\mathbb{N} \cup \{\infty\})^n$ denote the set of *money availabilities for the agents*, where $\mathbb{N}$ is the set of natural numbers (zero included). Given a money availability $\vec{\$} \in \mathscr{N}$, its *i*-th component $\vec{\$}[i]$ is the

money availability of agent a_i[1]. Finally, the set Π is a finite set of *atomic propositions*.

The formulae of PRB-ATL are given by the following grammar:

$$\varphi ::= p \mid \neg\varphi \mid \varphi \wedge \varphi \mid \langle\langle A^{\vec{\$}}\rangle\rangle \bigcirc \varphi \mid \langle\langle A^{\vec{\$}}\rangle\rangle\varphi\,\mathscr{U}\,\varphi \mid \langle\langle A^{\vec{\$}}\rangle\rangle\Box\varphi \mid \sim\vec{m}$$

where $p \in \Pi$, $A \subseteq \mathscr{A}\mathscr{G}$, $\sim\,\in\{<,\leq,=,\geq,>\}$, $\vec{m}\in\mathscr{M}$ and $\vec{\$}\in\mathscr{N}$. Formulae of the kind $\sim\vec{m}$ test the current availability of resources on the market. As usual, other standard operators can be considered as abbreviation, e.g., the operator $\langle\langle A^{\vec{\$}}\rangle\rangle\Diamond\psi$ can be defined as $\langle\langle A^{\vec{\$}}\rangle\rangle\top\,\mathscr{U}\,\psi$, for every formula ψ.

Priced game structure. *Priced game structures* are defined by extending the definitions of concurrent game structure and resource-bounded concurrent game structure given in, respectively, [3] and [2].

Definition 1 *A priced game structure G is a tuple $\langle Q,\pi,d,D,qty,\delta,\rho,\vec{m}_0\rangle$, where:*
- *Q is the finite set of* locations*; $q_0 \in Q$ is called* initial location.
- *$\pi: Q \to 2^{\Pi}$ is the* evaluation function*, which determines the atomic propositions holding true in each location.*
- *$d: Q \times \mathscr{A}\mathscr{G} \to \mathbb{N}$ is the* action function *giving the number $d(q,a) \geq 1$ of* actions *available to an agent $a \in \mathscr{A}\mathscr{G}$ at a location $q \in Q$. The actions available to a at q are identified with the numbers[2] $1,\ldots,d(q,a)$ and a generic action is usually denoted by α. We assume that each agent has at least one available action at each location, that could be thought of as the action* do-nothing *and we assume that it is always the first.*
- *$D: Q \to 2^{\mathbb{N}^n}$ is a function that maps each location q to the set of vectors $\{1,\ldots,d(q,a_1)\}\times\ldots\times\{1,\ldots,d(q,a_n)\}$. Each vector, called* action profile *and denoted by $\vec{\alpha}$, identifies a choice among the actions available for each agent in the location q. (The action of the agent a in $\vec{\alpha}$ is $\vec{\alpha}(a)$.)*
- *$qty: Q \times \mathscr{A}\mathscr{G} \times \mathbb{N} \to \mathbb{Z}^r$ is a partial function, where $qty(q,a,\alpha)$, with $1 \leq \alpha \leq d(q,a)$, defines at location q the amount of resources required by the a's action α. We define $qty(q,a,1) = \vec{0}$, that is the vector whose components are all equal to 0, for every $q \in Q$, $a \in \mathscr{A}\mathscr{G}$ (doing nothing neither consumes nor produces resources).*
- *$\delta: Q \times \mathbb{N}^n \to Q$ is the* transition function. *For $q \in Q$ and $\vec{\alpha} \in D(q)$, $\delta(q,\vec{\alpha})$ defines the next location reached from q if the agents perform the actions in the action profile $\vec{\alpha}$.*
- *$\rho: \mathscr{M} \times Q \times \mathscr{A}\mathscr{G} \to \mathbb{N}^r$ is the* price function. *It returns the* price *vector of the resources (a price for each resource), based on the current resource availability and location, and on the acting agent.*
- *$\vec{m}_0 \in \mathscr{M}$ is the initial global availability of resources. It represents the resource availability on the market at the initial state of the system.*

Note that a negative value in $qty(q,a,\vec{\alpha})$ represents a resource consumption, while a positive one represents a resource production. We also consider the extension of the function qty, called again with the same name, to get the amount of resources required by a given team. Thus, for a location q, a team A and an action profile $\vec{\alpha}$, $qty(q,A,\vec{\alpha}) = \sum_{a\in A} qty(q,a,\vec{\alpha}(a))$. Moreover, we will use the function $consd : Q \times \mathscr{A}\mathscr{G} \times \mathbb{N} \to \mathbb{N}^r$ that for the tuple (q,a,α) returns the vector of the resources which are consumed by an agent a, being in state q, for an action α. This vector is obtained from $qty(q,a,\alpha)$ by replacing the positive components, representing a resource production, with zeros, and the negative components, representing a resource consumption, with their absolute values.

Example 1 *A priced game structure with two agents a_1 and a_2 and one resource R_1 is depicted in Figure 1. The only atomic proposition is p, labeling the locations q_0, q_1, q_2. The action profiles, labeling*

---

[1]Throughout all the paper, symbols identifying vectors are denoted with an arrow on the top (e.g., $\vec{\$}$, $\vec{m}$).

[2]No ambiguity will arise from the fact that actions of different agents are identified with the same numbers.
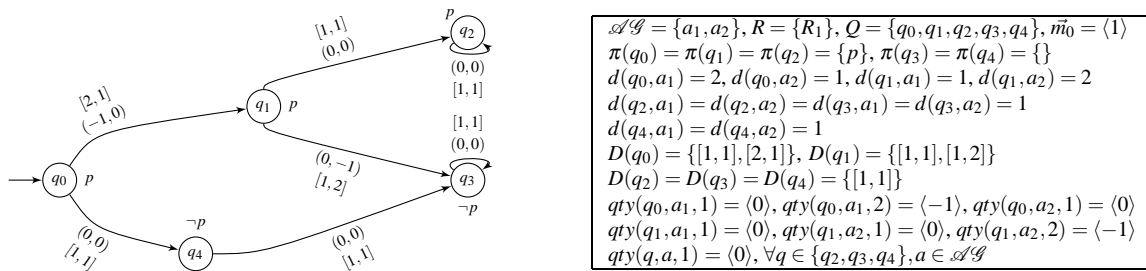
Figure 1: Example of priced game structure $G = \langle Q, \pi, d, D, qty, \delta, \rho, \vec{m}_0 \rangle$.

the transitions in the graph and depicted with square brackets, are as follows. $D(q_0) = \{[1,1],[2,1]\}$ *is due to the existence of two actions of a_1 and one action of a_2 at location q_0, $D(q_1) = \{[1,1],[1,2]\}$ corresponds to a single action of a_1 and two actions of a_2 at location q_1. In all the other locations the only action profile is $[1,1]$ corresponding to the existence of a single action of both the agents. The function qty is represented by parentheses. The price vector is not depicted.*

Semantics. In the following, given a resource availability $\vec{m}$, by $\mathcal{M}^{\leq \vec{m}}$ we denote the set $\{\vec{m}' \in \mathcal{M} \mid \vec{m}' \leq \vec{m}\}$. In order to give the formal semantics let us first define the following notions.

Definition 2 *A* configuration *c of a priced game graph G is a pair $\langle q, \vec{m} \rangle \in Q \times \mathcal{M}^{\leq \vec{m}_0}$. Given two configurations $c = \langle q, \vec{m} \rangle$ and $c' = \langle q', \vec{m}' \rangle$, and an action profile $\vec{\alpha} \in D(q)$, we say that $c \to_{\vec{\alpha}} c'$ if $q' = \delta(q, \vec{\alpha})$ and $\vec{m}' = \vec{m} + qty(q, \mathcal{AG}, \vec{\alpha})$. A* computation *over G is an infinite sequence $C = c_1 c_2 \ldots$ of configurations of G, such that for each i there is an action profile $\vec{\alpha}_i$ such that $c_i \to_{\vec{\alpha}_i} c_{i+1}$.*

Let $C = c_1 c_2 \ldots$ be a computation. We denote by $C[i]$ the i-th configuration c_i in C and by $C[i,j]$, with $1 \leq i \leq j$, the finite sequence of configurations $c_i c_{i+1} \ldots c_j$ in C. Given a configuration $c = \langle q, \vec{m} \rangle$ and a team A, a function $\vec{\alpha}_A : A \to \mathbb{N}$ is called *A-feasible in c* if there exists an action profile $\vec{\alpha} \in D(q)$ with $\vec{\alpha}_A(a) = \vec{\alpha}(a)$ for all $a \in A$ and $\vec{0} \leq qty(q, A, \vec{\alpha}) + \vec{m} \leq \vec{m}_0$. In this case we say that $\vec{\alpha}$ extends $\vec{\alpha}_A$.

Definition 3 *A* strategy *F_A of a team A is a function which associates to each finite sequence of configurations $c_1 c_2 \ldots c_s$, a function $\vec{\alpha}_A : A \to \mathbb{N}$ which is A-feasible in c_s.*

In other words, a strategy F_A returns a choice of the actions of the agents in the team A, considering only those actions whose resource consumption does not exceed the available amount and whose resource production does not exceed the amount consumed so far. Clearly, this constraint will limit both proponent and opponent team.

For each strategy F_A of a team A and for each sequence of configurations $c_1 c_2 \ldots c_s$, there are several possibilities for the next configuration c_{s+1}, depending on the different choices of the opponent team $\overline{A} = \mathcal{AG} \setminus A$. Anyway, fixed a strategy $F_{\overline{A}}$ of the opponent team, there is at most one action profile obtained according to both the strategies, that is the action profile $\vec{\alpha}$ extending both $\vec{\alpha}_A$, given by the strategy F_A, and $\vec{\alpha}_{\overline{A}}$, given by the strategy $F_{\overline{A}}$ (i.e. $\vec{\alpha}$ is such that $\vec{\alpha}(a) = \vec{\alpha}_X(a)$, for $X \in \{A, \overline{A}\}$ and $a \in X$). A computation $C = c_1, c_2 \ldots$, is the *outcome of the strategies F_A and $F_{\overline{A}}$ from the configuration c_1* if, for each $i \geq 1$, there is an action profile $\vec{\alpha}_i$ obtained according to both F_A and $F_{\overline{A}}$, such that $c_i \to_{\vec{\alpha}_i} c_{i+1}$. Given a strategy F_A and a configuration c, $out(c, F_A)$ denotes the set of the outcomes of F_A and $F_{\overline{A}}$ from c, for all the strategies $F_{\overline{A}}$ of the team $\overline{A}$. Observe that, given a finite sequence of configurations $C = c_1 c_2 \ldots c_s$, if the action profile $\vec{\alpha}$ according to the two strategies is not such that $\vec{0} \leq qty(q_s, \mathcal{AG}, \vec{\alpha}) + \vec{m}_s \leq \vec{m}_0$, then there is no next configuration. Thus outcome of the strategies F_A and $F_{\overline{A}}$ from a given configuration may be undefined (recall that we consider only infinite computations).

Example 2 *Consider the priced game structure in Figure 1, with teams $A = \{a_1\}$ and $B = \{a_2\}$, one resource type and initial global availability $\vec{m}_0 = \langle 1 \rangle$. Let $c = \langle q_0, \langle 1 \rangle \rangle$ be a sequence of configurations (of length 1). Team A has two possible strategies in c, one for each possible action of agent a_1, and team B has one strategy for the single available action of agent a_2. Suppose that, according to the strategy F_A, agent a_1 chooses to perform the action 2 ($F_A(c)(a_1) = 2$), then the action profile $[2,1]$ is performed and one unit of the unique resource is consumed. In the obtained configuration $\langle q_1, \langle 0 \rangle \rangle$ the agent a_1 has one available action while the agent a_2 has two actions. Anyway F_B cannot return the action 2 for the agent a_2, since this action would require an amount of the resource greater than 0, which is the current availability. Thus only the configuration $\langle q_2, \langle 0 \rangle \rangle$ can be reached and the computation $C = \langle q_0, \langle 1 \rangle \rangle \langle q_1, \langle 0 \rangle \rangle \langle q_2, \langle 0 \rangle \rangle \langle q_2, \langle 0 \rangle \rangle \ldots$ is the only one that belongs to $out(c, F_A)$.*

Now we introduce the concept of consistent strategy. Two properties have to be satisfied: first, the outcomes starting from c are always defined and also the agents of the proponent team have enough money to realize the chosen actions.

Definition 4 *Let $\vec{\$} \in \mathcal{N}$, c be a configuration, $A \subseteq \mathscr{AG}$ be the proponent team, and $\overline{A} = \mathscr{AG} \setminus A$ be the opponent team. A strategy F_A of A is said to be* consistent with respect to $\vec{\$}$ and c (($\vec{\$}$,c)-strategy)*, if*
  1. *for any strategy $F_{\overline{A}}$ of $\overline{A}$, the outcome of F_A and $F_{\overline{A}}$ from the configuration c is defined,*
  2. *for every $C = c_1 c_2 \ldots \in out(c, F_A)$, with $c_i = \langle q_i, \vec{m}_i \rangle$, for every $i \geq 1$ and $a_k \in A$: $\sum_{j=1}^{i} \rho(\vec{m}_j, q_j, a_k) \cdot consd(q_j, a_k, F_A(C[1, j])(a_k)) \leq \vec{\$}[k]$.*

In the above condition the dot operator denotes the usual scalar product of vectors. Observe that only the money availability of the team A is tested. Actually, we suppose that the opponent team $\overline{A}$ always has money enough to make its choice. Notice also that the actions *producing* resources do not cause a reimbursement of money to the agents. As it is usual when dealing with temporal logics, we guarantee that priced game structures are non-blocking, in the sense that at least a $(\vec{\$},c)$-strategy exists for a given team A. Indeed, agents of A can always jointly choose the *do-nothing* action.

A formula of PRB-ATL is evaluated with respect to a priced game structure G and a configuration $c = \langle q, \vec{m} \rangle$. The definition of the semantics is completed by the definition of the satisfaction relation $\models$:
  - $(G, c) \models p$ iff $p \in \pi(q)$
  - $(G, c) \models \neg\psi$ iff $(G, c) \not\models \psi$
  - $(G, c) \models \psi_1 \wedge \psi_2$ iff $(G, c) \models \psi_1$ and $(G, c) \models \psi_2$
  - $(G, c) \models \langle\langle A^{\vec{\$}} \rangle\rangle \bigcirc \psi$ iff there exists a $(\vec{\$}, c)$-strategy F_A such that, for all $C \in out(c, F_A)$, it holds that $(G, C[2]) \models \psi$
  - $(G, c) \models \langle\langle A^{\vec{\$}} \rangle\rangle \psi_1 \mathscr{U} \psi_2$ iff there exists a $(\vec{\$}, c)$-strategy F_A such that, for all $C \in out(c, F_A)$, there exists $i \geq 0$ such that $(G, C[i]) \models \psi_2$ and, for all $1 \leq j < i$, it holds that $(G, C[j]) \models \psi_1$
  - $(G, c) \models \langle\langle A^{\vec{\$}} \rangle\rangle \Box \psi$ iff there exists a $(\vec{\$}, c)$-strategy F_A such that, for all $C \in out(c, F_A)$, it holds that $(G, C[i]) \models \psi$ for all $i \geq 1$
  - $(G, c) \models \sim \vec{m'}$ iff $\vec{m} \sim \vec{m'}$ where $\sim \in \{<, \leq, =, \geq, >\}$.

Given a PRB-ATL formula and a priced game srtucture G, we say that G satisfies φ, $G \models \varphi$, if $(G, c_0) \models \varphi$ where $c_0 = \langle q_0, \vec{m}_0 \rangle$. The *model checking* problem for PRB-ATL consists in verifying whether $G \models \varphi$.

Example 3 *Consider the priced game structure in Figure 1, with teams $A = \{a_1\}$ and $B = \{a_2\}$. A formula $\psi = \langle\langle \mathscr{AG}^{\vec{\$}} \rangle\rangle \bigcirc \langle\langle A^{\vec{\$'}} \rangle\rangle \Box p$ holds true in the configuration $\langle q_0, \langle 1 \rangle \rangle$, provided that $\vec{\$}$ and $\vec{\$'}$ are enough to make the move. Indeed, a_1 and a_2 together are able to force the computation to reach the*

$\langle q_1, \langle 0 \rangle \rangle$ *(one unit of resource is consumed). From such a configuration, the opponent team B cannot force the computation into q_3, as the action 2 is not allowed for a_2 (no resources are available to perform the action), and thus ψ holds. Instead, ψ is false in the configuration $\langle q_0, \langle 2 \rangle \rangle$ (actually in each configuration $\langle q_0, \langle x \rangle \rangle$, with $x > 1$), because $\langle q_1, \langle 1 \rangle \rangle$ is reached after the execution of the first transition, and in that configuration action 2 for a_2 in B is allowed, leading to q_3. Finally, notice that the formula is false also when evaluated in $\langle q_0, \langle 0 \rangle \rangle$, as the only possible transition is the one leading from q_0 to q_4 (no resources are available to perform action 1 for agent a_1).*

4   Complexity lower bounds for the model checking problem

In [9], the authors presented an algorithm for model checking PRB-ATL, providing an exponential upper bound for the problem. In particular, let *n* be the number of agents, *r* the number of resources, and *M* the maximum component occurring in the initial resource availability vector, the proposed algorithm runs in exponential time in *n*, *r*, and the size of the representation of *M* (assuming that *M* is represented in binary). In this section we prove that an algorithm that behaves asymptotically better cannot exist, thus proving that the problem is EXPTIME-complete. To prove the inherent difficulty with respect to the multiple input parameters, we show two reductions: one parametric in the representation of *M* (the digit size), which assumes both *n* and *r* constant, and the other parametric in *r*, this time assuming constant both *n* and the value of *M*. We conjecture the existence of a third EXPTIME reduction, in which *r* and *M* are constant and the parameter is *n*. In fact, if it was not the case, it would be possible to improve the proposed model checking algorithm in a way that its complexity would not be exponential in *n*.

   We first recall the formalism of *linearly-bounded alternating Turing machines* (LB-ATM) and the notion of *hierarchical representation*, a succinct way of representing priced game structures inspired to the work done in [4] for classical Kripke structures. Finally, we present the two reductions from the acceptance problem for LB-ATM, known to be EXPTIME-complete [7], to the model checking problem for PRB-ATL.

4.1   Linearly-bounded alternating Turing Machines

A *linearly-bounded alternating Turing machines* (LB-ATM) is a tuple $\langle \mathscr{Q}, \Gamma, \mathscr{I}, q_0, \rangle$, where $\mathscr{Q}$ is the set of *states*, partitioned in $\mathscr{Q}_\forall$ (*universal states*) and $\mathscr{Q}_\exists$ (*existential states*); Γ is the set of *tape symbols*, including the 'blank' symbol B, and two special symbols $\llcorner$ and $\lrcorner$, denoting the left and right *tape delimiters*; $\mathscr{I} \subseteq \mathscr{Q} \times \Gamma \times \mathscr{Q} \times \Gamma \times \{\leftarrow, \rightarrow\}$ is the *instruction set*; $q_0 \in \mathscr{Q}$ is the *initial state*.

   Symbols from Γ are stored in the *tape cells*, and the first and the last cell of the tape store, respectively, the symbols $\llcorner$ and $\lrcorner$. A *tape configuration* s is a sequence of the symbols stored in the tape cells, and keeps trace of an *head cell*. A *configuration c* is a pair (q, s) of a state q and a tape configuration s, and $\mathscr{C}$ is the set of the configurations. The initial configuration is $c = (q_0, s_0)$, where s_0 contains the input, possibly followed by a sequence of blanks, and its head cell stores the first input symbol.

   An *instruction* $i = (q, \lambda, r, v, \sim) \in \mathscr{I}$ is also denoted $\langle q, \lambda \rangle \rightarrow \langle r, v, \sim \rangle$, where $\langle q, \lambda \rangle$ is called a *full state*. Its intuitive meaning is as follows: "whenever the machine is in the state q and the symbol in the head cell is λ, then the machine switches to state r, the symbol in the head cell is replaced with v, and the head position is moved to the left or to the right (according to $\sim$)". An *execution step* of the machine is denoted $c \xrightarrow{i} c'$, where $c, c' \in \mathscr{C}$, $i \in \mathscr{I}$ and c' is the configuration reached from c after the execution of the instruction *i*. Let $\mathscr{C}_{next(c)} = \{c' \in \mathscr{C} \mid c \xrightarrow{i} c'$ is an execution step, for some $i \in \mathscr{I}\}$. All the tape configurations are linear in the length of the input and we follow the common practice to only consider

machines whose tape length does not vary during the computation. We can also assume that LB-ATM have no infinite computations since any LB-ATM can be transformed into another, accepting the same language and haltingin a finite number of steps. Such a LB-ATM counts the number of execution steps and rejects any computation whose number of steps exceeds the number of possible configurations.

The *acceptance condition* is defined recursively. A configuration $c = (\mathsf{q}, \mathsf{s})$ is said to be *accepting* if either one of the following conditions is verified: (*i*) $q \in \mathcal{Q}_\forall$ and c' is accepting for all $c' \in \mathscr{C}_{next(c)}$ or (*ii*) $q \in \mathcal{Q}_\exists$ and there exists $c' \in \mathscr{C}_{next(c)}$ such that c' is accepting. Notice that an universal (existential) state always accepts (rejects) if $\mathscr{C}_{next(c)} = \emptyset$. A LB-ATM *accepts on an initial input tape* s_0, if the initial configuration $(\mathsf{q}_0, \mathsf{s}_0)$ is accepting.

Hierarchical representation. In order to exhibit our encoding proposal, we make use of a hierarchical representation analogous to the one described in [4, 12, 13] for model checking, and in [14] for module checking procedures. Given a finite state machine, the idea of hierarchical representation is to replace two or more substructures of the machine that are structurally equivalent, by another (structurally equivalent) module, that is a finite state machine itself. The use of hierarchical representation results in an exponentially more succinct representation of the system, that amounts (in most cases) to more efficient model checking procedures (in the other cases, this does not yield a more efficient behavior, as the analysis requires a flattening of the machine itself, thus incurring in an exponential blow up in its size).

In our context, this idea can be suitably adapted to deal with the presence of resources, as follows. Modules do not represent structurally equivalent substructures, but substructures that have the same impact on the values of resource variables. In principle, whenever the analysis is focused on the evolution of resource variables, it makes sense to consider as equivalent two substructures that can possibly differ in their structure but whose effect on the set of resource variables is exactly the same. This approach could be thought of as a hierarchical representation based on *functional* equivalence between substructures, as opposed to the classical notion of hierarchical representation based on *structural* equivalence.

4.2   A reduction from the acceptance problem for LB-ATM

Given an LB-ATM $\mathscr{A}$ and an input tape configuration s_0, we provide a priced game structure $G_{\mathscr{A}, \mathsf{s}_0}$, with two agents ag_1 and ag_2, and a formula $\phi_{\mathscr{A}, \mathsf{s}_0}$ such that $G_{\mathscr{A}, \mathsf{s}_0} \models \phi_{\mathscr{A}, \mathsf{s}_0}$ if and only if $\mathscr{A}$ accepts on s_0.

In the following, we exhibit the game structure by using a graphical (hierarchical) representation (Figures 2-7 in Appendix). Notice that only significant information is explicitly shown in the pictures. In particular, labels on transitions (arcs) represent consumptions/productions of resources due to the execution of the joint move (proponent and opponent moves) associated to that transition. For example, the label "$-1i, +1\bar{i}, +10\mu_L, -10\overline{\mu_L}$" on the loop transition of Figure 4b means that the actions associated to the transition will consume 1 unit of the (type) resource i and 10 unit of $\overline{\mu_L}$, and will produce 1 unit of the resource $\bar{i}$ and 10 unit of μ_L. Availability of other resources is unchanged, then the relative information is omitted.

The reduction uses the three resource variables μ_L, μ, and μ_R to encode the tape configuration, plus three auxiliary resource variables i, r, and t, that will be useful during the construction. Moreover, we associate to the above set of variables the set of *counterbalanced variables* $\{\overline{\mu_L}, \overline{\mu}, \overline{\mu_R}, \bar{i}, \bar{r}, \bar{t}\}$. The idea behind the use of counterbalanced variables, that is also the key idea of the reduction, consists of designing the game structure in a way that to every consumption (resp., production) of a resource, say for instance μ, a corresponding production (resp., consumption) of its counterbalanced $\overline{\mu}$ exists. In particular, this is true inside each module of the hierarchical structure, thus the sum of the availability of a resource variable and its counterbalanced variable is kept constant along all the computation at every module's entry and exit points, equal to a value *Max*, which depends on the input of the LB-ATM. This

will allow us to force the execution of specific transitions at specific availabilities of resource variables. Consider, for example, the node of Figure 4b with 2 outgoing transitions, one of which is a loop transition. The presence of 2 outgoing transitions means that either the proponent or the opponent can choose between 2 moves. But such a freedom is only potential, as in any moment of the computation the choice of the next move by the proponent/opponent is constrained by the resource availability: if the loop transition is enabled, then the availability of the resource i is greater than 0, and thus the availability of its counterbalanced variable $\bar{i}$ is less than *Max*, that means that the other transition, which consumes *Max* units of the resource $\bar{i}$, is disabled. On the contrary, if the non-loop transition is enabled, there are *Max* units of the resource $\bar{i}$ available, and thus the availability of the resource i is 0, that means that the loop transition is disabled. Thus, by taking advantage of the features of counterbalanced variables, we are able to force the executions to have a somehow deterministic behavior.

Encoding of the tape. Without loss of generality, we consider LB-ATM on input alphabet $\Sigma = \{1, 2, B\}$, thus Γ is the set $\{1, 2, B, \llcorner, \lrcorner\}$. Recall that the symbols B, $\llcorner$, and $\lrcorner$ denote the 'blank' symbol, the left delimiter, and the right delimiter, respectively. Tape symbols are encoded by the digits $0, 1, 2, 3$ and 4, in a pretty natural way: 0 encodes the 'blank' symbol, 1 and 2 encode the input symbols 1 and 2, and 3 and 4 encode the left and right delimiters. The tape configuration is encoded by means of the three resource variables μ_L, μ, and μ_R. The value of μ ranges over the set $\{0, 1, 2, 3, 4\}$ and encodes the value stored in the cell currently read by the head (according to the above encoding of tape symbols into digits). The value of μ_L encodes the tape configuration at the left of the current head position in a forward fashion. The value of μ_R encodes the tape configuration at the right of the current head position in a reverse fashion, that is, μ_R encodes the reverse of the string corresponding to the tape configuration at the right of current head position. As an example, consider the tape configuration $s = \llcorner B112\underline{1}1B2BB\lrcorner$, the symbol read by the head is the underlined one. Such a configuration is encoded by means of the three resource variables as follows: $\mu_L = 30112$, $\mu_L = 1$, and $\mu_R = 400201$. It can be noticed that the length of the representation of the three variables μ_L, μ, and μ_R is proportional to the length of the tape configuration which is at most linear in the size of the input, namely $O(|s_0|)$. Using such an encoding, the machine operation "shift the head to the left" can be represented by means of the following operations on resource variables:

- the new value of μ_R is $\mu_R * 10 + \mu$
- the new value of μ is $\mu_L \mod 10$,
- the new value of μ_L is $\mu_L / 10$ (/ is the integer division),

The operation "shift the head to the right" can be encoded analogously.

Notice that in order to encode in polynomial time the operations of shifting the head to left and right, we encode the string to the right of the current head position in a reverse order. Indeed, in this way the symbol stored on the cell immediately to the right of the head corresponds to the least significant digit of μ_R, and thus can be accessed by using the module operation ($\mu_R \mod 10$).

Encoding of the instructions. The encoding of the instructions is depicted in Figure 2. Transitions starting from a node labeled $\langle q, \lambda \rangle$ represent all the possible instructions matching the full state $\langle q, \lambda \rangle$ of the LB-ATM, that is, all the instructions that can be possibly performed at the full state $\langle q, \lambda \rangle$.

More in detail, given a full state $\langle q, \lambda \rangle$ of the machine, with $q \in \mathcal{Q}_\exists$, the encoding of the set $\{\langle q, \lambda \rangle \rightarrow \langle r_1, v_1, \sim_1 \rangle, \langle q, \lambda \rangle \rightarrow \langle r_2, v_2, \sim_2 \rangle, \ldots, \langle q, \lambda \rangle \rightarrow \langle r_m, v_m, \sim_m \rangle\}$ of matching instructions is shown in Figure 2a, (recall that $\sim_i \in \{\leftarrow, \rightarrow\}$). Analogously, the encoding of the set of instructions matching the full state $\langle q, \lambda \rangle$, with $q \in \mathcal{Q}_\forall$, is shown in Figure 2b. Let us underline that the action profiles $\langle \alpha_1, \beta \rangle, \ldots, \langle \alpha_m, \beta \rangle$ labeling transitions corresponding to an existential state are such that the first agent ag_1 has the capability to force a specific transition (instruction) to be executed, depending on the
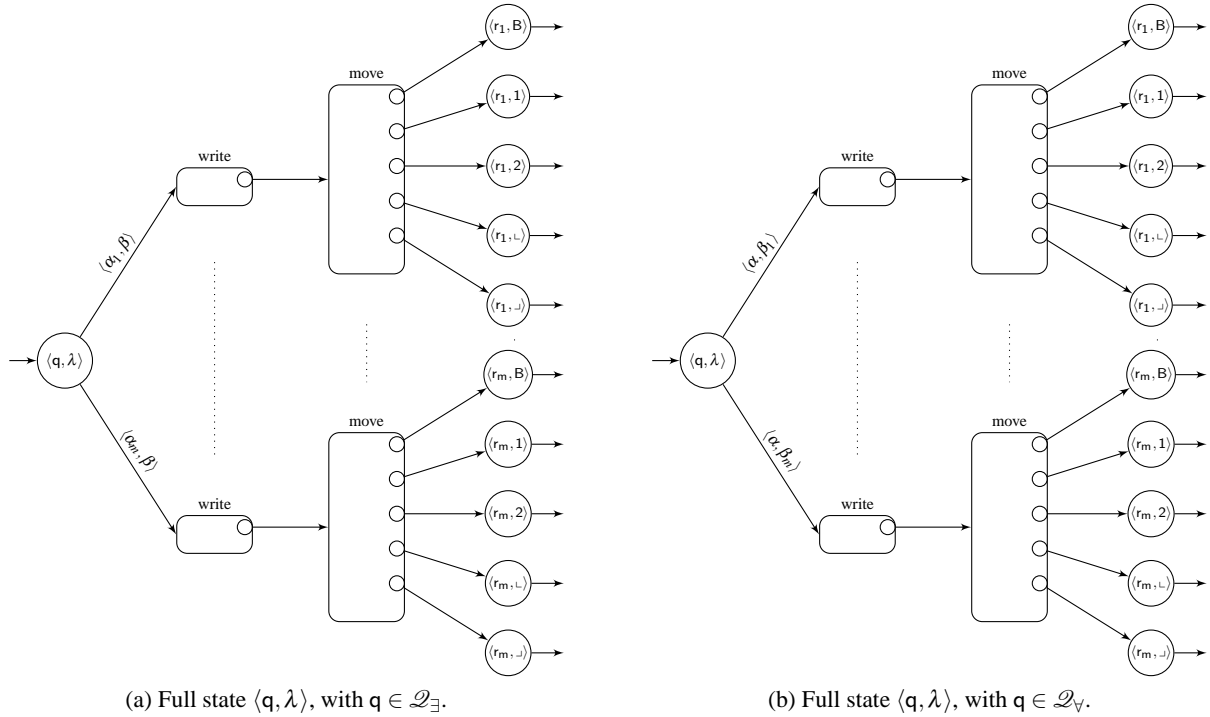
(a) Full state $\langle q, \lambda \rangle$, with $q \in \mathcal{Q}_\exists$.

(b) Full state $\langle q, \lambda \rangle$, with $q \in \mathcal{Q}_\forall$.

Figure 2: Encoding of the set of instructions matching a full state $\langle q, \lambda \rangle$ of a LB-ATM.

choice of the α_i for the next action, independently from the choice β of the other agent ag_2. On the other hand, the action profiles $\langle \alpha, \beta_1 \rangle, \ldots, \langle \alpha, \beta_m \rangle$ labeling transitions corresponding to an universal state are such that the roles of the agents are exchanged.

The LB-ATM representation of Figure 2 is hierarchical and involves the modules *write* and *move*. The former encodes the rewriting of the head cell performed by $\mathscr{A}$ and, to this aim, makes use of one of the following modules (Figure 3), depending on the symbol λ read by the head, and on the symbol ν to be written:

- *inc*, depicted in Figure 3a, is used when the rewriting corresponds to an increment, for example, when the symbol 2 has to be written in place of the symbol 1;
- *double_inc*, depicted in Figure 3b, is used when the rewriting corresponds to a double increment, for example, when the symbol 2 (encoded as 2) has to be written in place of the symbol B (encoded as 0);
- *dec*, depicted in Figure 3c, is used when the rewriting corresponds to a decrement, for example, when the symbol 1 has to be written in place of the symbol 2;
- *double_dec*, depicted in Figure 3d, is used when the rewriting corresponds to a double decrement, for example, when the symbol B has to be written in place of the symbol 2.

Obviously, the module does nothing when the symbol to be written corresponds to the symbol currently stored in the head cell.

The module *move* encodes the shift (to right or to left) of the head. It is designed in a way that the only next location that can be reached by the game is consistent with the value stored on the new head cell (after the shift operation). In Figure 4 and 5 the sub-modules encoding the operation "shift to right" are depicted. The encoding of the operation "shift to left" can be realized analogously.

As an example, we describe the first two modules of Figure 4. The module *shift_right*, depicted in

Figure 4a, is performed through the following steps:

- multiply by 10 the value of μ_L (module *times_10*(μ_L)),
- increment the value of μ_L by the value of μ (module *add*(μ_L, μ)),
- divide by 10 the value of μ_R (module *div_10*(μ_R) — the remainder of the division is stored in the resource variable *r*),
- assign to the resource variable μ the value of *r* (module *assign*(μ, r)),
- suitably lead the computation to the location corresponding to the next state of the LB-ATM, depending on the value read by the head, that is, the value stored on the resource variable μ (module *choose_next_state*(μ)).

The module *times_10*(μ_L), that multiplies by 10 the value of μ_L (Figure 4b), is performed by storing the value of μ_L in the resource variable *i*, by setting the value of μ_L to 0, and then by executing a transition (the loop transition), which consumes 1 unit of *i* and produces 10 units of μ_L (the suitable quantity of the counterbalanced variables is produced or consumed as well, to keep the sum constant) as long as items of the resource *i* are available. When the availability of *i* goes down to 0, the other transition is executed (the last transition is needed to keep constant the sum between *i* and its counterbalanced variable $\bar{i}$). It is easy to convince oneself that the value of μ_L in the exit node is equal to its value in the entry node times 10, and that the sum of each variable and its counterbalanced one is constant. As a last remark, we point out that the names of some of the modules are parametric, in the sense that the arguments between parenthesis are not actual resource variables, but parameters (e.g., *x*, x_1, x_2) to be instantiated. We adopted this notation for modules that are used more than once, and that are instantiated with actual resource variables when they are used (e.g., the module *assign* depicted in Figure 4c is called *assign*(x_1, x_2) and it is used, for instance, inside the module *times_10*(μ_L) (Figure 4b), where x_1 (resp., x_2) is instantiated with *i* (resp., μ_L), and inside the module *add*(μ_L, μ) (Figure 5a), where x_1 (resp., x_2) is instantiated with *t* (resp., μ).

Now, as resource productions are involved in the reduction, we need to guarantee that the availability of each resource never exceeds the initial one. To this end the values of the components of the vector $\vec{m}_0$ of initial resource availability are set to the value $Max = 322\ldots224$, that is the largest number corresponding to an encoding of any tape configuration (precisely, it encodes the tape configuration $\llcorner 22\ldots 22 \lrcorner$). Before starting the simulation of the LB-ATM, a preliminary step, depicted in Figure 6, modifies the value of the resource variables in such a way that they correctly encode the input tape s_0 and the sum of the availability of each resource variable and its counterbalanced is equal to *Max*. Thus, the value of the resource variables never exceed *Max*.

At this point, given a LB-ATM $\mathscr{A}$ and an input tape configuration s_0, the game structure $G_{\mathscr{A}, s_0}$ presents, among others, the following features (the other features of $G_{\mathscr{A}, s_0}$ are either irrelevant or represented in the graphical representation of the encoding — see Figures 2-6):

- 2 agents, ag_1 and ag_2;
- 5 locations, namely $\langle q, B \rangle$, $\langle q, 1 \rangle$, $\langle q, 2 \rangle$, $\langle q, \llcorner \rangle$, $\langle q, \lrcorner \rangle$, for each internal state q of $\mathscr{A}$ (plus other locations — the circles in the pictures — that do not correspond to particular states of the LB-ATM, but are needed to perform the encoding);
- only one atomic proposition *p*, that holds true over all and only the locations having no matching
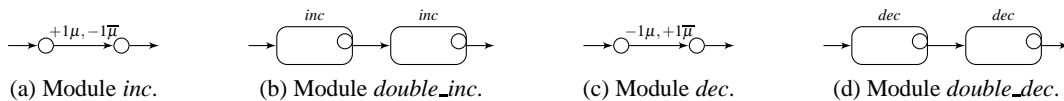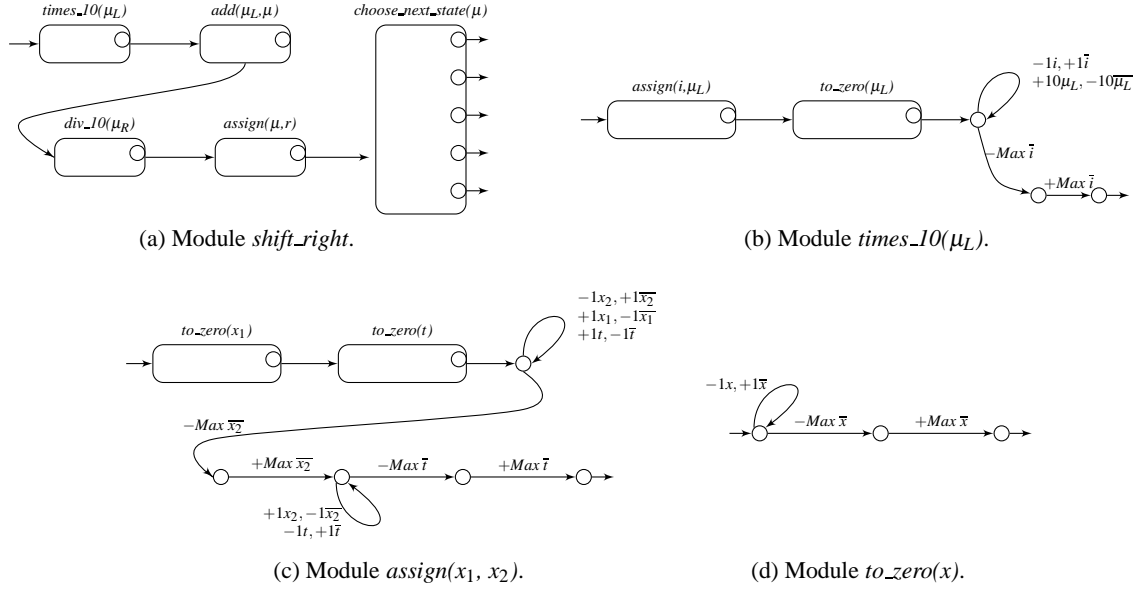

(a) Module *inc*.   (b) Module *double_inc*.   (c) Module *dec*.   (d) Module *double_dec*.

Figure 3: Encoding of the module *write*.

(a) Module *shift_right*.

(b) Module *times_10(μ_L)*.

(c) Module *assign(x_1, x_2)*.

(d) Module *to_zero(x)*.

Figure 4: Encoding of the module *shift_right* - part I.

instructions;

- initial global availability $\vec{m}_0$ is such that all resources are available in quantity *Max*, as already mentioned above; notice that *Max* also represents the maximum value occurring in the initial resource availability vector, that is, $M = Max$;
- initial location $\langle q_0, \lambda \rangle$, where q_0 is the initial state of the LB-ATM and λ is the first input symbol.

The formula $\phi_{\mathscr{A}, s_0} = \langle\langle A^{\vec{\$}} \rangle\rangle \Diamond p$, with $A = \{ag_1\}$ and the value of $\vec{\$}$ being irrelevant for our purposes, is such that $G_{\mathscr{A}, s_0} \models \phi_{\mathscr{A}, s_0}$ if and only if $\mathscr{A}$ accepts on input s_0.

Notice that, for the sake of readability, the game structure used in the reduction does not respect the requirement that, in every location, the first action of every agent is the *do-nothing* action, which does not consume or produce resources. Nevertheless, this omission does not affect the correctness of our reduction, that can be easily adapted using a game structure fulfilling the above requirement.

**Theorem 1** *Model checking* PRB-ATL *is EXPTIME-hard even assuming n and r constant.*

Let us stress that the above reduction makes use of a constant number of agents and resources, while the digit size of M (the maximum value occurring in $\vec{m}_0$) is linear in the size of the tape configuration. This is consistent with the complexity of the algorithm in [9], which remains exponential even if we consider a constant number of agents and resources as input.

Corollary 1 *The model checking problem for* PRB-ATL *is EXPTIME-complete.*

4.3 Another reduction.

As noted at the beginning of Section 4, it is possible to exhibit two more reductions according to which two parameters, out of three, are assumed constant. In the following, we briefly outline how to obtain a reduction from the acceptance problem for LB-ATM, when *n* and *M* are constant.

This reduction is simpler than the previous. Here the encoding of the tape is obtained using a number of resources which is linear in the length of the tape. Let |s| be the length of the tape, we use 2 sets of

(a) Module *add(μ_L, μ)*.

(b) Module *div_10(μ_R)*.

(c) Module *choose_next_state(μ)*.
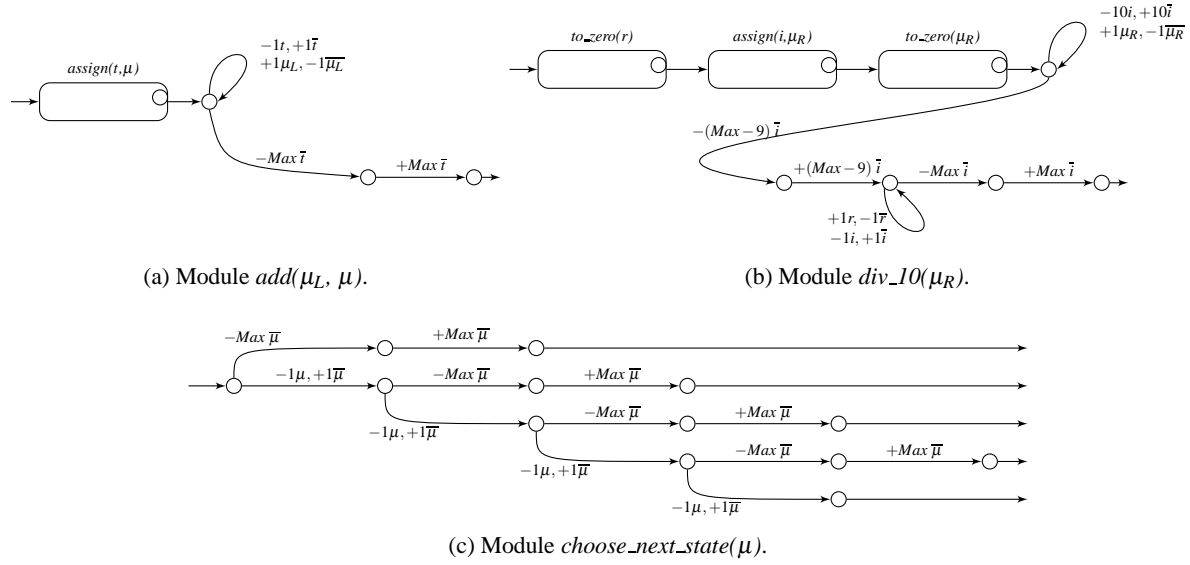
Figure 5: Encoding of the module *shift_right* - part II.

$|s|$ resource variables, namely, $\mu_L^1, \mu_L^2, \ldots, \mu_L^{|s|}$ and $\mu_R^1, \mu_R^2, \ldots, \mu_R^{|s|}$, plus the resource variable μ. Each variable encodes the content of a tape cell: variable μ encodes the content of the head cell, while, for each i, the variable μ_L^i (resp., μ_R^i) encodes the content of the i-th cell on the left (resp., right) of the tape cell. Notice that, since there are finitely many possible values for a tape cell, the value of M is upper bounded. Now, the encoding of the set of instructions matching a full state $\langle q, \lambda \rangle$ of a LB-ATM is the same used for the previous reduction and depicted in Figure 2. Nevertheless, the encoding of the module *move*, which encodes the shift (to right or to left) of the head, is slightly different. In Figure 7, the sub-modules encoding the operation "shift to right" are depicted. Essentially, the value of the variable representing a cell is transmitted to the variable representing the cell on the right, and the next location reached on the game structure is set according to the value stored on the current head cell (after the shift operation). The encoding of the operation "shift to left" is made analogously.

Theorem 2 *Model checking* PRB-ATL *is EXPTIME-hard even assuming n and M constant.*

5 Discussion

In this paper we have presented a formalism which is very suitable to model properties of multi-agent systems when the agents share resources and the need of avoiding an unbounded consumption of such resources is crucial. Within our framework it is possible to keep trace of a real global availability of the
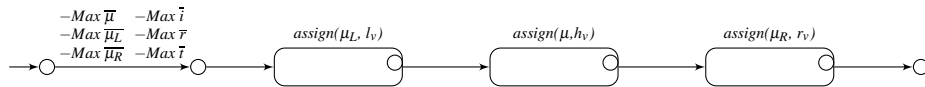
Figure 6: Preliminary step of the reduction (l_v, h_v, and r_v encode the input tape configuration).
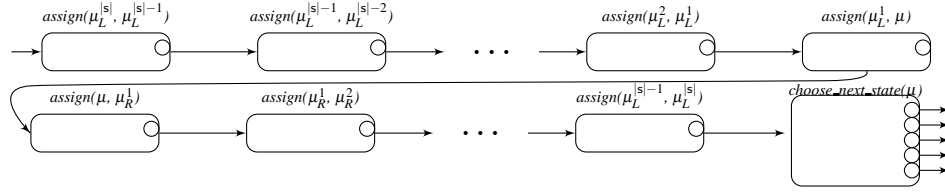
Figure 7: Encoding of the module *shift_right*.

resources, used by both the proponent and opponent players, avoiding thus unrealistic situations in which an unbounded quantity of resources is used in a game.

The technical focus of the paper has been on the complexity of the model checking problem, and we proved that it is EXPTIME complete (recall that also for simpler formalism this problem is in EXPTIME, though the lower bound is not known). Other problems of interest exist in the context of multi-agents system verification. The most important one is the *reachability problem*, that is the problem of determining whether a team, with a given amount of money and a given initial global resource availability, has a strategy to force the execution of the system to reach a given location. More precisely, the reachability problem for a team A on a priced game structure G is a particular instance of the model checking problem, namely, the problem of verifying the truth at the initial configuration of G of a PRB-ATL formula of the kind $\langle\langle A^{\vec{\$}}\rangle\rangle\Diamond p$, for a team A, a money endowment $\vec{\$}$ and $p \in \Pi$. An upper bound on the complexity of this problem is clearly given by the algorithm for solving the model checking problem for PRB-ATL. Let us observe that the reductions given in section 4 apply also to the reachability problem, since the formula used there was $\phi_{\mathscr{A},s_0} = \langle\langle A^{\vec{\$}}\rangle\rangle\Diamond p$, thus we have the following corollary.

Corollary 2 *The reachability problem for* PRB-ATL *is EXPTIME-complete.*

One of the novelties of our logic is that *the resource production* is allowed in the actions, though with some limitations. Model checking and reachability problems seem both to be simpler in the case one restricts our formalism by considering agent actions that cannot produce resources. The reachability problem is indeed NP-hard in this case: it immediately follows from a result in [11], when the number of agents is not constant. Anyway, we can prove the NP-hardness for just two agents using a reduction from 3-SAT (due to lack of space we omit here the proof). The model checking problem, instead, turns out to be PSPACE-hard, since the reduction from QBF problem given in [9] works also in this case, when actions cannot produce resources. Observe that PRB-ATL with this restriction is again different from other formalisms in literature, mainly for the possibility of tracking resources avalability and for considering shared resources.

Finally, we want to note that also the more general problem, called *optimal coalition problem*, is EXPTIME-complete (the upper bound was shown in [9]). It is the problem of finding optimal (with respect to a suitable cost function) coalitions that are capable to satisfy a given *parametric* PRB-ATL formula, that is, a PRB-ATL formula in which *parametric team operators* $\langle\langle X^{\vec{\$}}\rangle\rangle$ may occur in place of the classical team operators $\langle\langle A^{\vec{\$}}\rangle\rangle$. One could also investigate other optimization problems. The most interesting is, perhaps, to consider the money availability not as an input of the problem, but rather as a parameter to minimize, that is to establish how much money each agent should be provided with, to perform a given task.

Further research directions concern the study of variants of the logic. First, one can consider extensions based on the full alternating-time temporal language ATL*, as already done in [6], and its fragment ATL$^+$.

References

[1] Natasha Alechina, Brian Logan, Nguyen Hoang Nga & Abdur Rakib (2009): *A Logic for Coalitions with Bounded Resources*. In: *Proc. of the 21st International Joint Conference on Artificial Intelligence*, IJCAI '09, pp. 659–664.

[2] Natasha Alechina, Brian Logan, Nguyen Hoang Nga & Abdur Rakib (2010): *Resource-bounded alternating-time temporal logic*. In: *Proc. of the 9th International Conference on Autonomous Agents and Multiagent Systems: Volume 1*, AAMAS '10, pp. 481–488.

[3] Rajeev Alur, Thomas A. Henzinger & Orna Kupferman (2002): *Alternating-time temporal logic*. Journal of ACM 49(5), pp. 672–713, doi:10.1145/585265.585270.

[4] Rajeev Alur & Mihalis Yannakakis (2001): *Model checking of hierarchical state machines*. ACM Transactions on Programming Languages and Systems (TOPLAS) 23(3), pp. 273–303, doi:10.1145/503502.503503.

[5] Nils Bulling & Berndt Farwer (2009): *Expressing Properties of Resource-Bounded Systems: The Logics RTL* and RTL*. In Jürgen Dix, Michael Fisher & Peter Novák, editors: *Computational Logic in Multi-Agent Systems (CLIMA X)*, Springer, pp. 22–45, doi:10.1007/978-3-642-16867-3_2.

[6] Nils Bulling & Berndt Farwer (2010): *On the (Un-)Decidability of Model Checking Resource-Bounded Agents*. In: *Proc. of the 19th European Conference on Artificial Intelligence*, ECAI '10, pp. 567–572, doi:10.3233/978-1-60750-606-5-567.

[7] Ashok K. Chandra, Dexter C. Kozen & Larry J. Stockmeyer (1981): *Alternation*. Journal of ACM 28(1), pp. 114–133, doi:10.1145/322234.322243.

[8] Mehdi Dastani, Koen V. Hindriks & John-Jules Charles Meyer, editors (2010): *Specification and Verification of Multi-agent Systems*, 1st edition. Springer Publishing Company, Incorporated.

[9] D. Della Monica, M. Napoli & M. Parente (2011): *On a Logic for Coalitional Games with Priced-Resource Agents*. Electronic Notes in Theoretical Computer Science (ENTCS) 278, pp. 215–228, doi:10.1016/j.entcs.2011.10.017. Proc. of the 7th Workshop on Methods for Modalities (M4M 2011) and the 4th Workshop on Logical Aspects of Multi-Agent Systems (LAMAS 2011).

[10] Valentin Goranko (2001): *Coalition games and alternating temporal logics*. In: *Proc. of the 8th Conference on Theoretical Aspects of Rationality and Knowledge*, TARK '01, Morgan Kaufmann, pp. 259–272.

[11] Wojciech Jamroga & Jürgen Dix (2005): *Do Agents Make Model Checking Explode (Computationally)?* In: *Proc. of the 4th International Central and Eastern European Conference on Multi-Agent Systems (CEEMAS 2005)*, Lecture Notes in Computer Science 3690, Springer, pp. 398–407, doi:10.1007/11559221_40.

[12] Salvatore La Torre, Margherita Napoli, Mimmo Parente & Gennaro Parlato (2003): *Hierarchical and Recursive State Machines with Context-Dependent Properties*. In Jos C. M. Baeten, Jan Karel Lenstra, Joachim Parrow & Gerhard J. Woeginger, editors: *Proc. of the 30th International Colloquium on Automata, Languages and Programming (ICALP)*, Lecture Notes in Computer Science 2719, Springer, pp. 776–789, doi:10.1007/3-540-45061-0_61.

[13] Salvatore La Torre, Margherita Napoli, Mimmo Parente & Gennaro Parlato (2008): *Verification of scope-dependent hierarchical state machines*. Information and Computation 206(9-10), pp. 1161–1177, doi:10.1016/j.ic.2008.03.017.

[14] Aniello Murano, Margherita Napoli & Mimmo Parente (2008): *Program Complexity in Hierarchical Module Checking*. In Iliano Cervesato, Helmut Veith & Andrei Voronkov, editors: *Proc. of the 15th International Conference on Logic for Programming, Artificial Intelligence, and Reasoning (LPAR)*, Lecture Notes in Computer Science 5330, Springer, pp. 318–332, doi:10.1007/978-3-540-89439-1_23.

[15] Marc Pauly (2001): *A Logical Framework for Coalitional Effectivity in Dynamic Procedures*. Bulletin of Economic Research 53(4), pp. 305–324, doi:10.1111/1467-8586.00136.

[16] Marc Pauly (2002): *A Modal Logic for Coalitional Power in Games*. Journal of Logic and Computation 12(1), pp. 149–166, doi:10.1093/logcom/12.1.149.