Artificial Intelligence 245 (2017) 56–85

Contents lists available at ScienceDirect

Artificial Intelligence

www.elsevier.com/locate/artint

The virtues of idleness: A decidable fragment of resource agent logic

Natasha Alechina [a,*], Nils Bulling [b,c], Brian Logan [a], Hoang Nga Nguyen [d]

[a] School of Computer Science, University of Nottingham, UK
[b] Delft University of Technology, The Netherlands
[c] TU Clausthal, Germany
[d] Centre for Mobility and Transport, Coventry University, UK

ARTICLE INFO

Article history:
Received 26 April 2016
Received in revised form 13 December 2016
Accepted 30 December 2016
Available online 4 January 2017

Keywords:
Strategy logics
Resource constraints
Model checking

ABSTRACT

Alternating Time Temporal Logic (ATL) is widely used for the verification of multi-agent systems. We consider Resource Agent Logic (RAL), which extends ATL to allow the verification of properties of systems where agents act under resource constraints. The model checking problem for RAL with unbounded production and consumption of resources is known to be undecidable. We review existing (un)decidability results for fragments of RAL, tighten some existing undecidability results, and identify several aspects which affect decidability of model checking. One of these aspects is the availability of a 'do nothing', or *idle* action, which does not produce or consume resources. Analysis of undecidability results allows us to identify a significant new fragment of RAL for which model checking is decidable.

1. Introduction

Many problems in AI and multi-agent systems research are most naturally formulated in terms of the abilities of a group or coalition of agents. For example, a group of agents may be able to cooperate to achieve an outcome which cannot be achieved by any agent in the group acting individually. In many cases, whether the outcome can be achieved depends critically on the resources available to the agents. Money is an obvious example, but there are many kinds of resources that may be produced or consumed by the actions of agents. For example, whether a team of agents can cooperate to extinguish a fire may depend on the amount of fuel and water they have available. Several logics for reasoning about coalitional ability under resource bounds have been proposed in the literature [1–7]. These *resource logics* allow us to express properties such as: 'a coalition of agents A has a strategy (a choice of actions) requiring no more than b resources, such that whatever the actions by the agents outside the coalition, any evolution of the system generated by the strategy satisfies some temporal property'. Using model checking techniques we can then verify that a given coalition has a strategy requiring less than b resources to enforce an outcome, whatever the other agents in the system (or the environment) do. The ability to verify such properties can be useful when designing or developing a resource-constrained multi-agent system.

Unfortunately, the model checking problem for many resource logics where actions can produce resources is undecidable [2,5]. Recently, however, it was shown that some resource logics where actions can produce resources have a decidable

* Corresponding author.
E-mail addresses: nza@cs.nott.ac.uk (N. Alechina), Bulling@in.tu-clausthal.de (N. Bulling), bsl@cs.nott.ac.uk (B. Logan), hoang.nguyen@coventry.ac.uk (H.N. Nguyen).

http://dx.doi.org/10.1016/j.artint.2016.12.005

model checking problem [6–8].[1] In this paper, we investigate the reasons for the decidability or undecidability of the model checking problem for resource logics. Different syntactic and semantic choices give different variants of resource logics. Some of these choices are known to affect the decidability of the model checking problem. In particular, the decidability result in [6] was proven in the presence of two major restrictions, called, in the terminology of [2], *resource flat* and *proponent restricted*. The former assumes that agents are always re-equipped with fresh resources when they reconsider their strategies; the latter assumes that only the proponents act under resource bounds (i.e., agents outside the coalition are not resource bounded). In addition to these restrictions, another choice in the semantics is relevant for the decidability result in [6]. This choice, which is also related to the finitary and infinitary semantics of [2], stipulates that, in every model, agents always have a choice of doing nothing (executing an idle action) that produces and consumes no resources. Having an *idle* action makes model checking easier: intuitively, its availability ensures that in order to determine whether a coalition can enforce a ϕ-state after finitely many steps and within a given resource bound, we only need to find a finite strategy to enforce ϕ under the given resource bound, and after ϕ is achieved, the agents can always choose the idle action forever, which does not increase the 'cost' of the strategy. The presence of an idle action in the logic also guarantees some attractive formal properties. For example, as stated in [3], it ensures coalition monotonicity: if a coalition A can ensure a property under resource bound b, then any larger coalition can also ensure this property under the same resource bound (intuitively, the extra agents can always perform idle).

In this paper, we investigate the effects of various semantic choices, such as the availability of an idle action, on the decidability of the model checking problem for resource logics. First we show that both the resource-flat and the proponent-restricted fragments of resource agent logic remain undecidable in the presence of idle actions. We then identify and motivate a significant, non-resource-flat fragment that has a decidable model checking property in the presence of idle actions, and is not decidable otherwise. It follows that idle actions can make a difference for the decidability of model checking with respect to the semantics we consider.

The new fragment, which we call pprRAL, allows us to express statements about the existence of nested strategies for a coalition of agents given some *initial* allocation of resources. Unlike the resource-flat fragment considered in [6], where for each new strategy agents are re-equipped with a fresh set of resources, pprRAL allows us to express properties such as 'given their initial battery charge, rescue robots A can safely get to a position from which they can perform rescue while in visual contact with the base'. There are two nested strategies implicit in this property: first, the robots should be able to reach some position (not necessarily maintaining visual contact with the base), and second, from this position, the agents should be able to perform rescue while in visual contact with the base. The first strategy (getting into position) will require certain resources (in this case battery charge), and the amount of resources required will depend on the environment. Then, with *whatever resources are left*, the agents need a strategy to perform the rescue. In this example, the model checking problem essentially corresponds to finding two nested conditional resource-constrained plans, see e.g., [10]. The plans are nested because it is impossible to decouple the second plan (for rescue) from the results of the first plan (getting into position), since we do not know the resource availability for the initial state of the second plan; the resource availability in that state is determined by resource consumption of the first plan. Compared to conditional planning with resources, resource logics provide an easy way to talk not just about reachability, but also about invariants and nested goals/strategies achieved by (potentially different) coalitions.

This paper extends results presented in [7] in several respects, including: a more general definition of a decidable fragment, more elaborated intuitions regarding the (un)decidability results, detailed proofs of all theorems, and tighter undecidability results (in terms of the number of agents and resource types required for undecidability). The remainder of the paper is organised as follows. In Section 2 we briefly survey related work. In Section 3 we introduce resource agent logic, its models and the semantics. In Section 4, we review known decidability results for resource agent logic, and investigate the reasons for (un)decidability. We present new undecidability results for systems with a single resource type, and, based on these results, we motivate and introduce a new non-resource flat fragment of RAL, pprRAL. In Section 5, we present our second main technical result: a decidability result for pprRAL. We conclude in Section 6.

2. Related work

Early work on resource logics considered only the consumption of resources (i.e., no action produces resources), and initial results on the complexity of model checking were encouraging. One of the first logics capable of expressing resource requirements of agents was a version of Coalition Logic (CL)[2] called Resource-Bounded Coalition Logic (RBCL), where actions only consume (and do not produce) resources. It was introduced in [1] with the primary motivation of modelling systems of resource-bounded reasoners; however the framework is sufficiently general to model any type of action. The model checking problem for RBCL was shown to be decidable in time polynomial in the size of the transition system and of the property, and exponential in the number of resource types in [12]. A resource-bounded version of ATL, RB-ATL, where again actions only consume (and do not produce) resources was introduced in [3]. The model checking problem for this logic is also decidable in time polynomial in the size of the transition system and of the property, and exponential in the number of

[1] A preliminary version of [8] is available as a technical report [9].

[2] CL is a fragment of ATL with only the next time $\langle\langle A\rangle\rangle\mathbf{X}$ modality, introduced in [11].

resource types [3]. (For a single resource type, e.g., energy, the model checking problem is no harder than for ATL.) Practical work on model checking many standard computer science transition systems (not multi-agent systems) with resources also falls in the category of consumption-only systems. For example, probabilistic model checking of systems with numerical resources as in the PRISM model checker [13] assumes that costs increase monotonically with time.

However, when resource production is considered in addition to consumption, the situation changes. In a separate strand of work, a range of different formalisms for reasoning about resources was introduced in [14,2], including Resource Agent Logic RAL which is the main focus of this paper. In these formalisms, both consumption and production of resources was considered. In [2], it was shown that the model checking problem for most variants and fragments of RAL is undecidable. The only decidable cases considered in [2] (and the related [14]) are an extension of Computation Tree Logic (CTL) with resources (essentially one-agent ATL), and a version where on every path only a fixed finite amount of resources can be produced. The models satisfying this property were referred to as bounded in [2]. It was pointed out in [2] that RBCL and RB-ATL are logics over a special kind of bounded models (where no resources are produced at all). Other decidability results for bounded resource logics have also been reported in the literature. For example, in [15] a decidable logic, PRB-ATL (Priced Resource-Bounded ATL) is defined, where the total amount of resources in the system has a fixed bound. The model checking algorithm for PRB-ATL requires time polynomial in the size of the transition system, and exponential in the number of resource types and the resource bound on the system. In [4] an EXPTIME lower bound in the number of resource types for the PRB-ATL model checking problem is shown. In [16], an extension of PRB-ATL to μ-calculus is also shown to have a decidable model checking problem.

A general logic over systems with numerical constraints, Quantitative ATL (QATL*), was introduced in [5], and undecidability results for the model checking problem for QATL* and some of its fragments were shown. For example, QATL is undecidable even if no nestings of cooperation modalities are allowed. The main proposals for restoring decidability to the model checking problem for QATL in [5] are removing negative payoffs (similar to removing resource production), and introducing memoryless strategies (the latter idea is not pursued in any detail).

This brief survey of work suggests that the boundary between decidability and undecidability for the model checking problem of resource logics is very subtle. No systematic study of the reasons for decidability and undecidability of this problem has been undertaken to date, and with this paper we aim to address this task. We believe a better understanding of the boundary between decidability and undecidability will be useful in developing new decidable fragments of resource logics.

Of course, searching for decidable fragments is not the only way of addressing the undecidability of model checking for temporal logics with infinite-state transition systems (RAL can be seen as a special case of such logics). Another approach is to design algorithms which return definite answers where possible, and 'unknown' otherwise (see, e.g., [17–21]). A promising direction of future research would be to explore connections between the two approaches. Our work connects to many other areas of computer science, such as planning [10] and the verification of autonomous systems [22,23]. The model checking problem can essentially be seen as an approach to computing a robust plan for a set of autonomous agents, such as robots. Techniques used in our work are related to many subfields of theoretical computer science. In particular, techniques developed for Petri nets, vector addition systems and model checking over pushdown systems (see, e.g., [24,25]), are closely connected to the techniques we use in establishing our decidability and undecidability results. The existing, deep theoretical results in these areas also provide a starting point for establishing complexity bounds for our model checking algorithms, and ideas for restrictions that give fragments with good computational properties. Finally, another branch of work on reasoning about resources is based on linear logic [26,27], and related logics such as the logic of bunched implication [28–31]. In the future, it would be interesting to explore deeper connections between the resource logics considered in this paper, and reasoning about resources using linear logic techniques.

3. Resource agent logic

In this section we define *resource agent logic* (RAL) and *resource-bounded models* (**RBM**s). We essentially follow [2], combined with aspects from [6]. We summarise the similarities and differences between RAL and the resource logics considered in [2,6] in more detail in Section 3.5.

3.1. Syntax of RAL

The logic is defined over a set of agents $\mathbb{A}\mathrm{gt}$, a set of resources types *Res*, and a set of propositional symbols Π. We denote the set of natural numbers by $\mathbb{N}$, the set of natural numbers with zero by $\mathbb{N}_0$, the set of natural numbers with infinity by $\mathbb{N}^\infty$, and the set of natural numbers with zero and infinity by $\mathbb{N}_0^\infty$. An *endowment (function)* $\eta : \mathbb{A}\mathrm{gt} \times Res \to \mathbb{N}_0^\infty$ assigns resources to agents; $\eta_a(r) = \eta(a, r)$ is the number of resources agent a has of resource type r. En denotes the set of all possible endowments. Resource types can represent, for example, money, fuel, battery power, etc. Special minimal and maximal endowment functions are denoted by $\bar{0}$ and $\bar{\infty}$, respectively. The former expresses that there are no resources at all, whereas the latter equips all agents with an infinite amount of each resource type. (In what follows, for readability we will talk about amounts of some resource, rather than of some resource type.) The logic RAL is defined according to the grammar of ATL [32]. RAL-formulae are defined by:

$$\phi ::= \mathsf{p} \mid \neg\phi \mid \phi \wedge \phi \mid \langle\langle A\rangle\rangle^{\downarrow}_{B}\mathbf{X}\varphi \mid \langle\langle A\rangle\rangle^{\eta}_{B}\mathbf{X}\varphi \mid \langle\langle A\rangle\rangle^{\downarrow}_{B}\varphi\mathbf{U}\psi \mid \langle\langle A\rangle\rangle^{\eta}_{B}\varphi\mathbf{U}\psi \mid \langle\langle A\rangle\rangle^{\downarrow}_{B}\mathbf{G}\varphi \mid \langle\langle A\rangle\rangle^{\eta}_{B}\mathbf{G}\varphi$$

where $p \in \Pi$ is a proposition, $A, B \subseteq \mathbb{A}\text{gt}$ are sets of agents, and η is an endowment. We also define $\langle\langle A\rangle\rangle^{\downarrow}$ and $\langle\langle A\rangle\rangle^{\eta}$ as abbreviations for $\langle\langle A\rangle\rangle^{\downarrow}_{A}$ and $\langle\langle A\rangle\rangle^{\eta}_{A}$, respectively. The operators **X**, **U**, and **G** denote the standard temporal operators expressing that some property holds in the *next* point in time, *until* some other property holds, and *now and always* in the future, respectively. There are two types of cooperation modalities, $\langle\langle A\rangle\rangle^{\downarrow}_{B}$ and $\langle\langle A\rangle\rangle^{\eta}_{B}$. In both types of cooperation modality, the actions performed by agents in $A \cup B$ consume and produce resources (actions by agents in $\mathbb{A}\text{gt} \setminus (A \cup B)$ do not change their resource endowment). The reading of $\langle\langle A\rangle\rangle^{\eta}_{B}\varphi$ is that *when agents $A \cup B$ have a resource endowment η, agents A have a strategy compatible with this endowment to enforce φ (whatever the agents in $\mathbb{A}\text{gt} \setminus A$ do, compatible with their resource constraints, if any)*. The evaluation of a modality $\langle\langle A\rangle\rangle^{\eta}_{B}$ (re-)equips all agents with a *fresh* amount of resources: the current resource endowment is overwritten by endowment η. The formula $\langle\langle A\rangle\rangle^{\downarrow}_{B}\varphi$ reads similarly but the strategy must be compatible with the resources *currently* available to the agents. In both cases compatible means that the strategy can be executed given the agents' resources. For both modalities it is therefore necessary to keep track of resource production and consumption during the execution of a strategy.

3.2. Semantics of RAL

We define the models of RAL as in [2]. Following [6] we also define a special case of these models in which all agents have an *idle action* in their repertoire which neither consumes nor produces resources.

Definition 1 (**RBM**, **iRBM**). A resource-bounded model (**RBM**) is given by $\mathfrak{M} = (\mathbb{A}\text{gt}, Q, \Pi, \pi, Act, d, o, Res, t)$ where

- $\mathbb{A}\text{gt} = \{1, \ldots, k\}$ is a non-empty set of agents;
- Q is a non-empty set of states;
- $\pi : Q \to \wp(\Pi)$ is a valuation of propositions;
- Act is a finite non-empty set of actions;
- $d : \mathbb{A}\text{gt} \times Q \to \wp(Act)\setminus\{\emptyset\}$ indicates the actions available to agent $a \in \mathbb{A}\text{gt}$ in state $q \in Q$;
- o maps each state $q \in Q$ and action profile $\alpha = (\sigma_1, \ldots, \sigma_k)$ such that $\sigma_a \in d(a, q)$ for each $a \in \{1, \ldots, k\}$, to another state $q' = o(q, \alpha)$;
- $t : Act \times Res \to \mathbb{Z}$ models the resources consumed and produced by actions; if $t(\sigma, r)$ is positive resource r is produced by σ, if $t(\sigma, r)$ is negative resource r is consumed by σ.

An **RBM** *with idle actions*, **iRBM** for short, is an **RBM** $\mathfrak{M}$ such that for all agents a and states q in $\mathfrak{M}$, there is an action $\sigma \in d(a, q)$ with $t(\sigma, r) = 0$ for all resource types r. We refer to this action (or to one of them if there is more than one) as the *idle action* of a and denote it by *idle*.

We will write $d_a(q)$ instead of $d(a, q)$, and use $d(q)$ to denote the set $d_1(q) \times \ldots \times d_k(q)$ of action profiles in state q. Similarly, $d_A(q)$ denotes the action tuples available to $A \subseteq \mathbb{A}\text{gt}$ in q. For $\alpha = (\sigma_1, \ldots, \sigma_k)$, we use α_A to denote the sub-tuple consisting of the actions of agents $A \subseteq \mathbb{A}\text{gt}$; moreover, we write α_a to refer to σ_a for $a \in \mathbb{A}\text{gt}$. Act^A is a set of tuples of actions by agents in A. We define $\mathsf{prod}(\sigma, r) := \max\{0, t(\sigma, r)\}$ (resp. $\mathsf{cons}(\sigma, r) := |\min\{0, t(\sigma, r)\}|$) as the amount of resource r produced (resp. consumed) by action σ. Note that $\mathsf{cons}(\sigma, r)$ is a non-negative number, and for any action σ and a resource type r it is not possible that both $\mathsf{prod}(\sigma, r)$ and $\mathsf{cons}(\sigma, r)$ are greater than 0.

In what follows, Q^{ω} denotes the set of all infinite sequences of elements from Q, and Q^{+} denotes the set of all finite sequences. A *path* $\lambda \in Q^{\omega}$ is an infinite sequence of states such that there is a transition between two adjacent states. A *finite path* is a finite segment of a path. We define $\lambda[i]$ to be the $(i+1)$-th state of λ, and $\lambda[i, \infty]$ to be the suffix $\lambda[i]\lambda[i+1]\ldots$. We denote a finite sequence λ extended by q by λq. A *resource-extended* path $\lambda \in (Q \times \mathsf{En})^{\omega}$ is an infinite sequence over $Q \times \mathsf{En}$ such that the restriction to states (the first component), denoted by $\lambda|_Q$, is a path in the underlying model. The projection of λ to the second component of each element in the sequence is denoted by $\lambda|_{\mathsf{En}}$. We call any initial (finite) suffix of a resource-extended path a *finite* resource extended path.

A *strategy* for a coalition $A \subseteq \mathbb{A}\text{gt}$ is a function $s_A : Q^{+} \to Act^A$ such that $s_A(\lambda q) \in d_A(q)$ for $\lambda q \in Q^{+}$. Such a strategy gives rise to a set of (resource-extended) paths. A (η, s_A, B)-*path* is a resource-extended path λ where for all $i = 0, 1, \ldots$ with $\lambda[i] := (q_i, \eta^i)$ there is an action profile $\alpha \in d(\lambda|_Q[i])$ such that:

1. $\eta^0 = \eta$ (η describes the initial resource distribution);
2. $s_A(\lambda|_Q[0, i]) = \alpha_A$ (A follow their strategy);
3. $\lambda|_Q[i+1] = o(\lambda|_Q[i], \alpha)$ (transition according to α);
4. for all $a \in A \cup B$ and $r \in Res$: $\eta^i_a(r) \geq \mathsf{cons}(\alpha_a, r)$ (each agent has enough resources to perform its action);
5. for all $a \in A \cup B$ and $r \in Res$: $\eta^{i+1}_a(r) = \eta^i_a(r) + t(\alpha_a, r)$ (resources are updated);
6. for all $a \in \mathbb{A}\text{gt} \setminus (A \cup B)$ and $r \in Res$: $\eta^{i+1}_a(r) = \eta^i_a(r)$ (the resources of agents not in $A \cup B$ do not change).

The (η, B)-*outcome* of a strategy s_A in q, $out(q, \eta, s_A, B)$ is defined as the set of all (η, s_A, B)-paths starting in q. Truth is defined over an **RBM** $\mathfrak{M}$, a state $q \in Q_{\mathfrak{M}}$, and an endowment η.

The semantics is given by the satisfaction relation $\models$ where the cases for propositions, negation and conjunction are standard and omitted:

$\mathfrak{M}, q, \eta \models \langle\langle A\rangle\rangle_B^{\downarrow}\varphi$ iff there is a strategy s_A for A such that for all $\lambda \in out(q, \eta, s_A, B)$, $\mathfrak{M}, \lambda, \eta \models \varphi$
$\mathfrak{M}, q, \eta \models \langle\langle A\rangle\rangle_B^{\zeta}\varphi$ iff there is a strategy s_A for A such that for all $\lambda \in out(q, \zeta, s_A, B)$, $\mathfrak{M}, \lambda, \zeta \models \varphi$
$\mathfrak{M}, \lambda, \eta \models \mathbf{X}\varphi$ iff $\mathfrak{M}, \lambda|_Q[1], \lambda|_{\mathsf{En}}[1] \models \varphi$
$\mathfrak{M}, \lambda, \eta \models \varphi\mathbf{U}\psi$ iff there exists i with $i \geq 0$ and $\mathfrak{M}, \lambda|_Q[i], \lambda|_{\mathsf{En}}[i] \models \psi$ and for all j with $0 \leq j < i$, $\mathfrak{M}, \lambda|_Q[j], \lambda|_{\mathsf{En}}[j] \models \varphi$
$\mathfrak{M}, \lambda, \eta \models \mathbf{G}\varphi$ iff for all $i \geq 0$, $\mathfrak{M}, \lambda|_Q[i], \lambda|_{\mathsf{En}}[i] \models \varphi$

The *model checking problem* for RAL is stated as follows: does $\mathfrak{M}, q, \eta \models \varphi$ hold? When the context is clear, we simply write $q, \eta \models \varphi$; if φ is only a propositional formula, we sometimes also omit η.

Observe that the standard ATL modalities $\langle\langle A\rangle\rangle$ can be defined as $\langle\langle A\rangle\rangle_{\mathbb{A}\mathrm{gt}}^{\infty}$, so the logic is a proper extension of ATL.

Remark 1 *(Infinitary and finitary semantics).* We refer to the semantics introduced above as *infinitary semantics*. In [2] the main semantics also allows for finite (maximal) paths. We refer to that semantics as *finitary semantics*. We note that both semantics coincide over **iRBM**s, as it is always possible to extend a path using idle actions.

3.3. The syntactic fragments rfRAL, prRAL and rfprRAL

Following [2] we define three fragments of RAL. The *resource-flat fragment*, rfRAL, only allows cooperation modalities of type $\langle\langle A\rangle\rangle_B^{\eta}$: agents are always (re-)equipped with a fresh set of resources whenever they re-consider their strategies. The *proponent-restricted fragment*, prRAL, only allows cooperation modalities of types $\langle\langle A\rangle\rangle^{\downarrow}$ and $\langle\langle A\rangle\rangle^{\eta}$: only the proponents are resource bounded. The fragment combining both restrictions (resource-flat and proponent-restricted) is denoted by rfprRAL.

3.4. Running example

We introduce a simple running example to illustrate the syntax and semantics of RAL and its fragments. The example represents interactions between two agents: a robot (agent 1) and its environment (agent 2). We consider only one resource type, energy. The robot needs to move into a position where it is capable of sending information to the base regularly and is also able to charge its battery. Both moving and the communication action require energy, and the charging action produces energy. We denote 'being in a suitable position to send information to the base' by p. The environment can make moving more or less difficult for the agent. We model this by giving the environment an 'obstruct' action which has the effect of requiring the agent to execute two move actions instead of one (and hence to spend more energy) in order to get into position; for the sake of the example, obstructing requires energy. The initial state is q_0 where the agent can move and the environment can obstruct, and both also can do nothing. If the agent moves and the environment idles, then the system reaches a state q_1 where p holds and the agent can loop forever between q_1 and q_3 (which also satisfies p) sending data and charging. If the agent does nothing upon reaching the position, it returns to the initial state (under the influence of gravity, for example). If in q_0 the environment obstructs the agent, the systems reaches a state q_2 where the agent can execute the move action again to reach q_1. To keep the example simple, we assume that in all states apart from q_0 the environment can only idle.

Formally, we have a resource-bounded model $\mathfrak{M} = (\mathbb{A}\mathrm{gt}, Q, \Pi, \pi, Act, d, o, Res, t)$ where

- $\mathbb{A}\mathrm{gt} = \{1, 2\}$
- $Q = \{q_0, q_1, q_2, q_3\}$
- $\Pi = \{p\}$
- $\pi(q_0) = \pi(q_2) = \emptyset$, $\pi(q_1) = \pi(q_3) = \{p\}$
- $Act = \{idle, move, send, charge, obstruct\}$
- Actions available to the robot: $d(1, q_0) = \{idle, move\}$, $d(1, q_1) = \{idle, send\}$, $d(1, q_2) = \{idle, move\}$, $d(1, q_3) = \{idle, charge\}$. Actions available to the environment: $d(2, q_0) = \{idle, obstruct\}$, $d(2, q_1) = d(2, q_2) = d(2, q_3) = \{idle\}$.
- The transition function is as follows:

Transitions from q_0	Transitions from q_1	Transitions from q_2	Transitions from q_3
$o(q_0, (idle, idle)) = q_0$	$o(q_1, (idle, idle)) = q_0$	$o(q_2, (idle, idle)) = q_2$	$o(q_3, (idle, idle)) = q_0$
$o(q_0, (move, idle)) = q_1$	$o(q_1, (send, idle)) = q_3$	$o(q_2, (move, idle)) = q_1$	$o(q_3, (charge, idle)) = q_1$
$o(q_0, (move, obstruct)) = q_2$			
$o(q_0, (idle, obstruct)) = q_0$			

- $Res = \{energy\}$
- $t(idle, energy) = 0$, $t(move, energy) = -2$, $t(send, energy) = -1$, $t(charge, energy) = 1$, $t(obstruct, energy) = -1$.

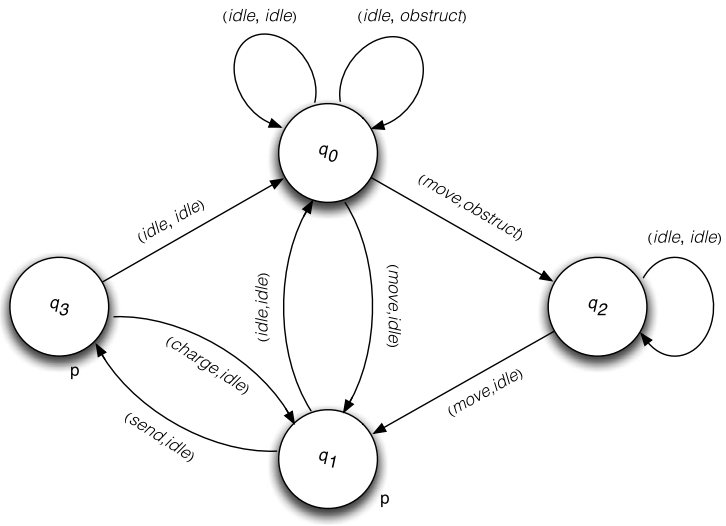

Fig. 1. State transition system.

The model is shown in Fig. 1.

Here are some example RAL properties which hold in the model:

- If both agents are resource-bounded, and the initial allocation of resources is 3 units of energy for the robot and 0 units for the environment, then the robot has a strategy to reach a state from where with the remaining resources it can maintain the invariant p. We represent an endowment η that assigns 3 units of energy to agent 1 and 0 units to agent 2 by $1{:}3, 2{:}0$:

  $$\langle\langle 1\rangle\rangle^{1:3,2:0}_{\{1,2\}} \top \mathbf{U} \langle\langle 1\rangle\rangle^{\downarrow}_{\{1,2\}} \mathbf{G} p$$

  In fact, with environment unable to obstruct, the robot is guaranteed to reach q_1 in one step:

  $$\langle\langle 1\rangle\rangle^{1:3,2:0}_{\{1,2\}} \mathbf{X} \langle\langle 1\rangle\rangle^{\downarrow}_{\{1,2\}} \mathbf{G} p$$

  This property belongs to full RAL: it is not in rfRAL since it uses $\downarrow$ in $\langle\langle 1\rangle\rangle^{\downarrow}_{\{1,2\}}\mathbf{G}$, nor in prRAL since it restricts the resources of the opponent agent 2.
- When both agents are resource bounded, and the environment is restricted to 0 units of energy, then with 2 units of energy, the agent can reach the state where it can maintain the invariant with 1 unit of energy:

  $$\langle\langle 1\rangle\rangle^{1:2,2:0}_{\{1,2\}} \top \mathbf{U} \langle\langle 1\rangle\rangle^{1:1,2:0}_{\{1,2\}} \mathbf{G} p$$

  This property belongs to the resource-flat fragment, since the second strategy for the invariant uses a fresh resource allocation.
- If only the robot is resource-bounded, and the initial allocation of resources is 5 units of energy for the robot and 0 units for the environment, then the robot has a strategy to reach a state from where with the remaining resources it can maintain the invariant p. The strategy is to execute the move action until the state q_1 is reached; in the worst case this would require 4 units of energy (since the environment is not resource-bounded, its initial allocation does not matter and it can perform the obstruct action). Then with at least one unit of energy remaining, the agent can enter the loop between q_1 and q_3:

  $$\langle\langle 1\rangle\rangle^{1:5,2:0}_{\{1\}} \top \mathbf{U} \langle\langle 1\rangle\rangle^{\downarrow}_{\{1\}} \mathbf{G} p$$

  This property does not belong to rfRAL but it does belong to prRAL. It can be written without the argument for the set of resource-bounded agents:

  $$\langle\langle 1\rangle\rangle^{1:5,2:0} \top \mathbf{U} \langle\langle 1\rangle\rangle^{\downarrow} \mathbf{G} p$$

  In fact, this property belongs to the fragment with a decidable model checking problem, pprRAL (positive fragment of prRAL).
- If only the robot is resource-bounded, then with initial allocation of 4 units of energy, it can reach a state where with one unit of energy it can maintain the invariant:

$$\langle\langle 1\rangle\rangle^{1:4,2:0}\top\mathbf{U}\langle\langle 1\rangle\rangle^{1:1,2:0}\mathbf{G}p$$

This property belongs to rfprRAL.

3.5. Similarities and differences

We conclude this section with a discussion of similarities and differences between the variant of RAL presented here and the original resource agent logics of [2] and the logic of [6]. In the interests of readability, we refer to the setting of [2] by S_1, and to that of [6] by S_2.

The language of RAL as given above is almost identical to the setting of S_1, except that we do not allow the release operator.[3] Setting S_2 essentially corresponds to the resource-flat and proponent-restricted fragment of RAL. **RBM**s serve as models of S_1, where S_2 uses **iRBM**s. There are also differences in how the production and consumption of resources are handled. In S_2 the resources of a coalition of agents are combined before the resource requirements of actions are evaluated. A shortage of resources of one agent can thus be balanced by surplus resources of another agent in the coalition. The implicit assumption is that agents in the proponent coalition share their resources. It is not necessary to decide how to divide any resources produced, as the coalition sticks together throughout the relevant part of the evaluation of the current formula. When a new cooperation modality is encountered, all agents are re-equipped with a new endowment. This is a property of the resource-flat and proponent restricted fragment of the logic. This approach cannot be used if the restriction of resource-flatness or proponent restrictiveness is dropped. First, a coalition may split-up in a nested modality in which agents are not re-equipped with new resources. In this case it is important to know how many resources each individual agent has. A similar difficulty arises if an agent in the proponent coalition becomes an opponent in a nested cooperation modality. If the logic is not proponent restricted it is necessary to know how many resources this agent possesses. In S_1 this issue is addressed by introducing *shares*. A share models how many resources an individual agents contributes to the pool of resources needed to execute the joint action, and also the amount of resources each agent receives when resources are produced. This can be seen as a binding agreement about the resource distribution. Again, the underlying assumption is that agents in the proponent and opponent coalitions share their resources within the coalition.

As we consider a non-resource-flat variant of RAL here, the approach of S_2 is not sufficient, whereas the approach of S_1 complicates the presentation. We therefore adopt a less involved formalisation for ease of readability: resources *cannot* be shared within a coalition and each agent is entirely responsible for its own resource balance. Thus, at each moment agents have a clearly defined resource endowment. Finally, most results of S_1 are given in terms of the finitary semantics whereas we require that paths are always infinite (cf. Remark 1).

4. The quest for decidability

If unbounded production of resources is allowed, the model checking problem for many resource logics is undecidable. In particular, most fragments of the resource agent logic considered in [2,5] are undecidable. The case of the resource-flat, proponent-restricted fragment remained open in [2], but was shown to be decidable in [6,8] (see also [9]):

Observation 1. rfprRAL *is decidable over* **iRBM***s.*

A natural question arises: *can we extend decidability to more expressive fragments*? Which restrictions are essential for decidability, and which can be relaxed?

The result above relies on three restrictions on RAL: (1) the availability of an idle action; (2) resource flatness, that is, each nested quantifier has a fresh endowment; and (3) proponent restriction, that is, there are no resource bounds on the opponents. It turns out that all three restrictions are essential for the decidability of rfprRAL, as we explain below.

It follows from [2] that the availability of an idle action is essential for the decidability of rfprRAL:

Observation 2. rfprRAL *is undecidable over* **RBM***s.*

However, the availability of an idle action on its own is not sufficient for decidability. Replacing **RBM**s with **iRBM**s does not always make the model checking problem decidable.

In this section we present the main idea underlying the undecidability proofs of model checking RAL from [2] and investigate the reasons for the (un)decidability. We show that the model checking problem for RAL, i.e., the logic without any additional restrictions, remains undecidable over **iRBM**s (see Theorem 1 below). This result also holds for both the proponent-restricted fragment and the resource-flat fragment. For these fragments we also investigate the effect of the number of agents and resource types on the undecidability. The results of [2,7] depend on the availability of two resource

[3] In S_1 the release operator is used to show the undecidability of model checking resource-bounded agents using memoryless strategies. As we focus on perfect-recall strategies in this paper, we do not need the extra expressivity provided by the release operator.

types. Here we show that undecidability holds even if each agent has only a *single* resource type available. In this case, however, additional agents are required for undecidability; more precisely, one or two additional agents are required depending on the setting. We also show that in the case of prRAL over **iRBM**s, although the model checking problem remains undecidable, the formula expressing an undecidable problem in the logic is more complex than the formula required for rfRAL. This suggests the idea of a syntactic restriction of prRAL which does not allow expression of the undecidable property. We end this section by motivating a new fragment of RAL, the positive proponent restricted fragment, pprRAL. This fragment is more expressive than that introduced in [7], in that the formula φ_1 on the left-hand-side of $\varphi_1 \mathbf{U} \varphi_2$ is not constrained to be purely propositional. In Section 5 we show that the model checking problem for pprRAL is decidable over **iRBM**s.

For all the results below, the undecidability of the model checking problem is shown by a reduction of the halting problem for two-counter machines (also called Minsky machines, see [33] for details). A *two-counter machine* (TCM) is essentially a pushdown automaton with two stacks. The stacks are represented as two counters over natural numbers. Each of the counters (1 and 2) can be incremented, decremented (if non-zero), and tested for zero. In [33] it is shown that these machines are expressively equivalent to Turing machines. As a consequence the halting problem of two-counter machines is undecidable as well. For this paper we only need to consider TCMs with empty inputs; therefore, we only introduce this special type of TCMs.

Definition 2 *(Empty-band two-counter machine (cf. [33]), empty-band).* An empty band TCM $\mathcal{A}$ is given by $(S, s_{\text{init}}, S_f, \Delta)$ where S is a finite set of states, $s_{\text{init}} \in S$ is the initial state, $S_f \subseteq S$ is a set of final states, and $\Delta \subseteq (S \times \{0,1\}^2) \times (S \times \{-1,0,1\}^2)$ is the transition relation such that if $((s, E_1, E_2), (s', C_1, C_2)) \in \Delta$ and $E_i = 0$ then $C_i \neq -1$ for $i = 1, 2$ (to ensure that an empty counter is not decremented). In the following we sometimes use infix notation and write $(s, E_1, E_2)\Delta(s', C_1, C_2)$ instead of $((s, E_1, E_2), (s', C_1, C_2)) \in \Delta$. We call $((s, E_1, E_2), (s', C_1, C_2))$ a *transition* if $(s, E_1, E_2)\Delta(s', C_1, C_2)$ and denote a typical transition by τ.

As we focus on empty-band TCMs, we often simply say *automaton* or *machine* to refer to such a TCM. A TCM can be considered as a transition system equipped with two counters that influence the transitions. Each transition step of the automaton depends on whether the counters are zero or non-zero, and in each step the counters can be incremented or decremented. It is important to emphasise that a TCM cannot access the specific value of the counters. In the following let $\tau = ((s, E_1, E_2), (s', C_1, C_2))$ be a transition. Here, $E_i = 1$ (resp. $= 0$) represents that counter i is non-zero (resp. zero), and $C_k = 1$ (resp. $= -1$) denotes that counter i is incremented (resp. decremented) by 1. A value $C_k = 0$ indicates that counter k is left unchanged. The transition encodes that in state s the automaton can change its state to s' provided that the first (resp. second) counter meets condition E_1 (resp. E_2). The value of counter k changes according to C_k for $k = 1, 2$. For example, the transition $((s, 1, 0), (s', -1, 1))$ is enabled if the current state is s, counter 1 is non-zero, and counter 2 is zero. If the transition is enabled and taken, the state changes to s', counter 1 is decremented and counter 2 is incremented by 1.

The general mode of operation is as for pushdown automata. In particular, a *configuration* is a triple $(s, v_1, v_2) \in S \times \mathbb{N}_0^2$ describing the current state (s), the value of counter 1 (v_1) and of counter 2 (v_2). An $\mathcal{A}$*-computation* ρ (or simply *computation* if the two-counter machine is clear from context) is a sequence of subsequent configurations resulting from transitions according to Δ, such that the first state is s_{init}. An *accepting* computation is a finite computation $\rho = (s^i, v_1^i, v_2^i)_{i=1,\ldots,l}$ where the last state $s^l \in S_f$ is a final state. We use $\rho_i = ((s_i, E_1^i, E_2^i), (s_{i+1}, C_1^i, C_2^i))$ to denote the transition that leads from the ith configuration (s_i, v_1^i, v_2^i) to the $(i+1)$th configuration $(s_{i+1}, v_1^{i+1}, v_2^{i+1})$ for $i < l$. Note that we have that $v_k^{i+1} = v_k^i + C_k^i$ for $k = 1, 2$.

Finally, we say that a transition $\tau = ((s, E_1, E_2), (s', C_1, C_2))$ is *enabled* in a configuration (s, v_1, v_2) if the value v_k of counter k satisfies condition $E_k \in \{0, 1\}$ for $k \in \{1, 2\}$ (with the obvious meaning of being zero or non-zero), i.e. $v_k > 0$ iff $E_k = 1$. If an enabled transition τ is taken the automaton changes its control state from s to s', and counter i is updated by adding $C_i \in \{-1, 0, +1\}$. The automaton halts on empty input iff there is an *accepting computation*.[4]

4.1. Undecidability of rfRAL and prRAL with two resources types over iRBMs

In this section we essentially extend the undecidability results of [2] to **iRBM**s. We first give a generic construction of an **iRBM** $\mathfrak{M}_1^{\mathcal{A}}$ for a two-counter machine $\mathcal{A}$ which is used to show that model checking rfRAL and prRAL are undecidable over **iRBM**s (Theorems 1 and 2). The key is provided by the Simulation Lemma 1.

4.1.1. Encoding of two-counter machines

In [2] it was shown that model checking formulae of the form $\langle\langle 1 \rangle\rangle^{\vec{0}}_{\mathbb{A}\text{gt}} \mathbf{F}\,\text{halt}$ is undecidable over **RBM**s, where halt is an arbitrary proposition encoding that the TCM halts. In the following we show that this result carries over to **iRBM**s. Before we give the formal definitions and proof we present the basic idea underlying the reductions of [2] regarding **RBM**s.[5] The key

[4] We only require that there is an accepting computation; in particular, there could be other (infinite) non-accepting computations of the automaton due to non-determinism.

[5] No complete, formal proofs are given in [2], only proof sketches.

idea is to encode the transition table of the automaton as an **RBM**, where the two counters are simulated by two resource types R_1 and R_2. We give a reduction for both one and two agents. First, we describe the variant with two agents. In this variant, agent 1 is the *simulator* and agent 2 is the *spoiler*. Essentially, the role of agent 1 is to select transitions τ of the automaton, while the role of agent 2 is to ensure that only enabled transitions are selected by agent 1. As an illustration, let us consider a single transition $\tau = ((s, E_1, E_2), (s', C_1, C_2))$. The model has different types of states, including *automaton states* S. In states $s \in S$ the simulator agent 1 can execute an action E_1E_2 followed by an action $s'^{C_1C_2}_{E_1E_2}$. Both actions together simulate the selection of τ. The first action, E_1E_2, is used to select and to (partially) check whether a transition of the TCM is enabled. That is, if $E_i = 1$, agent 1 must have a resource of type R_i to execute the action. After executing E_1E_2 the model enters a *test state* sE_1E_2. The purpose of the test state is to check whether a transition with $E_i = 0$ was selected by agent 1 only if counter i was indeed zero. That is, the test state ensures that the simulation is sound. Note that, in general, nothing prevents agent 1 from executing such an action if it has resources available. The problem is that it is not possible to *test for zero* directly in the model.[6] The workaround proposed in [2] is to use the spoiler agent 2 to encode the "zero test". In a test state sE_1E_2, agent 2 must not be able to reach the *fail state* q_f. Reaching the fail state is only possible if resources are available in cases where there should not be any. This is encoded in the model by test actions $test_i$ with $i \in \{1, 2\}$. For example, if counter 1 should be empty, $E_1 = 0$, the action $test_1$ can only be executed if resources of type 1 are available. That is, the executability of the action indicates a flawed simulation. To work correctly, this requires that agent 2 correctly mirrors agent 1's resource balance, i.e. agent 2 also simulates the counter values. This is achieved by essentially making the model turn-based, in the sense that agent 2 frequently has no alternatives: once agent 1 has executed an action $s'^{C_1C_2}_{E_1E_2}$ to update the counter values, an *intermediate* state $s'\underline{C_1C_2}$ is introduced in which agent 2 has a single choice with the same effect on its resource balance as agent 1's previous action.[7] Based on this idea, it is shown in [2] (using the finitary semantics) that the TCM $\mathcal{A}$ halts on the empty input if, and only if, the formula $\langle\langle 1\rangle\rangle^{\bar{0}}_{\{1,2\}}\mathbf{F}\,\mathsf{halt}$ holds in the corresponding model. The state corresponding to the automaton's accepting state is labelled halt.

In extending the reduction to **iRBM**s, the main difficulty is correctly mirroring agent 1's resources by agent 2 in the presence of idle actions. It is no longer possible to give agent 2 only a single action to execute; an action with no costs must also be available. We extend the construction outlined above accordingly. The key idea is that executing the idle action does not help agent 2 spoil the execution. The next definition formalises an appropriate encoding to work over **iRBM**s.

Definition 3 ($\mathfrak{M}_1^{\mathcal{A}}$). Let $\mathcal{A} = (S, s_{\text{init}}, S_f, \Delta)$ be an empty-band TCM. From $\mathcal{A}$ we construct the **iRBM** $\mathfrak{M}_1^{\mathcal{A}} = (\{1, 2\}, Q, \Pi, \pi, Act, d, o, \{R_1, R_2\}, t)$ with

1. $Q = S \cup Q_1 \cup Q_2 \cup \{q_f, q_h, q_{\circlearrowleft}\}$ where $Q_1 = \{sE_1E_2 \mid s \in S, E_1, E_2 \in \{0, 1\}\}$ and $Q_2 = \{s\underline{C_1C_2} \mid s \in S, C_1, C_2 \in \{-1, 0, 1\}\}$. State q_f (resp. q_h and $q_{\circlearrowleft}$) is called a *fail state* (resp. *auxiliary halting state* and *loop state*).
2. The set Act of actions is defined as follows. For each transition $((s, E_1, E_2), (s', C_1, C_2))$ of $\mathcal{A}$ the set contains actions E_1E_2, $s'^{C_1C_2}_{E_1E_2}$, and $s'C_1C_2$. Additionally, there are the idle action idle and test actions $test_i$ for $i \in \{1, 2\}$.
3. The action availability is defined according to Δ. For agent 1 we have:

$$E_1E_2 \in d_1(s) \text{ iff there is a transition } ((s, E_1, E_2), (s', C_1, C_2)) \in \Delta$$
$$idle \in d_1(q) \text{ for all } q \in Q$$
$$s'^{C_1C_2}_{E_1E_2} \in d_1(sE_1E_2) \text{ iff } ((s, E_1, E_2), (s', C_1, C_2)) \in \Delta$$

   and for agent 2:

$$idle \in d_2(q) \text{ for all } q \in Q$$
$$s'C_1C_2 \in d_2(s'\underline{C_1C_2}) \text{ for all } s'\underline{C_1C_2} \in Q_2$$
$$test_i \in d_2(sE_1E_2) \text{ iff } E_i = 0 \text{ with } i \in \{1, 2\}$$

4. The set of propositions is defined by $\Pi = \{\mathsf{halt}, \mathsf{fail}\}$. All states in $\{q_h\} \cup S_f$ are labelled with halt and q_f is labelled with fail.
5. The transition function is defined as follows:

$$o(s, (E_1E_2, idle)) = sE_1E_2$$
$$o(s, (idle, idle)) = o(sE_1E_2, (idle, idle)) = q_{\circlearrowleft}$$
$$o(q_{\circlearrowleft}, (idle, idle)) = q_{\circlearrowleft}$$

[6] Testing for zero is a delicate property, the satisfaction of which seems crucial for the undecidability of other formalisms, such as Petri nets [34].

[7] We note that we need the underscore in order to make the two types of states syntactically different. Otherwise, in case $C_1 = E_1$, $C_2 = E_2$ and $s = s'$, we could not have two 'copies' of a state.

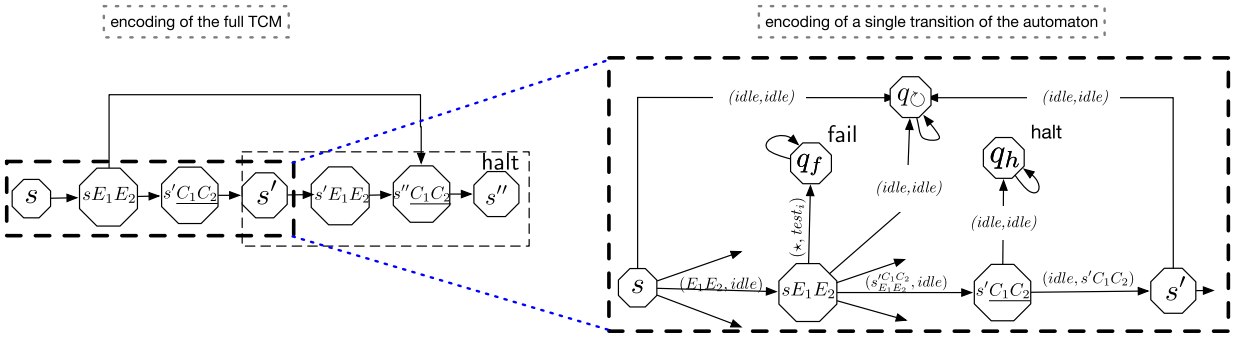

Fig. 2. The left figure schematically shows an abstraction of the encoding of the automaton $\mathfrak{M}_1^{\mathcal{A}}$ for $\Delta = \{((s, E_1, E_2), (s', C_1, C_2)), ((s, E_1, E_2), (s'', C_1, C_2)), ((s', E_1, E_2), (s'', C_1, C_2))\}$. The right figure shows an excerpt of the encoding of a transition $(s, E_1, E_2)\Delta(s', C_1, C_2)$ using two resource types and two agents.

$$o(sE_1E_2, (s'^{C_1C_2}_{E_1E_2}, idle)) = s'\underline{C_1C_2}$$
$$o(sE_1E_2, (\star, test_i)) = o(q_f, (idle, idle)) = q_f$$
$$o(s'\underline{C_1C_2}, (idle, s'C_1C_2)) = s'$$
$$o(s'\underline{C_1C_2}, (idle, idle)) = o(q_h, (idle, idle)) = q_h$$

where $\star$ represents any action available to the respective agent in that state.

6. The actions' resource consumption/production is defined by function t where $i, r \in \{1, 2\}$:

$$t(E_1E_2, R_r) = -E_r$$
$$t(idle, R_r) = 0$$
$$t(s^{C_1C_2}_{E_1E_2}, R_r) = C_r + E_r$$
$$t(sC_1C_2, R_r) = C_r$$
$$t(test_i, R_r) = \begin{cases} -1 & \text{if } i = r \\ 0 & \text{else} \end{cases}$$

Let us consider a TCM $\mathcal{A} = (S, s_{\text{init}}, S_f, \Delta)$. The construction of the model $\mathfrak{M}_1^{\mathcal{A}}$ is sketched in Fig. 2 (left), and the encoding of a single transition $\tau = ((s, E_1, E_2), (s', C_1, C_2))$ is illustrated in Fig. 2 (right). As explained above, the action E_1E_2 consumes $-E_r$ resources of R_r for $r = 1, 2$. This simulates that only enabled transitions τ can be taken. If $E_r = 1$ then the action E_1E_2 can only be taken if resources $R_r \geq 1$. Actions of type $s^{C_1C_2}_{E_1E2}$ consume/produce $C_r + E_r$ units of resource R_r, $r = 1, 2$. The component C_r simulates the decrement and increment of counter r where E_r corrects the possible (temporary) subtraction from the previous action E_1E_2. The necessary information to select the correct action is stored in the state sE_1E_2. Clearly, actions can only be performed if sufficient resources are available. The difficulty is to ensure that actions E_1E_2 with some $E_r = 0$ are only performed if the counter r is actually 0; that is, if *no* resources of type R_r are available. For this purpose, *test actions* $test_r$ that cost -1 units of resource R_r for $r \in \{1, 2\}$, are introduced. Such an action $test_r$ can only be performed in states sE_1E_2 if $E_r = 0$, and it always leads to the fail state q_f. Now, in a state sE_1E_2 with some element equal to 0, say $E_1 = 0$, $E_2 = 1$, (representing that counter 1 should be zero and 2 should be non-zero) the action $test_1$ can be used to verify whether the currently available resources model the counter correctly: if q_f is reachable, resources of type R_1 are available, although this should not be the case according to E_1.

4.1.2. Properties of the encoding: the simulation lemma

In the following, we state properties of the encoding, and prove a simulation lemma which relates runs of the two-counter machine with paths in the model. First, we make a straightforward observation:

Observation 3. *The model $\mathfrak{M}_1^{\mathcal{A}}$ is an* **iRBM**.

We define the concept of a *computation pre-encoding*. This is a finite path in the model which will later be shown to encode a partial computation of the automaton.

Definition 4 *(Computation pre-encoding of $\mathfrak{M}_1^{\mathcal{A}}$).* Let $\mathcal{A}$ be an empty-band TCM. A finite resource-extended path $\lambda \in (Q \times \mathsf{En})^+$ in $\mathfrak{M}_1^{\mathcal{A}}$ is called an *$\mathcal{A}$-computation pre-encoding of $\mathfrak{M}_1^{\mathcal{A}}$* if it satisfies the following properties:

1. $\eta(1, R_r) = \eta(2, R_r)$ where $\lambda[0] = (q, \eta)$ for $r \in \{1, 2\}$; and
2. $\lambda|_Q = (s^i(s^i E_1^i E_2^i)(s^{i+1}\underline{C_1^i C_2^i}))_{i=1,\ldots,k} s^{k+1}$ or $\lambda|_Q = s^1$.

An $\mathcal{A}$-computation pre-encoding of $\mathfrak{M}_1^{\mathcal{A}}$ is called *accepting* if its final state s^{k+1} is an accepting state of the TCM, $s^{k+1} \in S_f$, or if $\lambda|_Q = s^1 \in S_f$. In the following we shall often omit "of $\mathfrak{M}_1^{\mathcal{A}}$".

The first requirement states that the endowments of both agents must be the same in the initial state. The second requirement expresses that a fail, auxiliary halting, or loop state must never be visited, and the path ends in a state that is also a state of $\mathcal{A}$; moreover, it specifies the order in which states in the model are visited. The latter is inherent in the construction of the model. The next proposition states that, on a computation pre-encoding, agent 2 correctly mirrors the resources of agent 1 whenever a state in the model, representing a state of the TCM, is visited.

Proposition 1. *Let $\mathcal{A}$ be an empty-band TCM and $\lambda = (q^i, \eta^i)_{i=1,\ldots,3(k-1)+1}$ be an $\mathcal{A}$-computation pre-encoding.*

(a) We have that $\eta^{3(k-1)+1}(1, R_r) = \eta^{3(k-1)+1}(2, R_r)$ for $r = 1, 2$.
(b) If $\lambda \circ (sE_1E_2, \eta)(s'\underline{C_1C_2}, \eta')$ is a partial resource-extended path, then $\lambda \circ (sE_1E_2, \eta)(s'\underline{C_1C_2}, \eta')(s', \eta'')$ is a partial resource-extended path with a uniquely defined endowment η''.
(c) If $\lambda[i] = (q^i, \eta^i) = (sE_1E_2, \eta^i)$ with $E_r = 0$, then ($\eta^i(1, R_r) = 0$ if, and only if, $\eta^i(2, R_r) = 0$), where $r \in \{1, 2\}$.

The proof is given in Appendix A.1. Part (a) expresses that the resource endowments of agents 1 and 2 are identical whenever a state q corresponding to an automaton state s is reached. Part (b) says that agent 2 always has enough resources in states of type $s'\underline{C_1C_2}$ to mirror the action executed by agent 1 one step before. Finally, in (c) the crucial observation is made that in the test states sE_1E_2 with $E_r = 0$ for $r \in \{1, 2\}$, both agents both have either no resources of R_r available, or they both have resources of R_r available. This ensures that the test actions are correctly executed.

The next definition relates a computation pre-encoding to the computations of the automaton it simulates. This means that essentially the same automaton states are visited, and the resources of agent 1 correctly simulate the counter values.

Definition 5 *(Simulation, $\mathcal{A}$-computation encoding).* We say that the $\mathcal{A}$-computation pre-encoding $\lambda = (q^i, \eta^i)_{i=1,\ldots,3(k-1)+1}$, $k \in \mathbb{N}$, *simulates the computation ρ* iff the following holds:

1. ρ has length k;
2. for every $i \in \{1, \ldots, k\}$ if $\rho[i] = (s, v_1, v_2)$ then $\lambda[3(i-1)+1] = (s, \eta)$ with $\eta(1, R_r) = v_r$ for $r \in \{1, 2\}$; and
3. for any configuration $\lambda[i] = (sE_1E_2, \eta)$ on λ with $E_r = 0$, $r \in \{1, 2\}$, it holds that $\eta(1, R_r) = 0$.

An $\mathcal{A}$-computation pre-encoding is called an *$\mathcal{A}$-computation encoding in $\mathfrak{M}_1^{\mathcal{A}}$* if it simulates some computation of $\mathcal{A}$. Consequently, an $\mathcal{A}$-computation encoding in $\mathfrak{M}_1^{\mathcal{A}}$ is called *accepting* if it simulates an accepting computation of $\mathcal{A}$.

The following lemma is the key step in our reduction. It specifies that the computation pre-encodings do exactly characterise the computations of the automaton. In other words, the behaviour of the automaton is exactly captured by the computation pre-encoding in the constructed **iRBM**. This lemma concludes the construction of $\mathfrak{M}_1^{\mathcal{A}}$ and the analysis of its structural properties.

Lemma 1 *(Simulation Lemma for $\mathfrak{M}_1^{\mathcal{A}}$). There is a bijection $f^{\mathcal{A}}$ between computations of $\mathcal{A}$ and $\mathcal{A}$-computation encodings of $\mathfrak{M}_1^{\mathcal{A}}$ such that $f^{\mathcal{A}}(\rho)$ simulates the computation ρ. In particular, if ρ is an accepting computation then $f^{\mathcal{A}}(\rho)$ is also accepting.*

The proof is given in Appendix A.1.

4.1.3. Resource-flat fragment with two resource types

We first show that proponent restrictedness is essential for decidability over **iRBM**s, by showing that resource flatness is not sufficient, and rfRAL is undecidable over **iRBM**s. In [2] it was shown that model checking formulae of type $\langle\langle 1\rangle\rangle^{\bar{0}}_{\mathrm{Agt}}\mathbf{F}\,\mathsf{halt}$ is undecidable over **RBM**s. The decidability of this fragment was open over **iRBM**s. In Theorem 1, we show that undecidability continues to hold. The proof adapts the approach of [2] to work over **iRBM**s.

As in [2] we show that the empty-band TCM $\mathcal{A}$ halts iff $\mathfrak{M}_1^{\mathcal{A}}, q_{\mathrm{init}}, \bar{0} \models \langle\langle 1\rangle\rangle^{\bar{0}}_{\{1,2\}}\mathbf{F}\,\mathsf{halt}$. By Lemma 1 this is equivalent to showing that there is an accepting $\mathcal{A}$-computation encoding iff $\mathfrak{M}_1^{\mathcal{A}}, q_{\mathrm{init}}, \bar{0} \models \langle\langle 1\rangle\rangle^{\bar{0}}_{\{1,2\}}\mathbf{F}\,\mathsf{halt}$.

Consider the encoding of $(s, E_1, E_2)\Delta(s', C_1, C_2)$ shown in Fig. 2. First, we observe that executing an idle action is not helpful for agent 1 in states s and sE_1E_2; neither is it helpful for agent 2 to idle in a state $s'\underline{C_1C_2}$. If agent 1 executes idle in states s or sE_1E_2, this would yield the state q_f or $q_{\circlearrowleft}$ which cannot help to make the formula true; on the contrary, if the formula is not already true, these states make it false. Similarly, if agent 2 executes idle in $s'C_1C_2$, the formula would

be true when state q_h is reached. As we are looking for a winning strategy for agent 1 against *all* strategies of agent 2, we can neglect the cases where either agent executes an idle action in the aforementioned states. As a result, we only need to consider paths that have the structure of $\mathcal{A}$-computation pre-encodings. The second agent is needed to ensure that agent 1 chooses actions that yield an $\mathcal{A}$-computation encoding, i.e. that the selection of actions simulates a possible behaviour of the automaton. By construction, agent 2 only has a choice in states sE_1E_2 and $s'C_1C_2$. In the former state the agent can execute a *test action* if sufficient resources are available. In the latter, it could idle—as discussed above, an action the agent should not execute. As a consequence, in states of type $s'C_1C_2$, agent 2 should essentially only perform the action $s'C_1C_2$ ensuring that the agent's resources mirror agent 1's resources (cf. Proposition 1). This essentially ensures condition 3 of Definition 5 (simulation). Formally, we have:

Lemma 2. *The empty-band TCM $\mathcal{A}$ halts iff* $\mathfrak{M}_1^{\mathcal{A}}, s_{init}, \bar{0} \models \langle\langle 1\rangle\rangle^{\bar{0}}_{\{1,2\}} \mathbf{F}\,\mathsf{halt}$.

We briefly sketch the main idea of the proof below; the full proof is given in Appendix A.1:

- Suppose that $\mathcal{A}$ halts. Then, agent 1 simulates the transitions of the machine's accepting run. Due to the simulation, agent 2 will never be able to enforce the fail state q_f. Moreover, either agent 2's resources correctly mirror agent 1's resources, or agent 2 performs the idle action. In both cases, either an accepting state or the auxiliary halting state q_h, both labelled halt, are reached. The formula is true.
- Let the formula be true. Agent 1 must have a strategy that guarantees reaching a state labelled halt against all strategies of agent 2, including agent 2's strategy in which agent 2 never performs the idle action. This strategy of agent 2 correctly mirrors agent 1's resources and ensures that agent 1's strategy only selects enabled transitions. Thus, the strategy of agent 1 yields a $\mathcal{A}$-computation encoding.

The previous lemma immediately yields the following theorem which concludes our study of rfRAL with two resource types.

Theorem 1. *Model checking* rfRAL *over the class of* **iRBM***s is undecidable, even for two agents and two resource types.*

4.1.4. The proponent restricted fragment with two resource types

Theorem 1 shows that the restriction of resource-flatness is not enough to obtain a decidable model checking property. In this section, we consider the proponent-restricted fragment. We show that proponent-restrictedness on its own is also not sufficient for decidability, and prRAL is undecidable over **iRBM**s. We do this by adapting the undecidability proof of [2] for prRAL to work over **iRBM**s. This is a negative result. However, in contrast to Theorem 1, the formula used in the reduction is more complex. This leaves room for restrictions on the temporal structure of prRAL. Indeed, this is the motivation for the decidable fragment of prRAL that we introduce in Section 4.3.

The proof of the undecidability result for prRAL over **RBM**s of [2] essentially follows an encoding similar to the one shown in Fig. 3. However, in contrast to the previous encoding, the second agent is removed, and agent 1 itself is used to perform the zero test. This requires a slightly more sophisticated formula. First, we show a reduction with respect to our original, two-player model $\mathfrak{M}_1^{\mathcal{A}}$: the automaton $\mathcal{A}$ halts if, and only if, $\mathfrak{M}_1^{\mathcal{A}}, q_{\text{init}}, \bar{0} \models \langle\langle 1,2\rangle\rangle^{\bar{0}}((\neg\langle\langle 1,2\rangle\rangle^{\downarrow}\mathbf{X}\,\mathsf{fail})\mathbf{U}\,\mathsf{halt})$. The main idea is that in test state sE_1E_2, agents $\{1,2\}$ must not be able to reach the *fail state* q_f, which is expressed by $\neg\langle\langle\{1,2\}\rangle\rangle^{\downarrow}\mathbf{X}\,\mathsf{fail}$. The next lemma follows essentially as a Corollary of [2], by adding idle loops to the construction. However, for uniformity, we base the proof on the model $\mathfrak{M}_1^{\mathcal{A}}$.

Lemma 3. *The empty-band TCM $\mathcal{A}$ halts iff* $\mathfrak{M}_1^{\mathcal{A}}, q_{init}, \bar{0} \models \langle\langle 1,2\rangle\rangle^{\bar{0}}((\neg\langle\langle 1,2\rangle\rangle^{\downarrow}\mathbf{X}\,\mathsf{fail})\mathbf{U}\,\mathsf{halt})$.

Proof. The proof is analogous to the one given for Lemma 2. For the direction "⇒" it is sufficient to observe that the only states from which the fail state q_f can be reached within one step are of the form sE_1E_2 with $E_r = 0$ for $r \in \{1,2\}$. Thus, the strategy profile (s_1, s_2), where s_1 is the strategy of agent 1 as defined in Lemma 2 and s_2 is an arbitrary strategy for agent 2, witnesses the truth of the formula. The other direction is done analogously to Lemma 2. □

In the formula used in the reduction of Lemma 3, $\langle\langle 1,2\rangle\rangle^{\bar{0}}((\neg\langle\langle 1,2\rangle\rangle^{\downarrow}\mathbf{X}\,\mathsf{fail})\mathbf{U}\,\mathsf{halt})$, the two agents 1 and 2 always act as a team; there is no opponent. Thus, the two agents can be merged into a single agent. The next result makes this observation precise.

Theorem 2. *Model checking* prRAL *over the class of* **iRBM***s is undecidable, even in the case of a single agent and two resource types.*

Proof. We modify the model $\mathfrak{M}_1^{\mathcal{A}} = (\{1,2\}, Q, \Pi, \pi, Act, d, o, \{R_1, R_2\}, t)$ to a model $\widehat{\mathfrak{M}}_1^{\mathcal{A}} = (\{1\}, Q \setminus \{q_h\}, \Pi, \widehat{\pi}, \widehat{Act}, \widehat{d}, \widehat{o}, \{R_1, R_2\}, \widehat{t})$. We remove the auxiliary halting state as it must not be reached by the proponent agent 1. Essentially, we merge the two agents into one. The encoding of a single automaton transition is shown in Fig. 3. We define $\widehat{d}_1(q)$ as $d_1(q)$

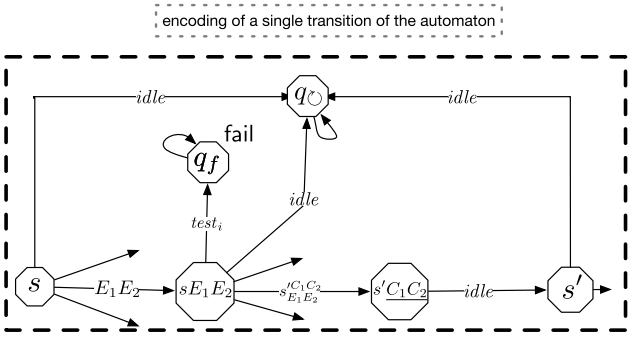

Fig. 3. Excerpt of model $\widehat{\mathfrak{M}}_1^{\mathcal{A}}$: encoding of a transition $(s, E_1, E_2)\Delta(s', C_1, C_2)$ using two resource types and one agent.

where we additionally require that $test_i \in \widehat{d}_1(sE_1E_2)$ iff $E_i = 0$; agent 1 can now make all decisions. The action set $\widehat{Act}$ equals Act but all actions of type sC_1C_2 are removed. The transition function $\widehat{o}$ is obtained from o:

$$\begin{aligned}
\widehat{o}(s, E_1E_2) &= sE_1E_2\\
\widehat{o}(s, idle) = \widehat{o}(sE_1E_2, idle) &= q_{\circlearrowleft}\\
\widehat{o}(q_{\circlearrowleft}, idle) &= q_{\circlearrowleft}\\
\widehat{o}(sE_1E_2, s'^{C_1C_2}_{E_1E_2}) &= s\underline{C_1C_2}\\
\widehat{o}(sE_1E_2, test_i) = \widehat{o}(q_f, idle) &= q_f\\
\widehat{o}(s'\underline{C_1C_2}, idle) &= s'
\end{aligned}$$

The cost function $\widehat{t}$ and the labelling function $\widehat{\pi}$ are defined as before restricted to the new action set.

Now, it is easy to see that each resource-extended path $\lambda = (q^i, \eta^i)_{i\in\mathbb{N}}$ in $\mathfrak{M}_1^{\mathcal{A}}$ that does not visit state q_h corresponds to a path $\lambda = (q^i, \widehat{\eta}^i)_{i\in\mathbb{N}}$ in $\widehat{\mathfrak{M}}_1^{\mathcal{A}}$ with $\widehat{\eta}^i(1, R_r) = \eta^i(1, R_r)$. By Proposition 1(c), the zero-test in the test states can equivalently be defined with respect to agent 1's resource endowment. It is immediate that $\mathfrak{M}_1^{\mathcal{A}}, q_{\mathbf{init}}, \bar{0} \models \langle\langle 1,2\rangle\rangle^{\bar{0}}((\neg\langle\langle 1,2\rangle\rangle^{\downarrow}\mathbf{X}\,\mathsf{fail})\mathbf{U}\,\mathsf{halt})$ if, and only if, $\widehat{\mathfrak{M}}_1^{\mathcal{A}}, q_{\mathbf{init}}, \bar{0} \models \langle\langle 1\rangle\rangle^{\bar{0}}((\neg\langle\langle 1\rangle\rangle^{\downarrow}\mathbf{X}\,\mathsf{fail})\mathbf{U}\,\mathsf{halt})$. □

Remark 2. We note that we can further simplify model $\widehat{\mathfrak{M}}_1^{\mathcal{A}}$. For example, states of type $s'\underline{C_1C_2}$ are not needed. Keeping them, however, allows us to reuse the previous notation and thus simplifies the presentation.

4.2. Undecidability of prRAL *and* rfRAL *with one resource type over* **iRBM***s*

The reductions presented in the previous section use two resource types to simulate the two counters of the two-counter machine. In this section we show that the model checking problem for prRAL and rfRAL remain undecidable even if each agent has only one resource type available. This shows that proponent restriction and resource flatness are essential even with one resource type. Settings restricted to a single resource type are an important special case, as having only one resource type available might be expected to make model checking less complex. For example, with one resource type, the complexity of RB±ATL goes from EXPSPACE-hard to PSPACE-complete [8,9]. The reductions in the case of one resource type are more complex. Specifically, the number of agents doubles: instead of one agent in the proponent-restricted setting, we need two; and instead of two agents in the resource-flat setting, we need four.

4.2.1. Encoding two-counter machines

The encoding of the TCM is very similar to the encoding presented in Section 4.1.1. The key difference is that we can no longer use a single action to update both resource types at the same time. The actions must be split into two. Instead of an action E_1E_2, we introduce two actions $\underline{E}_1E_2$ and $E_1\underline{E}_2$, where the first and second action are controlled by the first and second agent, respectively, and change the agent's single resource type according to E_1 and E_2, respectively. Similarly, actions of type $s^{E_1E_2}_{C_1C_2}$ are split into $s^{\underline{E}_1E_2}_{\underline{C}_1C_2}$ and $s^{E_1\underline{E}_2}_{C_1\underline{C}_2}$. The underscore indicates which parts of the action should be used to update the resources of the agent executing the action.

The technical presentation requires a little more notation. The decomposition ensures that the agents coordinate their actions so that the action tuples consisting of the actions of the first and second agent have a counterpart in the TCM. This is best illustrated by an example. Suppose the automaton contains the transitions $((s, 0, 1), (s', 1, 1))$ and $((s, 1, 0), (s', 1, 1))$, and that these are the only two transitions which should be enabled in state s. In the new encoding, agent 1 would have the actions $\underline{0}1$, $\underline{1}0$, and idle in its repertoire at state s. Similarly, agent 2 has actions $0\underline{1}$, $1\underline{0}$, and idle available at state s.

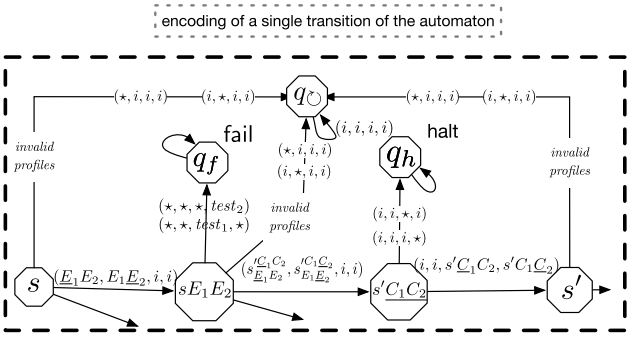

Fig. 4. Excerpt of model $\mathfrak{M}_2^{\mathcal{A}}$. Encoding of a transition $(s, E_1, E_2)\Delta(s', C_1, C_2)$ using only one resource type and four agents. In the figure, we use i to refer to the idle action idle.

As both agents are autonomous decision makers they are free to choose actions independently. Hence, the action profile $(\underline{0}1, 1\underline{0})$ may result from their action selection. Clearly, this is an undesirable action tuple, as it does not correspond to any transition of the automaton. We need to ensure that such action profiles never yield a behaviour which encodes an accepting run of the automaton. Therefore, the model is constructed in such a way that *invalid* action profiles result in the loop state $q_{\circlearrowleft}$.

With this change, the other parts of the previous encoding can be used with only minor modifications. Agents $\{1, 2\}$ are used to simulate the behaviour of the automaton, and agents $\{3, 4\}$ play the role of the spoiler agents who ensure that the simulation is sound. Thus, the coalition $\{3, 4\}$ is used to encode the zero-test. This requires that agent 3 mirrors the resources of agent 1, and agent 4 the resources of agent 2. The new encoding of a transition is illustrated in Fig. 4, and the formal definition of the model is given in Appendix A.2 (Definition 7).

4.2.2. Properties of the encoding: the simulation lemma

Analogously to Section 4.1.2, we present a simulation lemma. We also need to introduce computation pre-encodings etc. with respect to $\mathfrak{M}_2^{\mathcal{A}}$. For the sake of readability, we mostly refrain from giving formal definitions, and focus on the key modifications. An *$\mathcal{A}$-computation pre-encoding of $\mathfrak{M}_2^{\mathcal{A}}$* is defined as for $\mathfrak{M}_1^{\mathcal{A}}$, but the initial condition is changed to $\eta(a, R) = \eta(b, R)$ for agents $(a, b) \in \{(1, 3), (2, 4)\}$ where R refers to the single resource type. (In the following we use a and b to denote the agents, where b simulates the resources of a.) That is, the initial endowment for agents 1 and 3, as well as for agents 2 and 4 must be identical. With this notion we can also prove basic properties analogously to Proposition 1. The new version of Proposition 1 reads as follows:

(a) We have that $\eta^{3(k-1)+1}(a, R) = \eta^{3(k-1)+1}(b, R)$ for $(a, b) \in \{(1, 3), (2, 4)\}$.
(b) If $\lambda \circ (sE_1E_2, \eta)(s'\underline{C_1C_2}, \eta')$ is a resource-extended path, then $\lambda \circ (sE_1E_2, \eta)(s'\underline{C_1C_2}, \eta')(s', \eta'')$ is a resource-extended path with a uniquely defined endowment η''.
(c) If $\lambda[i] = (q^i, \eta^i) = (sE_1E_2, \eta^i)$ with $E_a = 0$, then $(\eta^i(a, R) = 0$ if and only if $\eta^i(b, R) = 0)$, where $(a, b) \in \{(1, 3), (2, 4)\}$.

(b) remains unchanged, and expresses that agents 3 and 4 correctly mirror the resources of agents 1 and 2, respectively. We introduce the revised notion of simulation. We say that the $\mathcal{A}$-computation pre-encoding $\lambda = (q^i, \eta^i)_{i\in\{1,\ldots,3(k-1)+1\}}$ of $\mathfrak{M}_2^{\mathcal{A}}$, $k \in \mathbb{N}$, *simulates the $\mathcal{A}$-computation* ρ (wrt. $\mathfrak{M}_2^{\mathcal{A}}$) if the following holds:

1. ρ has length k;
2. for every $i \in \{1, \ldots, k\}$ if $\rho[i] = (s, v_1, v_2)$ then $\lambda[3(i-1)+1] = (s, \eta)$ with $\eta(1, R) = v_1$ and $\eta(2, R) = v_2$; and
3. for any configuration $\lambda[i] = (sE_1E_2, \eta)$ on λ with $E_r = 0$, $r \in \{1, 2\}$, it holds that $\eta(r, R) = 0$.

Note that counters now refer to the unique resource type of different agents, rather than to different resource types of a single agent.

Analogously to Lemma 1 we can prove the following adapted simulation lemma.

Lemma 4 *(Simulation Lemma for $\mathfrak{M}_2^{\mathcal{A}}$). There is a bijection $f^{\mathcal{A}}$ between computations of $\mathcal{A}$ and $\mathcal{A}$-computation encodings of $\mathfrak{M}_2^{\mathcal{A}}$ such that $f^{\mathcal{A}}(\rho)$ simulates the computation ρ. In particular, if ρ is an accepting computation then $f^{\mathcal{A}}(\rho)$ is also accepting.*

4.2.3. Resource-flat fragment with one resource type

In this section we prove undecidability of rfRAL with only one resource type. We proceed analogously to Section 4.1.3. We show that the empty-band TCM $\mathcal{A}$ halts if, and only if, $\mathfrak{M}_2^{\mathcal{A}}, s_{\text{init}}, \bar{0} \models \langle\langle 1, 2\rangle\rangle^{\bar{0}}_{\{1,2,3,4\}}\mathbf{F}\,\mathsf{halt}$. Again, we observe that for any encoding of a transition $(s, E_1, E_2)\Delta(s', C_1, C_2)$ shown in Fig. 4, agents 1 and 2 (resp. 3 and 4) have no incentive to idle in

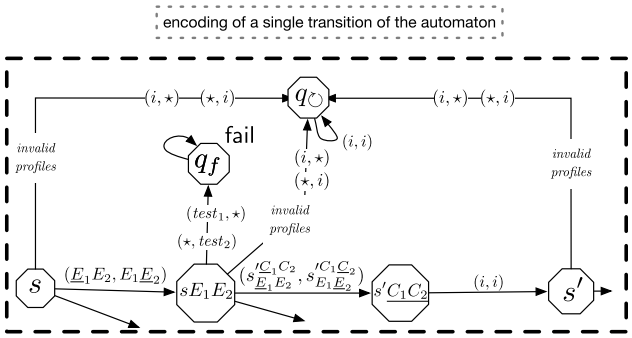

Fig. 5. Excerpt of model $\widehat{\mathfrak{M}}_2^{\mathcal{A}}$. Encoding of a transition $(s, E_1, E_2)\Delta(s', C_1, C_2)$ using only one resource type and two agents. Again, we denote the idle action idle by i.

states s and sE_1E_2 (resp. in state $s'\underline{C_1C_2}$). Instead of looking for a winning strategy for $\{1\}$ we look for a winning strategy for $\{1,2\}$. The key idea is that the coalition behaves in such a way that their combined action corresponds to the action selection of $\{1\}$ in Section 4.1.3. Given the reformulation of properties above, we can make the following observations:

- If $\mathcal{A}$ halts, then coalition $\{1,2\}$ simulates the transitions of the machine's accepting run. Due to the simulation, coalition $\{3,4\}$ will never be able to enforce the fail state q_f. Moreover, either agent b's resources correctly mirror agent a's resources, or agent b performs the idle action for $(a,b) \in \{(1,3),(2,4)\}$. In both cases, either an accepting state or the auxiliary halting state q_h, both labelled halt, are reached. The formula is true.
- Let the formula be true. Coalition $\{1,2\}$ must have a strategy that guarantees reaching a state labelled halt against all strategies of $\{3,4\}$, including the collective strategy of $\{3,4\}$ in which an idle action is never performed in states of type $s'\underline{C_1C_2}$. This strategy of agent b correctly mirrors agent a's resources and ensures that agent a's strategy only selects enabled transitions, $(a,b) \in \{(1,3),(2,4)\}$. Thus, the strategy of $\{1,2\}$ yields an $\mathcal{A}$-computation encoding. By Simulation Lemma 4 the TCM has an accepting run and halts.

Formally, we capture the previous discussion in the following Lemma and Theorem.

Lemma 5. *The empty-band TCM $\mathcal{A}$ halts iff* $\mathfrak{M}_2^{\mathcal{A}}, s_{init}, \bar{0} \models \langle\langle 1,2\rangle\rangle^{\bar{0}}_{\{1,2,3,4\}}\mathbf{F}\,\mathsf{halt}$.

Theorem 3. *Model checking* rfRAL *over the class of* **iRBM***s is undecidable, even in the case of a single resource type and four agents.*

The proofs of Lemma 5 and Theorem 3 are analogous to those of Lemma 2 and Theorem 1, respectively.

4.2.4. The proponent restricted fragment with one resource type

Finally, we consider the proponent-restricted fragment. Again, the line of argument is analogous to the one followed in Section 4.1.4, but there is one caveat when it comes to merging the agents. We first state the analogue of Lemma 3.

Lemma 6. *The empty-band TCM $\mathcal{A}$ halts iff* $\mathfrak{M}_2^{\mathcal{A}}, q_{init}, \bar{0} \models \langle\langle 1,2,3,4\rangle\rangle^{\bar{0}}((\neg\langle\langle 1,2,3,4\rangle\rangle^{\downarrow}\mathbf{X}\,\mathsf{fail})\mathbf{U}\,\mathsf{halt})$.

Next, we observe that the agents $\{1,2,3,4\}$ act as a team. Thus, we can consider their decision making as the decision making of a single "merged" agent. However, in contrast to the setting of Section 4.1.4, we cannot explicitly model this by a single agent as there is only one resource type. Thus, we merge agents $\{1,3\}$ and agents $\{2,4\}$ into two distinct agents. The resulting model $\widehat{\mathfrak{M}}_2^{\mathcal{A}}$ is illustrated in Fig. 5.

Theorem 4. *Model checking* prRAL *over the class of* **iRBM***s is undecidable, even in the case of a single resource type and two agents.*

The proof is given in Appendix A.2.

4.3. The positive proponent-restricted fragment of RAL

Following the observation made in the previous sections about the proponent-restricted variants of RAL (cf. Theorems 2 and 4) we define a proponent-restricted but not resource-flat fragment of RAL, pprRAL, that has a decidable model checking property over **iRBM**s. We first define the *positive fragment* of RAL as the set of all RAL-formulae where no cooperation modality is under the scope of a negation symbol.

Definition 6 *(The fragment* pprRAL*).* The logic pprRAL is defined as the proponent-restricted and positive fragment of RAL.[8]

As noted in the introduction, the pprRAL-fragment allows us to express properties of coalitions of agents which reconsider their strategies *without* being re-equipped with fresh resources. For example, we can formalise the property "given their initial battery charge, rescue robots A can safely get to a position from which they can perform rescue while in visual contact with the base" as: $\langle\langle A\rangle\rangle^{\eta_{\text{init}}}(\text{safe}\,\mathbf{U}(\langle\langle A\rangle\rangle^{\downarrow}(\text{visual}\,\mathbf{U}\,\text{rescue})))$. Intuitively, this reflects the constraint that the robots cannot recharge their batteries after reaching the position where they can perform rescue while in visual contact with the base. This is expressible in pprRAL but not in rfRAL. Another example is the formula $\langle\langle 1,2\rangle\rangle^{\eta_{\text{init}}}\mathbf{F}(\text{rob}\wedge\langle\langle 1\rangle\rangle^{\downarrow}\mathbf{F}\,\text{escape})$, expressing that the coalition $\{1,2\}$ can cooperate to eventually rob a bank, following which agent 1 has a strategy to escape on its own using only its remaining resources.

Before we show the decidability of pprRAL over **iRBM**s in the next section, we make the following observation which follows from [2, Theorem 6]:

Observation 4. *Model checking* pprRAL *over* **RBM***s is undecidable.*

To give a flavour of the basic idea of the undecidability proof, let us consider Fig. 2. We have to modify the construction in such a way that, in test states sE_1E_2, the opponent can always execute a test action corresponding to a counter that should be zero, after which a new state is reached in which the proponent has to execute a specific action. In a sense the opponent can challenge the proponent to execute this specific action. Now, there are two options. First, the proponent has not sufficient resources to execute the action, which means that the counter is simulated correctly. In that case the history leading to the current state is disregarded as it cannot be extended to a resource-extended path (recall that such paths have to be infinite). Second, the proponent can execute the action. This would result in a new fail state labelled with a specific proposition, say error, indicating that the reduction is flawed. Then, the simulation is continued by connecting the fail state with the state $s'\underline{C_1C_2}$ which would have been reached if the opponent had not executed the test action. Given this modification, we can show formally that $\mathfrak{M}, q_{init}, \bar{0} \models \langle\langle 1\rangle\rangle^{\bar{0}}(\neg\text{error})\mathbf{U}\,\text{halt}$ iff the automaton $\mathcal{A}$ halts on the empty input, where $\mathfrak{M}$ is a modified version of $\mathfrak{M}_2^{\mathcal{A}}$ essentially along the lines sketched above (in particular, all idle actions are removed).

It is important to note that in the presence of idle actions, this reduction no longer works, as the proponent always has a choice. Even in the case where the proponent has no resources left, the computation of the system can always be extended to be infinite, either by visiting the fail state, or by looping in some state. As a consequence, a halting state may never be reached. This implies that the opponent has too much power, and can always spoil the simulation by performing a test action in cases where the simulation is sound and no resources are available. That there is no way to save the reduction is shown by the decidability result we present in the next section.

5. Model-checking pprRAL over iRBMs

In this section we prove that the model checking problem for the fragment pprRAL over **iRBM**s is decidable. We first present the model checking algorithm for pprRAL over **iRBM**s, and then prove termination and correctness of the algorithm in Lemmas 7 and 8, respectively.

5.1. Model checking algorithm for pprRAL

The model checking algorithm for pprRAL over **iRBM**s takes as input a model $\mathfrak{M}$, formula ϕ, and initial endowment η, and labels the set of states $[\phi]^{\eta}_{\mathfrak{M}}$, where $[\phi]^{\eta}_{\mathfrak{M}} = \{q \mid \mathfrak{M}, q, \eta \models \phi\}$ is the set of states satisfying ϕ (see Algorithm 1).[9]

Algorithm 1 Labelling ϕ.

1: **procedure** LABEL($\mathfrak{M}, \phi, \eta$)
2: **for** $\phi' \in Sub(\phi)$ **do**
3: $[\phi']^{\eta}_{\mathfrak{M}} \leftarrow$ ATL-LABEL($\mathfrak{M}, \phi'$)
4: $[\phi]^{\eta}_{\mathfrak{M}} \leftarrow \{\, q \mid q \in Q \wedge$ STRATEGY($node_0(q, \eta, \eta, prop(\phi)), \phi$)$\}$

Given ϕ, we produce the set of subformulae of ϕ, $Sub(\phi)$, in the usual way, except that $\langle\langle A\rangle\rangle^{\downarrow}$ and $\langle\langle A\rangle\rangle^{\zeta}$ modalities are replaced by standard ATL modalities $\langle\langle A\rangle\rangle$. $Sub(\phi)$ is ordered in increasing order of complexity. For a formula $\phi' \in Sub(\phi)$, we will write $s \models \phi'$ to indicate that state s has been labelled by ϕ'. Note that if a state s is not annotated with the standard

[8] A more restricted version of pprRAL was introduced in [7], where in addition the formula φ_1 on the left-hand-side of $\varphi_1\mathbf{U}\varphi_2$ is constrained to be propositional.

[9] The model checking algorithm for pprRAL is a slightly modified version of the algorithm given in [7]. In particular, Algorithm 4 incorporates an extra call to STRATEGY to allow arbitrary positive formulae on the left of **U**. Other changes simply clarify the presentation and/or correct minor bugs in the algorithm given in [7].

ATL modality $\langle\langle A\rangle\rangle$, then it cannot satisfy $\langle\langle A\rangle\rangle^{\downarrow}$ or $\langle\langle A\rangle\rangle^{\zeta}$. Algorithm 1 simply labels states with the subformulae of ϕ using the standard ATL labelling algorithm [32] (lines 2–3). It then calls the function STRATEGY to label states with ϕ (line 4). *prop* is a function that returns either the proponents $A \subseteq \mathbb{A}\text{gt}$ if ϕ is of the form $\langle\langle A\rangle\rangle^{*}\mathbf{X}\psi$, $\langle\langle A\rangle\rangle^{*}\psi_1\mathbf{U}\psi_2$, $\langle\langle A\rangle\rangle^{*}\mathbf{G}\psi$ where $*$ is either $\downarrow$ or an endowment, or $\mathbb{A}\text{gt}$ otherwise. The function $node_0$ initialises the root node for the function STRATEGY and is explained below.[10]

Algorithm 2 Strategy.

```
1: function STRATEGY(n, φ)
2:     case φ = p ∈ Π
3:         return s(n) ⊨ φ
4:     case φ = ¬p where p ∈ Π
5:         return s(n) ⊭ φ
6:     case φ = ψ1 ∧ ψ2
7:         return STRATEGY(node0(s(n), e(n), v(n), c(n)), ψ1) ∧ STRATEGY(node0(s(n), e(n), v(n), c(n)), ψ2)
8:     case φ = ψ1 ∨ ψ2
9:         return STRATEGY(node0(s(n), e(n), v(n), c(n)), ψ1) ∨ STRATEGY(node0(s(n), e(n), v(n), c(n)), ψ2)
10:    case φ = ⟨⟨A⟩⟩↓ Xψ
11:        return X-STRATEGY(node0(s(n), e(n), v(n), A), φ)
12:    case φ = ⟨⟨A⟩⟩ζ Xψ
13:        return X-STRATEGY(node0(s(n), ζ, ζ, A), φ)
14:    case φ = ⟨⟨A⟩⟩↓ ψ1Uψ2
15:        return U-STRATEGY(node0(s(n), e(n), v(n), A), φ)
16:    case φ = ⟨⟨A⟩⟩ζ ψ1Uψ2
17:        return U-STRATEGY(node0(s(n), ζ, ζ, A), φ)
18:    case φ = ⟨⟨A⟩⟩↓ Gψ
19:        return G-STRATEGY(node0(s(n), e(n), v(n), A), φ)
20:    case φ = ⟨⟨A⟩⟩ζ Gψ
21:        return G-STRATEGY(node0(s(n), ζ, ζ, A), φ)
```

The function STRATEGY is shown in Algorithm 2 and proceeds by depth-first and-or search. That is, we examine each path in the search space in turn, as in standard depth-first search, but treat nodes corresponding to a particular choice of action by A as and-nodes, i.e., all branches corresponding to this choice must return true for the choice to be part of a successful strategy. The function STRATEGY processes each coalition modality in turn, starting from the outermost modality. The logical connectives are standard, and simply call STRATEGY on the subformulae. Each temporal operator is handled by a separate function: X-STRATEGY for $\mathbf{X}\psi$, U-STRATEGY for $\psi_1\mathbf{U}\psi_2$, and G-STRATEGY for $\mathbf{G}\psi$, and are explained below. We record information about the state of the search in a search tree of nodes. A *node* is a structure which consists of a state of $\mathfrak{M}$, the resources available to all the agents in that state, and a finite path of nodes leading to this node from the root node. Edges in the search tree correspond to joint actions by all agents. Note that the resources available to the agents in a state on a path constrain the edges from the corresponding node to be those action profiles α_A where for all proponent agents a, $(\mathsf{cons}(\alpha_a, r))_{r\in Res}$ is less than or equal to the available resources of agent a. We compare vectors of resources in the usual way; for example, $\zeta_a \geq (\mathsf{cons}(\alpha_a, r))_{r\in Res}$ stands for $\zeta_a(r) \geq \mathsf{cons}(\alpha_a, r)$ for all resources r. For an action profile α_A of $A \subseteq \mathbb{A}\text{gt}$, we write $\mathsf{cons}(\alpha)$ to refer to the tuple $((\mathsf{cons}(\alpha_a))_{r\in Res})_{a\in A}$. For each node n in the tree, we have a function $s(n)$ which returns its state, $p(n)$ which returns the sequence of nodes on the path to n, $e(n)$ which returns an endowment specifying the resource availability for all agents as a result of following $p(n)$, $v(n)$ which returns the resources potentially available to the agents as a result of traversing cycles on $p(n)$ additional times,[11] and $c(n)$ which returns the current set of proponents. The function $node_0(s, \eta, \eta', A)$ returns the root node, i.e., a node n_0 such that $s(n_0) = s$, $p(n_0) = [\,]$ (empty list), $e(n_0) = \eta$, $v(n_0) = \eta'$, and $c(n_0) = A \subseteq \mathbb{A}\text{gt}$ is the current set of proponents. The function $node(n, s', \alpha, A)$ returns a node n' where $s(n') = s'$, $p(n') = [p(n) \cdot n]$, $c(n') = A = c(n)$,

$$e_a(n') = \begin{cases} e_a(n) & \text{if } a \notin A \\ e_a(n) + \mathsf{prod}(\alpha) - \mathsf{cons}(\alpha) & \text{if } a \in A \end{cases}$$

and

$$v_a(n') = \begin{cases} v_a(n) & \text{if } a \notin A \\ v_a(n) + \mathsf{prod}(\alpha) - \mathsf{cons}(\alpha) & \text{if } a \in A \end{cases}$$

where vectors are added and subtracted as usual unless their components are not integers. For technical reasons, we introduce an extra value for an agent's resource endowment, *arb*, which denotes an arbitrary finite value; for any $m \in \mathbb{Z}$, $m < arb$. *arb* cannot be incremented or decremented: $arb + m = arb$ and $arb - m = arb$. Above, $e(n)(a, r)$ is used to keep track of the 'real' cost of the loops and does not contain *arb* values, while $v(n)(a, r) = arb$ indicates that the path $p(n)$

[10] Note that we do not label states with subformulae of ϕ involving $\langle\langle A\rangle\rangle^{\downarrow}$ or $\langle\langle A\rangle\rangle^{\zeta}$ modalities as in [6].

[11] A cycle on $p(n)$ is a subsequence of $p(n)$ with start and end nodes sharing the same state.

contains a *productive loop*, which can be traversed multiple times to generate an arbitrary finite amount of resource r for agent a. Intuitively, *arb* represents an arbitrary finite number; hence, having *arb* resources allows the agent to execute any action as well as any finite number of loop traversals, but does not allow the agent to traverse a loop infinitely many times.

Algorithm 3 X-strategy (both types of modalities).

```
1: function X-STRATEGY(n, ⟨⟨A⟩⟩*Xψ)
2:     if s(n) ⊭ atl(⟨⟨A⟩⟩*Xψ) then
3:         return false
4:     ActA ← {α′ ∈ d_A(s(n)) | cons(α′) ≤ v_A(n)}
5:     for α′ ∈ ActA do
6:         ActAgt ← {α ∈ d(s(n)) | α_A = α′}
7:         strat ← true
8:         for α ∈ ActAgt do
9:             s′ ← o(s(n), α)
10:            strat ← strat ∧ STRATEGY(node(n, s′, α, A), ψ)
11:        if strat then
12:            return true
13:    return false
```

The function X-STRATEGY for formulae of types $\langle\langle A\rangle\rangle^{\downarrow}\mathbf{X}\psi$ and $\langle\langle A\rangle\rangle^{\zeta}\mathbf{X}\psi$ is shown in Algorithm 3 and is straightforward. After checking if the search should be terminated with false because the ATL version of the formula is false (lines 2–3),[12] we simply check if there is an action of A that is possible given the current endowment (line 4), and where in all outcome states A has a strategy to enforce ψ (lines 5–12). $atl(\phi)$ is a function that returns the formula where each $\langle\langle A\rangle\rangle^{\downarrow}$ and $\langle\langle A\rangle\rangle^{\zeta}$ in ϕ is replaced by $\langle\langle A\rangle\rangle$.

The function U-STRATEGY for formulae of types $\langle\langle A\rangle\rangle^{\downarrow}\psi_1\mathbf{U}\psi_2$ and $\langle\langle A\rangle\rangle^{\zeta}\psi_1\mathbf{U}\psi_2$ is shown in Algorithm 4. First U-STRATEGY checks whether the search should be terminated with false because either the ATL version of the formula is false (lines 2–3), or the current path ends in an unproductive loop (lines 4–5). We then check the path for a productive loop, and update $v(n)$ if we find one (lines 6–7). If the ATL version of ψ_2 is true, we try to find a strategy to enforce ψ_2 from $s(n)$, and, if we are successful, U-STRATEGY returns true (lines 8–9). We then check if the endowment in n is insufficient to enforce ψ_1, and terminate the search with false if it is not (lines 10–11). (This check is required as only the ATL version of the formula is checked at lines 2–3.) Otherwise the search continues, as the node where STRATEGY(n, ψ_2) returns true may be found later on the path. Each action available at $s(n)$ is considered in turn (lines 12–20). For each action $\alpha' \in ActA$, we check whether a recursive call of the algorithm returns true in all outcome states s' of α' (i.e., α' is part of a successful strategy). If such an α' is found, the algorithm returns true. Otherwise the algorithm returns false. Note that we never traverse a productive loop more than twice: if an arbitrary amount of the resource(s) produced by the loop is insufficient to enforce ψ_2 (and hence return true), at the beginning of the third traversal the search will be terminated with false at the test for an unproductive loop (since the second traversal of the loop did not result in a change in the endowment).

The function G-STRATEGY for formulae of types $\langle\langle A\rangle\rangle^{\downarrow}\mathbf{G}\phi$ and $\langle\langle A\rangle\rangle^{\zeta}\mathbf{G}\phi$ is shown in Algorithm 5. Again we check if the search should be terminated with false, either because the standard ATL modality does not hold (lines 2–3), or because the current path terminates in a resource consuming cycle (lines 4–7). The first check is for cycles where at least one resource is consumed and no resources are produced (lines 4–5). The second check is for cycles which both produce and consume resources (so the previous test does not apply), and where we have already shown we can produce an arbitrary amount of the resource being consumed (lines 6–7). As any arbitrary amount of resource is insufficient to maintain such a loop indefinitely, we terminate the search with false. We then check the path for a productive loop, and update $v(n)$ if we find one (lines 8–9). Note that, to enforce an invariant, only a path ending in a nondecreasing loop (as opposed to a productive loop) is required. However we must correctly update the endowment available in n in order to evaluate ψ in $\langle\langle A\rangle\rangle^{\downarrow}\mathbf{G}\psi$. We then check if the endowment in n is insufficient to enforce ψ from $s(n)$, and terminate the search with false if it is not (lines 10–11). If the current path terminates in a nondecreasing loop, we return true (lines 12–13): ψ is enforceable from each of the states on the path, and the loop can be traversed indefinitely. Otherwise we continue the search for a nondecreasing loop (lines 14–22).

To illustrate the execution of the algorithm, we revisit the running example from Section 3.4 and consider the property

$$\langle\langle 1\rangle\rangle^{1:5,2:0}\top\mathbf{U}\langle\langle 1\rangle\rangle^{\downarrow}\mathbf{G}p$$

We skip the ATL labelling step and consider the initial call to

$$\text{U-STRATEGY}(node_0(q_0,(1{:}5,2{:}0),(1{:}5,2{:}0),\{1\}),\langle\langle 1\rangle\rangle^{1:5,2:0}\top\mathbf{U}\langle\langle 1\rangle\rangle^{\downarrow}\mathbf{G}p)$$

where $n_0 = node_0(q_0,(1{:}5,2{:}0),(1{:}5,2{:}0),\{1\})$. For n_0, no cases of Algorithm 4 are applicable until line 12. $ActA$ (actions of agent 1 for which the resource consumption is less than $v_1(n_0)$, i.e., less than 5) consists of *idle* and *move* actions.

[12] Note that the checks for the ATL versions of the formula in X-STRATEGY, U-STRATEGY and G-STRATEGY are only for efficiency, and are not required for the correctness of the algorithms.

Algorithm 4 U-strategy (both types of modalities).

```
1: function U-STRATEGY(n, ⟨⟨A⟩⟩* ψ1 U ψ2)
2:     if s(n) ⊭ atl(⟨⟨A⟩⟩* ψ1 U ψ2) then
3:         return false
4:     if ∃n' ∈ p(n) : s(n') = s(n) ∧ v_A(n') ≥ v_A(n) then
5:         return false
6:     for (a, r) ∈ {(a, r) ∈ A × Res | ∃n' ∈ p(n) : s(n') = s(n) ∧ v_A(n') ≤ v_A(n) ∧ v(n')(a, r) < v(n)(a, r)} do
7:         v(n)(a, r) ← arb
8:     if s(n) ⊨ atl(ψ2) ∧ STRATEGY(n, ψ2) then
9:         return true
10:    if ¬ STRATEGY(n, ψ1) then
11:        return false
12:    Act A ← {α' ∈ d_A(s(n)) | cons(α') ≤ v_A(n)}
13:    for α' ∈ Act A do
14:        Act Agt ← {α ∈ d(s(n)) | α_A = α'}
15:        strat ← true
16:        for α ∈ Act Agt do
17:            s' ← o(s(n), α)
18:            strat ← strat ∧ U-STRATEGY(node(n, s', α, A), ⟨⟨A⟩⟩* ψ1 U ψ2)
19:        if strat then
20:            return true
21:    return false
```

Algorithm 5 G-strategy (both types of modalities).

```
1: function G-STRATEGY(n, ⟨⟨A⟩⟩* G ψ)
2:     if s(n) ⊭ atl(⟨⟨A⟩⟩* G ψ) then
3:         return false
4:     if ∃n' ∈ p(n) : s(n') = s(n) ∧ e_A(n') ≥ e_A(n) ∧
          (∃a ∈ A, r ∈ Res : e(n')(a, r) > e(n)(a, r)) then
5:         return false
6:     if ∃n' ∈ p(n) : s(n') = s(n) ∧
          ∀a ∈ A ∀r ∈ Res : (v(n')(a, r) = v(n)(a, r) = arb ∨ e(n')(a, r) = e(n)(a, r)) ∧
          ∃a ∈ A ∃r ∈ Res : e(n')(a, r) > e(n)(a, r) then
7:         return false
8:     for (a, r) ∈ {(a, r) ∈ A × Res | ∃n' ∈ p(n) : s(n') = s(n) ∧ v_A(n') ≤ v_A(n) ∧ v(n')(a, r) < v(n)(a, r)} do
9:         v(n)(a, r) ← arb
10:    if ¬ STRATEGY(n, ψ) then
11:        return false
12:    if ∃n' ∈ p(n) : s(n') = s(n) ∧ e_A(n') ≤ e_A(n) then
13:        return true
14:    Act A ← {α' ∈ d_A(s(n)) | cons(α') ≤ v_A(n)}
15:    for α' ∈ Act A do
16:        Act Agt ← {α ∈ d(s(n)) | α_A = α'}
17:        strat ← true
18:        for α ∈ Act Agt do
19:            s' ← o(s(n), α)
20:            strat ← strat ∧ G-STRATEGY(node(n, s', α, A), ⟨⟨A⟩⟩* G ψ)
21:        if strat then
22:            return true
23:    return false
```

Let us trace the algorithm calls for *idle* first. There are two joint actions we need to consider (line 14): $(idle, idle)$ and $(idle, obstruct)$. In both cases the result will be the same, the next call to U-STRATEGY$(n, \langle\langle 1\rangle\rangle^{1:5,2:0}\top \mathbf{U}\langle\langle 1\rangle\rangle^{\downarrow}\mathbf{G}p)$, where n is the successor node by the joint action, will return false on line 4. This is because n will have the same state (q_0, since $o(q_0, (idle, idle)) = o(q_0, (idle, obstruct)) = q_0$) and $v_1(n) = v_1(n_0)$ (resources available to agent 1 have not changed since *idle* costs nothing). Let us consider the choice of $\sigma = move$ on line 13. There are two joint actions on line 14, $\{(move, idle), (move, obstruct)\}$.

First let us consider $(move, idle)$. On line 17, s' is q_1 and $node(n_0, q_1, (move, idle), \{1\})$ is n_1 where $s(n_1) = q_1$, $p(n_1) = [n_0]$, $e_1(n_1) = v_1(n_1) = 3$ (since move actions cost 2 units of energy, agent 1's resources are decremented) and $e_2(n_1) = v_2(n_1) = 0$ (agent 2's resources do not change since it is not in the proponent coalition). When we call U-STRATEGY$(n_1, \langle\langle 1\rangle\rangle^{1:5,2:0}\top \mathbf{U}\langle\langle 1\rangle\rangle^{\downarrow}\mathbf{G}p)$, the first applicable case is on line 8. The ATL version of $\langle\langle 1\rangle\rangle^{\downarrow}\mathbf{G}p$ is true, and in fact G-STRATEGY$(n_1, \langle\langle 1\rangle\rangle^{\downarrow}\mathbf{G}p)$ returns true. (We will show this after we finish tracing the calls to U-STRATEGY.) So on line 8 the call to U-STRATEGY$(n_1, \langle\langle 1\rangle\rangle^{1:5,2:0}\top \mathbf{U}\langle\langle 1\rangle\rangle^{\downarrow}\mathbf{G}p)$ returns true. Let us consider $(move, obstruct)$. On line 17, s' is q_2 and $node(n_0, q_2, (move, obstruct), \{1\})$ is n_2 where $s(n_2) = q_2$, $p(n_2) = [n_0]$, $e_1(n_2) = v_1(n_2) = 3$ (since move actions cost 2 units of energy, agent 1's resources are decremented) and $e_2(n_2) = v_2(n_2) = 0$ (agent 2's resources do not change since it is not in the proponent coalition). When we call U-STRATEGY$(n_2, \langle\langle 1\rangle\rangle^{1:5,2:0}\top \mathbf{U}\langle\langle 1\rangle\rangle^{\downarrow}\mathbf{G}p)$, the **if** statement on line 8 is not applicable since the invariant formula is not true in q_2. We continue to line 12 and collect all actions by agent 1 with resource con-

sumptions of at most $v_1(n_2)$. Such actions are *idle* and *move*. We skip the case of *idle*, as it is identical to choosing *idle* in n_0. Let us consider *move*. The only joint action possible if agent 1 choses *move* is $(move, idle)$, since in q_2, agent 2 has only the *idle* action. On line 17, s' is q_1 and $node(n_2, q_1, (move, idle), \{1\})$ is n_3 where $s(n_3) = q_1$, $p(n_3) = [n_0, n_2]$, $e_1(n_3) = v_1(n_3) = 1$ (since move actions costs 2 units of energy, agent 1's resources are decremented) and $e_2(n_3) = v_2(n_3) = 0$ (agent 2's resources do not change since it is not in the proponent coalition). When we call U-STRATEGY$(n_3, \langle\langle 1\rangle\rangle^{1:5,2:0}\top \mathbf{U}\langle\langle 1\rangle\rangle^{\downarrow}\mathbf{G}p)$, the call to G-STRATEGY$(n_3, \langle\langle 1\rangle\rangle^{\downarrow}\mathbf{G}p)$ returns true (which will be shown next), and hence all calls to U-STRATEGY from n_0 for the joint actions extending *move* return true.

Now we show that G-STRATEGY$(n_3, \langle\langle 1\rangle\rangle^{\downarrow}\mathbf{G}p)$ returns true (the case of G-STRATEGY$(n_1, \langle\langle 1\rangle\rangle^{\downarrow}\mathbf{G}p)$ is similar but easier, since agent 1 in n_1 has a greater resource availability). None of the cases in Algorithm 5 before line 14 are applicable. Here, the available actions are *idle* and *send* (the agent still has one unit of energy left). If we try *idle*, then the next call to the algorithm will return false on line 2 since *idle* will bring us back to q_0 which does not satisfy the ATL version of the formula $\langle\langle 1\rangle\rangle^{\downarrow}\mathbf{G}p$. If we select *send*, then in the resulting node n_4 the applicable actions are *idle* and *charge*; the choice of *idle* will again lead to failure, but by selecting *charge* we reach n_5 where the state is q_1 again (so $s(n_5) = s(n_3)$), and $e_1(n_5) = e_1(n_3)$, so the algorithm returns true on line 13.

5.2. Correctness of the model checking algorithm

In this section we show that the algorithm always terminates (Lemma 7) and that it gives the correct answer (Lemma 8). Together, the two lemmas give the proof of the main result:

Theorem 5. *The model checking problem for* pprRAL *over* **iRBM***s is decidable.*

Proof. Follows from Lemma 7 and Lemma 8 below. □

Lemma 7. *Algorithm 2 terminates.*

Proof. The proof is by induction on the length of the formula. Calls for propositional formulae clearly terminate. For the inductive step, we need to show that a call for any connective terminates provided calls for lower complexity formulae terminate. Conjunction and disjunction are obvious. X-STRATEGY makes a recursive call to determine if there is a strategy for a smaller complexity formula after one step. The only non-trivial cases are U-STRATEGY and G-STRATEGY.

Let us consider termination of U-STRATEGY first. We need to show that there cannot be an infinite sequence of recursive calls to U-STRATEGY$(node(n, s', \alpha, A), \langle\langle A\rangle\rangle^*\psi_1\mathbf{U}\psi_2)$ (see line 18 of Algorithm 4). Such an infinite sequence would imply that the search is stuck in an infinite loop and hence encounters the same state infinitely often. There are three types of loops to consider: (1) a consuming or neutral loop (where for all proponent agents and all resources, the amount of each resource stays the same or decreases); (2) a productive loop (where for all proponent agents and all resources, the amount of each resource stays at least the same and increases for at least one agent and resource type) and (3) mixed (for some agents and resource types, resource availability increases and for some it goes down). Clearly the search will terminate in case (1) because of the loop check on line 4. Note that we compare v rather than e endowments because we do not want to keep looping after discovering a way to earn *arb* resources. If the agents are in a productive loop (case (2)), eventually all increasing resources are set to *arb* in line 7, and v stops changing, hence we fall back to case (1) and terminate due to the check on line 4. Finally, consider a case of a mixed loop (case (3)). Here we have two sub-cases: (3a) when for at least one agent and resource pair, (a, r), the loop decreases the value of $e(n)(a, r)$, $v(n)(a, r) \neq arb$; and (3b) when for all such pairs where resources are consumed, $v(n)(a, r) = arb$. In case (3a) termination is trivial since the actions which constitute the loop will not be possible after the required resources are consumed. In case (3b) the 'decreasing' resources are all *arb*, so the same actions are still available. However, we assumed that in (3b) for all other pairs (a, r) resources grow (so $v(n)(a, r)$ is eventually set to *arb*) or stay the same. After all growing resources are set to *arb*, there is no further change and the search will terminate in the loop check on line 4. This concludes the proof that U-STRATEGY terminates.

Let us consider G-STRATEGY. Again we need to show that there cannot be an infinite sequence of recursive calls to G-STRATEGY$(node(n, s', \alpha, A), \langle\langle A\rangle\rangle^*\mathbf{G}\psi)$ (line 20 of Algorithm 5). Again such a sequence would need to involve a loop and there are three cases to consider: (1) an increasing or neutral loop (for all agents and resource types, endowments e increase or remain the same); (2) a decreasing loop (also for endowments e); and (3) a mixed loop. In case (1) we terminate on lines 12–13; in case (2) we terminate on lines 4–5. Case (3) again has two subcases: (3a) and (3b). The reasoning is the same as in the case for U-STRATEGY; case (3a) is straightforward, case (3b) is covered by the check on lines 6–7. □

Given $v : \mathbb{A}\text{gt} \times Res \to \mathbb{N}_0^\infty \cup \{arb\}$, the set of endowments compatible with v is defined as $compatible(v) = \{e : \mathbb{A}\text{gt} \times Res \to \mathbb{N}_0^\infty \mid \forall a \in \mathbb{A}\text{gt}, r \in Res : v(a, r) = arb \vee e(a, r) = v(a, r)\}$, i.e., all individual endowments of e for each agent and resource agree with v whenever $v(a, r) \neq arb$.

Lemma 8. *Algorithm 2 is correct, that is,* STRATEGY(n, ϕ) *returns true iff* $\exists e' \in compatible(v(n)) : s(n), e' \models \phi$.

The proof is given in Appendix B.

Table 1
Summary of known decidability and undecidability results.

Logic	Models	
	RBM	**iRBM**
RAL	Undecidable [2]	Undecidable (Corollary of [2])
rfRAL	Undecidable [2]	Undecidable for two agents and two resource types (Theorem 1) Undecidable for four agents and one resource type (Theorem 3)
prRAL	Undecidable [2]	Undecidable (Corollary of [2]) Undecidable for one agent and two resource types (Theorem 2) Undecidable for two agents and one resource type (Theorem 4)
rfprRAL	Undecidable [2]	Decidable (Corollary of [6])
pprRAL	Undecidable (Corollary of [2])	Decidable (Theorem 5)

6. Discussion

Over the last few years, logics for reasoning about strategic, resource-bounded agents and models have become a popular research topic, e.g. [1,3,6,2,14,15,4,5], and, given current trends in the development of intelligent systems (e.g., driverless cars, unmanned vehicles, autonomous robots), the formal verification of resource-bounded systems will become even more important in the near future. Unfortunately, formal, logic-based techniques for the verification of resource-bounded systems are often intractable or even undecidable.

In this paper we investigated the boundary of (un)decidable logics for verifying resource-bounded systems. We identified a significant fragment of Resource Agent Logic (RAL) with a decidable model checking property, and proved two new undecidability results. We have shown that a rather natural property of models — that agents can always decide to do nothing — can make model checking decidable. In particular, the positive proponent-restricted fragment of RAL that we present, pprRAL, is decidable in the presence of idle actions and undecidable without them. However, the availability of idle actions is not sufficient on its own to make the model checking of RAL, or even of the proponent-restricted fragment prRAL, decidable. We show that considering opponents acting under resource bounds makes model checking undecidable, as does allowing coalition modalities in the scope of negations. The summary of known decidability and undecidability results is presented in Table 1.

Note that **iRBM**s are very similar to **RBM**s with finite semantics of [2] (see [8] and [9] for a formal statement of correspondence). The result presented here, together with those of [8,9] implies that pprRAL is decidable over **RBM**s under finite semantics.

Finally, we did not discuss the complexity of the model checking problem for pprRAL over **iRBM**s in this paper. We conjecture that it is the same as the model checking problem for RB±ATL, which was recently shown in [35] to be 2EXPTIME-complete.

Acknowledgements

We thank the reviewers for their detailed and helpful reviews. This work was supported by the Engineering and Physical Sciences Research Council [grant EP/K033905/1].

Appendix A. Encodings and proofs from Section 4

A.1. Encodings and proofs from Section 4.1

Proposition 1. *Let $\mathcal{A}$ be an empty-band TCM and $\lambda = (q^i, \eta^i)_{i=1,\ldots,3(k-1)+1}$ be an $\mathcal{A}$-computation pre-encoding.*

(a) We have that $\eta^{3(k-1)+1}(1, R_r) = \eta^{3(k-1)+1}(2, R_r)$ for $r = 1, 2$.
(b) If $\lambda \circ (sE_1E_2, \eta)(s'\underline{C_1C_2}, \eta')$ is a finite resource-extended path, then so is $\lambda \circ (sE_1E_2, \eta)(s'\underline{C_1C_2}, \eta')(s', \eta'')$ for a uniquely defined endowment η''.
(c) If $\lambda[i] = (q^i, \eta^i) = (sE_1E_2, \eta^i)$ with $E_r = 0$, then ($\eta^i(1, R_r) = 0$ if, and only if, $\eta^i(2, R_r) = 0$), for $r \in \{1, 2\}$.

Proof. Let $\lambda = (q^i, \eta^i)_{i=1,\ldots,3(k-1)+1}$. **(a)** We show this by induction on k. For $k = 1$ the claim follows by definition. Now, suppose the claim is true for all computation pre-encodings up to (and excluding) $k \geq 2$. That is, it is true for $\lambda = (q^i, \eta^i)_{i=1,\ldots,3(k-1)+1}$, which is also an $\mathcal{A}$-computation pre-encoding. That is, $\eta^{3(k-1)+1}(1, R_r) = \eta^{3(k-1)+1}(2, R_r)$ for $r = 1, 2$. Consider the $\mathcal{A}$-computation pre-encoding (where we simply consider how λ has to be extended given the construction of the model)

$$\lambda' = (q^i, \eta^i)_{i=1,\ldots,3k+1}$$

$$= \lambda \circ (q^{3(k-1)+2}, \eta^{3(k-1)+2})(q^{3(k-1)+3}, \eta^{3(k-1)+3})(q^{3k+1}, \eta^{3k+1})$$
$$= \lambda \circ (s^k E_1^k E_2^k, \eta^{3(k-1)+2})(s^{k+1}\underline{C_1^k C_2^k}, \eta^{3(k-1)+3})(s^{k+1}, \eta^{3k+1})$$

Now, a simple computation, taking into consideration the structure of the model, gives: $\eta^{3k+1}(1, R_r) = \eta^{3(k-1)+3}(1, R_r) = \eta^{3(k-1)+2}(1, R_r) + C_r^k + E_r^k = \eta^{3(k-1)+1}(1, R_r) + C_r^k \stackrel{IH}{=} \eta^{3(k-1)+1}(2, R_r) + C_r^k = \eta^{3(k-1)+2}(2, R_r) + C_r^k = \eta^{3(k-1)+3}(2, R_r) + C_r^k = \eta^{3k+1}(2, R_r)$.

(b) We have to show that $\eta''(1, R_r) \geq 0$ and $\eta''(2, R_r) \geq 0$ for $r \in \{1, 2\}$. The former is clear because $\eta'(1, R_r) \geq 0$ and the endowment for player 1 does not change in the transition from $s'C_1C_2$ to s'. For the latter, we make the following observation: $\eta''(2, R_r) = \eta'(2, R_r) + C_r = \eta^{3(k-1)+1}(2, R_r) + C_r$ where the latter equality holds as the resources of agent 2 do not change between the state $q^{3(k-1)+1}$ and $s'C_1C_2$. Now, by (a), we have that $\eta^{3(k-1)+1}(2, R_r) + C_r = \eta^{3(k-1)+1}(1, R_r) + C_r$. Finally, we can compute that $\eta^{3(k-1)+1}(1, R_r) + C_r = \eta(1, R_r) + E_r + C_r = \eta'(1, R_r) \geq 0$ as $\lambda \circ (sE_1E_2, \eta)(s'\underline{C_1C_2}, \eta')$ is a finite resource-extended path by assumption. Finally, η'' is uniquely defined as there is only a unique action from $s'\underline{C_1C_2}$ to s'.

(c) Suppose a configuration (sE_1E_2, η^i) with $E_r = 0$ is reached. Then, action E_1E_2 (resp. idle) was performed in $\lambda[i-1] = (s, \eta^{i-1})$ by agent 1 (resp. 2). Note that none of the actions changes the resource balance of R_r. Suppose now that $\eta^i(x, R_r) = 0$ for $x \in \{1, 2\}$. Then, also $\eta^{i-1}(x, R_r) = 0$ and by (a) $\eta^{i-1}(3-x, R_r) = 0$. With the above observation we can conclude that also $\eta^i(3-x, R_r) = 0$. □

Lemma 1 *(Simulation Lemma for $\mathfrak{M}_1^{\mathcal{A}}$). There is a bijection $f^{\mathcal{A}}$ between computations of $\mathcal{A}$ and $\mathcal{A}$-computation encodings of $\mathfrak{M}_1^{\mathcal{A}}$ in such a way that $f^{\mathcal{A}}(\rho)$ simulates the computation ρ. In particular, if ρ is an accepting computation then $f^{\mathcal{A}}(\rho)$ is also accepting.*

Proof. Let $\rho = (s^i, v_1^i, v_2^i)_{i=1,\ldots,k}$ be a finite $\mathcal{A}$-computation. From ρ we inductively construct the following finite resource-extended path $\lambda = \lambda^\rho = (q^j, \eta^j)_{j=1,\ldots,3(k-1)+1}$.

(a) First, we consider the first configuration $i = 1$. Assume that $k > 1$. Let $\rho_1 = ((s^1, E_1, E_2), (s^2, C_1, C_2))$. We define (i) $q^1 = s^1$, $\eta^1(1, R_r) = \eta^1(2, R_r) = v_r^1$ for $r \in \{1, 2\}$; (ii) $q^2 = s^1E_1E_2$, $\eta^2(1, R_r) = \eta^1(1, R_r) - E_r$ for $r \in \{1, 2\}$; and (iii) $q^3 = s^2\underline{C_1C_2}$, $\eta^3(1, R_r) = \eta^2(1, R_r) + E_r + C_r$ for $r \in \{1, 2\}$. In the case that $k = 1$, we define $q^1 = s^1$, $\eta^1(1, R_r) = \eta^1(2, R_r) = v_r^1$ for $r \in \{1, 2\}$.

(b) Inductively, we assume that the path has been constructed up to position $i-1$ of computation ρ, that is up to $3(i-1)+3$ on λ. Suppose $i < k$ and let $\rho_i = ((s^i, E_1, E_2), (s^{i+1}, C_1, C_2))$. Again, we define (i) $q^{3i+1} = s^i$, $\eta^{3i+1}(1, R_r) = v_r^i$ for $r \in \{1, 2\}$; (ii) $q^{3i+2} = s^iE_1E_2$, $\eta^{3i+2}(1, R_r) = \eta^{3i+1}(1, R_r) - E_r$ for $r \in \{1, 2\}$; and (iii) $q^{3i+3} = s^{i+1}\underline{C_1C_2}$, $\eta^{3i+3}(1, R_r) = \eta^{3i+2}(1, R_r) + E_r + C_r$ for $r \in \{1, 2\}$. In the case that $k = i$, we define $q^{3i+1} = s^i$, $\eta^{3i+1}(1, R_r) = v_r^i$ for $r \in \{1, 2\}$.

First, we prove the following claim which is essential for the rest of the proof.

Claim. *For every $i = 1, \ldots, k$ we have that (i) $s^i = q^{3(i-1)+1}$, (ii) $v_r^i = \eta^{3(i-1)+1}(1, R_r)$ for $r \in \{1, 2\}$, and if $i < k$, then (iii) $q^{3(i-1)+2} = s^iE_1^iE_2^i$ with $E_r^i = 0$ for $r \in \{1, 2\}$ implies $\eta^{3(i-1)+2}(1, R_r) = 0$, and also (iv) $v_r^{i+1} = \eta^{3(i-1)+3}(1, R_r)$ for $r \in \{1, 2\}$.*

Proof of claim. We prove the claim by induction on $i \leq k$.

Induction base: Let $i = 1 \leq k$. We have that (i) $s^1 = q^1$ and (ii) $v_r^1 = \eta_j^1(1, R_r)$ by construction.

Induction step: Suppose the claim is true up to $i < k$. We show the case for $i+1$. First, suppose that $i+1 = k$. By induction, we have that $\eta^{3(i-1)+1}(1, R_r) = v_r^i$, $\eta^{3(i-1)+3}(1, R_r) = v_r^{i+1}$, and by Proposition 1(a) $\eta^{3(i-1)+1}(1, R_r) = \eta^{3(i-1)+1}(2, R_r)$. Thus, by Proposition 1(b) configuration $q^{3i+1} = (s^{i+1}, \eta^{3i+1})$ can be reached. Moreover, the resources for agent 1 do not change in the step transition from $q^{3(i-1)+3}$ to q^{3i+1}, which shows that $\eta^{3i+1}(1, R_r) = v_r^{i+1}$.

Now, we consider the case $i + 1 < k$. Cases (i) and (ii) follow completely analogously. By construction, if $q^{3i+2} = s^{i+1}E_1^{i+1}E_2^{i+1}$ with $E_r^{i+1} = 0$ then $\rho_{i+1} = ((s^{i+1}, E_1, E_2), (s^{i+2}, C_1, C_2))$ with $E_r = 0$. This transition can only be taken by the automaton if $v_r^{i+1} = 0$. Then, by (ii) $\eta^{3i+1}(1, R_r) = 0$. Finally, action $E_1^{i+1}E_2^{i+1}$ can only consume resources by construction of the automaton. This shows that also $\eta^{3i+2}(1, R_r) = 0$. For (iv) we observe that $\eta^{3i+2}(1, R_r) = \eta^{3i+1}(1, R_r) - E_r^{i+1}$ and thus $\eta^{3i+3}(1, R_r) = \eta^{3i+2}(1, R_r) + C_r + E_r^{i+1} = \eta^{3i+1}(1, R_r) + C_r \stackrel{(ii)}{=} v_r^{i+1} + C_r = v_r^{i+2}$. □

Using the claim we now have to show that the thus constructed sequence is indeed an $\mathcal{A}$-computation pre-encoding (Definition 4) and that it is a simulation (Definition 5). The two conditions of Definition 4 follow immediately. Also, by the claim it follows that λ is indeed a path in $\mathfrak{M}_1^{\mathcal{A}}$.

For the first condition of Definition 5, we consider two cases. If $\rho = (s, v_1, v_2)$ consists of a single configuration, then by (a) $\lambda = (s, \eta)$. Hence, λ has length $3 \cdot 0 + 1$. For the second case, $|\rho| = k > 1$, we observe that we added for each $i < k$, three states to λ (s, sE_1E_2, and $s'\underline{C_1C_2}$), and an additional state for $i = k$. Thus, λ has length $3(k-1)+1$. The other conditions (2) and (3) of the Definition follow from (i–iii) of the claim.

We refer to the thus constructed path as $f^{\mathcal{A}}(\rho)$. Different ρ's yield different $f^{\mathcal{A}}(\rho)$'s. It remains to show that $f^{\mathcal{A}}$ is surjective. For an arbitrary $\mathcal{A}$-computation encoding λ, each triple of states $(s)(sE_1E_2)(s'\underline{C_1C_2})$ on λ defines a unique transition $\tau_i = ((s, E_1, E_2), (s', C_1, C_2))$. Given the initial configuration (q^1, η^1) with $\eta^1(1, R_r) = \eta^1(2, R_r)$ for $r \in \{1, 2\}$ and the sequence of the τ_i's, we can compute the computation ρ. It is immediate that $f^{\mathcal{A}}(\rho) = \lambda$. □

Lemma 2. *The empty-band TCM $\mathcal{A}$ halts iff $\mathfrak{M}_1^{\mathcal{A}}, s_{init}, \bar{0} \models \langle\langle 1\rangle\rangle^{\bar{0}}_{\{1,2\}}\mathbf{F}\,\mathsf{halt}$.*

Proof. Let the empty-band TCM $\mathcal{A} = (S, s_{\text{init}}, S_f, \Delta)$ be given and $\mathfrak{M}_1^{\mathcal{A}}$ be the **iRBM** constructed according to Definition 3.

"$\Rightarrow$" Assume that $\mathcal{A}$ halts and let $\rho = (s^i, v_1^i, v_2^i)_{i=1,\ldots,k}$ be a minimal length accepting run of $\mathcal{A}$. By Lemma 1 (Simulation Lemma), $\lambda = f^{\mathcal{A}}(\rho) = (q^j, \eta^j)_{j=1\ldots3(k-1)+1}$ is an accepting $\mathcal{A}$-computation encoding that simulates ρ. Each subsequent configurations (q^i, η^i) and (q^{i+1}, η^{i+1}), for $i = 1, \ldots, 3(k-1)+1$ on λ define a unique action of agent 1. Let s_1 be the strategy which assigns to each history $q^1 \ldots q^i$ this unique action of agent 1 leading to q^{i+1}. The strategy assigns idle to all other histories, including λ itself. We assume that player 1 follows strategy s_1. As λ simulates ρ, agent 2 can never perform a test action in states of type sE_1E_2 according to Definition 5.3 and Proposition 1(c). Thus, agent 2 can only choose between actions in states of type $s\underline{C_1C_2}$, otherwise it can only perform a unique action (the idle action). As a consequence, the outcome set wrt. s_1 contains the following paths:

$$\begin{aligned} out(q^1, \bar{0}, s_1, \{1,2\}) = \{&(q^1, \eta^1) \ldots (q^k, \eta^k)(q_{\circlearrowleft}, \eta^{k+1}) \ldots\} \\ \cup\{&(q^{3(i-1)+1}, \eta^{3(i-1)+1}) \\ &(q^{3(i-1)+2}, \eta^{3(i-1)+2}) \\ &(q^{3(i-1)+3}, \eta^{3(i-1)+3})_{i=1,\ldots,j}(q_h, \eta^{3j+1}) \ldots \mid 1 \le j < k\}. \end{aligned}$$

All these paths contain a state labelled halt. This shows that $\mathfrak{M}_1^{\mathcal{A}}, s_{\text{init}}, \bar{0} \models \langle\langle 1\rangle\rangle^{\bar{0}}_{\{1,2\}}\mathbf{F}\,\mathsf{halt}$.

"$\Leftarrow$" Assume that $\mathfrak{M}_1^{\mathcal{A}}, s_{\text{init}}, \bar{0} \models \langle\langle 1\rangle\rangle^{\bar{0}}_{\{1,2\}}\mathbf{F}\,\mathsf{halt}$ holds. Then, there is a witnessing strategy s_1 of agent 1 such that for all $\lambda = (q^i, \eta^i)_{i\in\mathbb{N}} \in out(s^1, \bar{0}, s_1, \{1,2\})$ there is a minimal index k such that $\pi(q^k) = \mathsf{halt}$. In particular, the set contains a path in which the state q_h is never visited. This follows from Proposition 1(b). We denote the prefix of this path that is cut directly after the state labelled halt by λ'. On λ' it can never be the case that in a state $q^i = sE_1E_2$ with $E_r = 0$ we have that $\eta^i(2, E_r) > 0$; otherwise, the outcome would also contain a path which loops in q_f because agent 2 could perform the test action. But this would contradict s_1 being a witnessing winning strategy. Thus, we have $\eta^i(2, E_r) = 0$ and by Proposition 1(c) also $\eta^i(1, E_r) = 0$, for $r \in \{1,2\}$. Thus, by the Simulation Lemma, there is ρ with $f^{\mathcal{A}}(\rho) = \lambda'$ that is an accepting run of the automaton. The automaton halts. □

A.2. Encodings and proof from Section 4.2

The formal definition of the encoding of a two counter automaton as a resource-bounded model with idle actions and a single resource type is given next.

Definition 7 *($\mathfrak{M}_2^{\mathcal{A}}$).* Let $(S, s_{\text{init}}, S_f, \Delta)$ be an empty-band TCM. From $\mathcal{A}$ we construct the **iRBM** $\mathfrak{M}_2^{\mathcal{A}} = (\{1,2,3,4\}, Q, \Pi, \pi, Act, d, o, \{R\}, t)$ where:

1. The sets of states $Q = S \cup Q_1 \cup Q_2 \cup \{q_f, q_h, q_{\circlearrowleft}\}$ and of propositions and their valuations are defined as in Definition 3.
2. The set Act is defined as follows. For each transition $(s, E_1, E_2)\Delta(s', C_1, C_2)$ of $\mathcal{A}$ the set contains actions $\underline{E_1}E_2$, $E_1\underline{E_2}$, $s'^{\underline{C_1}C_2}_{\underline{E_1}E_2}$, $s'^{C_1\underline{C_2}}_{E_1\underline{E_2}}$, $s'\underline{C_1}C_2$, and $s'C_1\underline{C_2}$. Additionally, there is an action idle and test actions $test_i$ for $i \in \{1,2\}$.
3. The action availability is defined according to Δ. For agent 1 we have:

$$\begin{aligned} \underline{E_1}E_2 \in d_1(s) &\text{ iff } (s, E_1, E_2)\Delta(s', C_1, C_2) \text{ for some } (s', C_1, C_2) \\ idle \in d_1(q) &\text{ for all } q \in Q \\ s'^{\underline{C_1}C_2}_{\underline{E_1}E_2} \in d_1(sE_1E_2) &\text{ iff } (s, E_1, E_2)\Delta(s', C_1, C_2) \end{aligned}$$

and analogously for agent 2 but E_2 and C_2 are underlined instead of E_1 and C_1, respectively. For agent 3 we have:

$$\begin{aligned} idle \in d_3(q) &\text{ for all } q \in Q \\ s'\underline{C_1}C_2 \in d_3(s'\underline{C_1C_2}) &\text{ for all } s'\underline{C_1C_2} \in Q_2 \\ test_1 \in d_3(sE_1E_2) &\text{ iff } E_1 = 0 \end{aligned}$$

and again analogously for agent 4 but E_2 and C_2 are underlined instead of E_1 and C_1, respectively. Moreover, $test_2 \in d_4(sE_1E_2)$ iff $E_2 = 0$, that is, the test action is only available if the counter which the agent is supposed to simulate is empty.

4. We abstain from giving the transition function o and refer to Fig. 4 where we call an action profile $(\underline{E_1}E_2, E_1'\underline{E_2'}, idle, idle)$ *invalid* if (i) $E_i \neq E_i'$ for some $i \in \{1,2\}$ or (ii) if there is no transition $((s,E_1,E_2),(s',C_1,C_2)) \in \Delta$. Similarly, we call an action profile $(s^{\underline{C_1}C_2}_{\underline{E_1}E_2}, s^{C_1'\underline{C_2'}}_{E_1'\underline{E_2'}}, idle, idle)$ *invalid* in state sE_1E_2 if (i) $C_i \neq C_i'$ or $E_i \neq E_i'$ for some $i \in \{1,2\}$ or (ii) if there is no transition $((s,E_1,E_2),(s',C_1,C_2)) \in \Delta$.
5. For $i \in \{1,2,3,4\}$, the actions' resource consumption/production is defined by the function t:

$$
\begin{aligned}
t(\underline{E_1}E_2, R) &= -E_1\\
t(E_1\underline{E_2}, R) &= -E_2\\
t(idle, R) &= 0\\
t(s^{\underline{C_1}C_2}_{\underline{E_1}E_2}, R) &= C_1 + E_1\\
t(s^{\underline{C_1}C_2}_{E_1\underline{E_2}}, R) &= C_2 + E_2\\
t(s\underline{C_1}C_2, R) &= C_1\\
t(sC_1\underline{C_2}, R) &= C_2\\
t(test_1, R) &= -1\\
t(test_2, R) &= -1
\end{aligned}
$$

Theorem 4. *Model checking* prRAL *over the class of* **iRBM***s is undecidable even in the case of single resource and two agents.*

Proof. We modify the model $\mathfrak{M}_2^{\mathcal{A}} = (\{1,2,3,4\}, Q, \Pi, \pi, Act, d, o, \{R\}, t)$ to a model $\widehat{\mathfrak{M}}_2^{\mathcal{A}} = (\{1,2\}, Q\setminus\{q_h\}, \Pi, \widehat{\pi}, \widehat{Act}, \widehat{d}, \widehat{o}, \{R\}, \widehat{t})$. Essentially, we merge agents $\{1,3\}$ and agents $\{2,4\}$ into agent 1 and 2 in the new model, respectively. The main encoding is shown in Fig. 5. We define $\widehat{d}_i(q)$ as $d_i(q)$ for $i \in \{1,2\}$ where it is additionally required that $test_1 \in \widehat{d}_1(sE_1E_2)$ iff $E_1 = 0$, and $test_2 \in \widehat{d}_2(sE_1E_2)$ iff $E_2 = 0$; coalition $\{1,2\}$ can now make all decisions. The action set $\widehat{Act}$ equals Act but all actions of type $s\underline{C_1}C_2$ and $sC_1\underline{C_2}$are removed. The transition function $\widehat{o}$ is obtained from o:

$$
\begin{aligned}
\widehat{o}(s, (\underline{E_1}E_2, E_1\underline{E_2})) &= sE_1E_2\\
\widehat{o}(s, (idle, \star)) &= q_{\circlearrowleft}\\
\widehat{o}(s, (\star, idle)) &= q_{\circlearrowleft}\\
o(sE_1E_2, (idle, \star)) &= q_{\circlearrowleft}\\
o(sE_1E_2, (\star, idle)) &= q_{\circlearrowleft}\\
\widehat{o}(q_{\circlearrowleft}, (idle, idle)) &= q_{\circlearrowleft}\\
\widehat{o}(sE_1E_2, (s^{'\underline{C_1}C_2}_{\underline{E_1}E_2}, s^{'C_1\underline{C_2}}_{E_1\underline{E_2}})) &= s\underline{C_1C_2}\\
\widehat{o}(sE_1E_2, (test_1, \star)) &= q_f\\
\widehat{o}(sE_1E_2, (\star, test_2)) &= q_f\\
\widehat{o}(q_f, (idle, idle)) &= q_f\\
\widehat{o}(s'\underline{C_1C_2}, (idle, idle)) &= s'
\end{aligned}
$$

Moreover, invalid action profiles executed in states s and sE_1E_2 also result in the loop state $q_{\circlearrowleft}$. Here, we call an action profile $(\underline{E_1}E_2, E_1'\underline{E_2'})$ *invalid* if (i) $E_i \neq E_i'$ for some $i \in \{1,2\}$ or (ii) if there is no transition $((s,E_1,E_2),(s',C_1,C_2)) \in \Delta$. Similarly, we call an action profile $(s^{\underline{C_1}C_2}_{\underline{E_1}E_2}, s^{C_1'\underline{C_2'}}_{E_1'\underline{E_2'}})$ *invalid* in state sE_1E_2 if (i) $C_i \neq C_i'$ or $E_i \neq E_i'$ for some $i \in \{1,2\}$ or (ii) if there is no transition $((s,E_1,E_2),(s',C_1,C_2)) \in \Delta$.

The cost function $\widehat{t}$ is defined as before restricted to the new action set. (States of type $s\underline{C_1C_2}$ are only kept due to compatibility reasons.) As in the related case, it is easy to see that each resource-extended path $\lambda = (q^i, \eta^i)_{i\in\mathbb{N}}$ in $\mathfrak{M}_2^{\mathcal{A}}$ that does not contain state q_h corresponds to a path $\lambda = (q^i, \widehat{\eta}^i)_{i\in\mathbb{N}}$ in $\widehat{\mathfrak{M}}_2^{\mathcal{A}}$ with $\widehat{\eta}^i(1,R) = \eta^i(1,R)$ and $\widehat{\eta}^i(2,R) = \eta^i(2,R)$ for all i. By the analogue of Proposition 1(c) given in Section 4.2.2, the zero-test simulated in the test states can equivalently be defined with respect to agent 1's and 2's resource endowments. It holds that $\mathfrak{M}_2^{\mathcal{A}}, q_{\mathbf{init}}, \bar{0} \models \langle\langle\{1,2,3,4\}\rangle\rangle^{\bar{0}}((\neg\langle\langle\{1,2,3,4\}\rangle\rangle^{\downarrow}\mathbf{X}\,\mathsf{fail})\mathbf{U}\,\mathsf{halt})$ if, and only if, $\widehat{\mathfrak{M}}_2^{\mathcal{A}}, q_{\mathbf{init}}, \bar{0} \models \langle\langle\{1,2\}\rangle\rangle^{\bar{0}}((\neg\langle\langle\{1,2\}\rangle\rangle^{\downarrow}\mathbf{X}\,\mathsf{fail})\mathbf{U}\,\mathsf{halt})$. □

Appendix B. Proof from Section 5

Lemma 8. *Algorithm 2 is correct, that is,* STRATEGY(n, ϕ) *returns true iff* $\exists e' \in compatible(v(n)) : s(n), e' \models \phi$.

Proof. The proof is by induction on the structure of the formulae.
Base case:
Case $\phi = p$ for some $p \in \Pi$: STRATEGY(n, p) returns true iff $s(n) \models p$ (by lines 2–3 in Algorithm 2) iff $s(n), e' \models p$ for any $e' \in compatible(v(n))$ (by the semantics of RAL). Obviously, $compatible(v(n)) \neq \emptyset$.
Case $\phi = \neg p$ for some $p \in \Pi$: Similarly, STRATEGY$(n, \neg p)$ returns true iff $s(n) \not\models p$ (by lines 4–5 in Algorithm 2) iff $s(n), e' \not\models p$ for any $e' \in compatible(v(n))$. Again, $compatible(v(n)) \neq \emptyset$.
Induction step: The proof is done for each case of pprRAL formulae.
Case $\phi = \phi_1 \wedge \phi_2$: STRATEGY$(n, \phi)$ returns true iff STRATEGY(n, ϕ_1) and STRATEGY(n, ϕ_2) return true (by lines 6–7), iff $\exists e'_1 \in compatible(v(n)) : s(n), e'_1 \models \phi_1$ and $\exists e'_2 \in compatible(v(n)) : s(n), e'_2 \models \phi_2$ (by induction hypothesis), iff $s(n), e' \models \phi_1 \wedge \phi_2$ (by the semantics of RAL) where $e' = \overline{\max}\{e'_1, e'_2\} \in compatible(v(n))$ where $\overline{\max} V = v$ denotes the pointwise maximum from a finite set V of endowments, i.e., $v(i, r) = \max\{e'(i, r) \mid e' \in V\}$ for all $i \in \mathbb{A}gt$ and $r \in Res$ (note that this is straightforward by induction on the structure of formulae in pprRAL that $s, v \models \varphi$ implies $s, e' \models \varphi$ for all states s and $e' \geq v$ since pprRAL contains only positive formulae).
Case $\phi = \phi_1 \vee \phi_2$: STRATEGY$(n, \phi)$ returns true iff STRATEGY(n, ϕ_1) or STRATEGY(n, ϕ_2) return true (by lines 8–9), iff $\exists e'_1 \in compatible(v(n)) : s(n), e'_1 \models \phi_1$ or $\exists e'_2 \in compatible(v(n)) : s(n), e'_2 \models \phi_2$ (by induction hypothesis) iff $s(n), e' \models \phi_1 \vee \phi_2$ (by the semantics of RAL) for some $e' \in \{v_1, v_2\} \subseteq compatible(v(n))$.
Case $\phi = \langle\langle A\rangle\rangle^{\downarrow}\mathbf{X}\psi$: STRATEGY$(n, \phi)$ returns true iff X-STRATEGY(n', ϕ) returns true (by lines 10–11) where $n' = node_0(s(n), e(n), v(n), A)$, iff there exists $\alpha' \in ActA$ (according to lines 4–5 of Algorithm 3) such that for every $\alpha \in ActAgt$ (line 8 of Algorithm 3) STRATEGY(n'', ψ) returns true where $n'' = node(n', s_\alpha, \alpha, A)$ and $s_\alpha = o(s(n'), \alpha)$ (recall that $v_a(n'') = v_a(n) - \mathsf{cons}(\alpha_a) + \mathsf{prod}(\alpha_a)$ for $a \in A$, $v_a(n'') = v_a(n)$ for $a \notin A$ and $s(n'') = s_\alpha$), iff for every $\alpha \in ActAgt$, there exists $v_\alpha \in compatible(v(n) - \mathsf{cons}(\alpha) + \mathsf{prod}(\alpha))$ such that $s_\alpha, v_\alpha \models \psi$ (by induction hypothesis), iff there exists $e' \in compatible(v(n) - \mathsf{cons}(\alpha') + \mathsf{prod}(\alpha'))$ such that for every $\alpha \in ActAgt$ $s_\alpha, e' \models \psi$, where $e' = \overline{\max}\{v_\alpha \mid \alpha \in ActAgt\} \in compatible(v(n) - \mathsf{cons}(\alpha') + \mathsf{prod}(\alpha'))$, iff $s, e' + \mathsf{cons}(\alpha') - \mathsf{prod}(\alpha') \models \langle\langle A\rangle\rangle^{\downarrow}\mathbf{X}\psi$ where $e' + \mathsf{cons}(\alpha') - \mathsf{prod}(\alpha') \in compatible(v(n))$.

The case for $\langle\langle A\rangle\rangle^{\xi}\mathbf{X}\psi$ is similar to the above case, hence omitted here.
Case $\phi = \langle\langle A\rangle\rangle^{\downarrow}\psi_1\mathbf{U}\psi_2$: STRATEGY$(n, \phi)$ returns true iff U-STRATEGY(n_0, ϕ) returns true (by lines 14–15) where $n_0 = node_0(s(n), e(n), v(n), A)$.
($\Rightarrow$): Let T denote the search tree rooted at n_0 when U-STRATEGY(n_0, ϕ) returns true. For the purpose of this proof, we assume that each interior node n in T has an additional function $a(n)$ which returns the action tuple of the proponent at $s(n)$; and the function $node(n, s, \alpha, A)$ also assigns $a(n) = \alpha$.

For every leaf n of T, we have that STRATEGY(n, ψ_2) returns true according to lines 8–9 of Algorithm 4. By the induction hypothesis, we also have $s(n), v_n \models \psi_2$ for some $v_n \in compatible(v(n))$. Similarly, for every interior node n of T, STRATEGY(n, ψ_1) returns true according to lines 10–11 of Algorithm 4 and, hence, $s(n), v_n \models \psi_1$ for some $v_n \in compatible(v(n))$. We first update the value of $e(n)$ for every node in T so that resource availability is enough to satisfy ψ_1 at every interior node and ψ_2 at every leaf node. The update is carried out from the leaves to the root of T as follows:

- For a leaf n, $e(n) := \overline{\max}\{v_n, e(n)\}$;
- For an interior node n with k children $n_1, \ldots, n_k$, if $e(n_i)$ has been updated for all $i \in \{1, \ldots, k\}$, then $e(n) := \overline{\max}\{v_n, e(n), e(n_1) + \mathsf{cons}(a(n)), \ldots, e(n_k) + \mathsf{cons}(a(n))\}$.

Let s_T denote the strategy for A where for each node $n \in T$, with $p(n) = n_0 \ldots n_k$, $s_T(s(n_0) \ldots s(n_k)s(n)) = (a(n))_A$. However, this strategy may not be executable from n_0 if, as a result of the initial resource availability $e(n_0)$, there is some node n in T such that $e(n)(i, r) < 0$ for some resource r and agent i. Note that whenever $e(n)(i, r) < 0$, we have $v(n)(i, r) = arb$ according to lines 6, 7 and 12 of Algorithm 4. If $v(n_0)(i, r) = arb$, we can simply increase the value of $e(n_0)(i, r)$ to compensate for the lack of resources above. In particular, we increase $e(n_0)(i, r)$ to $e(n_0)(i, r) - e(n)(i, r)$, then recalculate the value of $e(n')(i, r)$ for every node n' in T. Then, $e(n)(i, r)$ becomes 0. Obviously, this step only removes negative values and can be repeated until no further negative values can be removed. This means for any $e(n)(i, r) < 0$ we have $v(n_0)(i, r) \neq arb$. Then, there must be a loop within the path $p(n)$ (according to lines 6–7 of Algorithm 4) that strictly increases resource r for agent i. To increase $e(n)(i, r)$ to a positive value, we need to determine the number of times the loop should be performed. In particular, there must be a node $n_1 \in p(n)$ such that $v(n_1)(i, r)$ is assigned arb by the statement in lines 6–7 of Algorithm 4. Let n_2 be the node n' in line 6. We denote n_1 by $stop_r(n)$ and n_2 by $start_r(n)$. Let $T(\hat{n})$ denote the subtree of T rooted in $\hat{n}$ for every $\hat{n}$ in T. For a resource r, if there is a node n with $e(n)(i, r) < 0$ for some i we extend T until $e(n)(i, r) \geq 0$ by repeating the branch between $start_r(n)$ and $end_r(n)$ finitely many times. There are two sub-cases to consider:

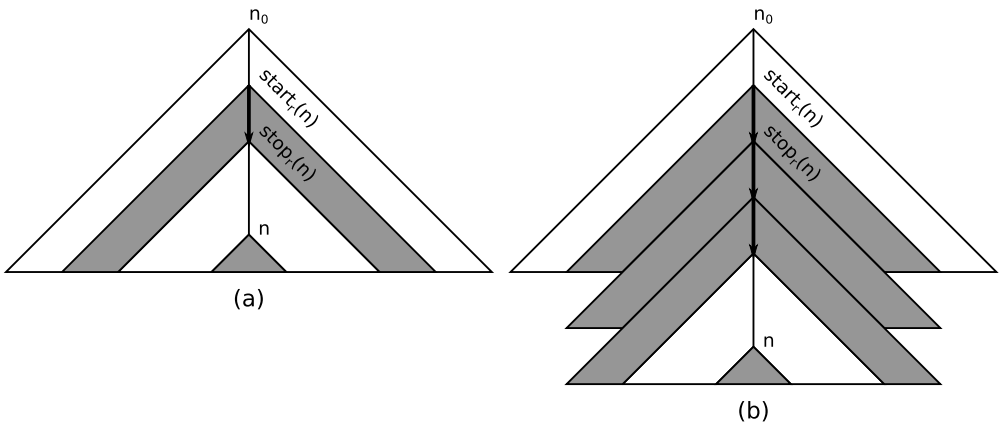

Fig. 6. n is the only node with $e(n)(i,r) < 0$ in $T(start_r(n))$.

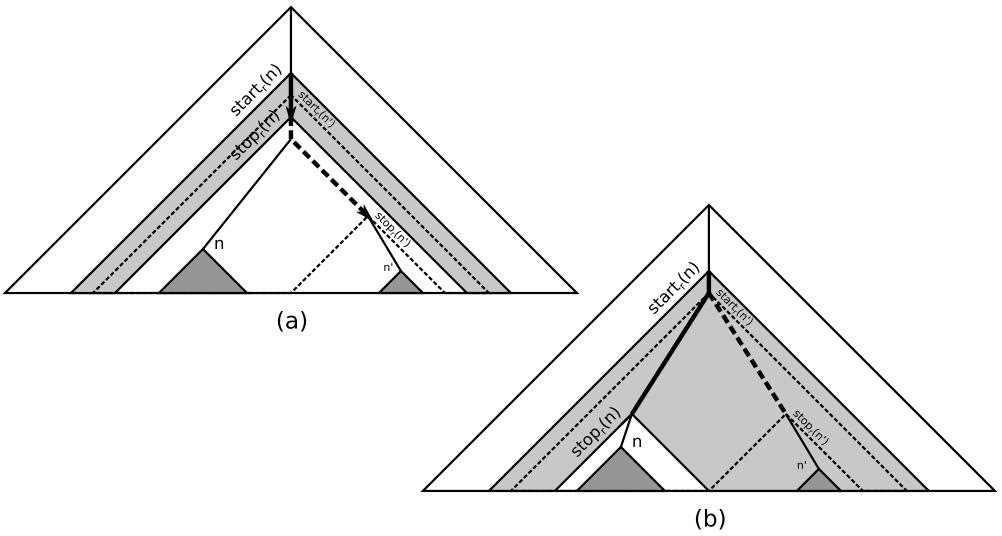

Fig. 7. $T(start_r(n))$ also have n' with $e(n')(a,r) < 0$.

- Sub-case 1: n is the only node in $T(start_r(n))$ with $e(n)(i,r) < 0$ as depicted by Fig. 6(a).
  As the path from $start_r(n)$ to $stop_r(n)$ increases r for agent i by an amount of $e(stop_r(n))(i,r) - e(start_r(n))(i,r)$, it is necessary to repeat this path $\lceil \frac{|e(n)(i,r)|}{e(stop_r(n))(i,r) - e(start_r(n))(i,r)} \rceil$ times, as depicted in Fig. 6(b).
- Sub-case 2: n is not the only node in $T(start_r(n))$ with $e(n)(i,r) < 0$. Other nodes n' are either in the subtree $T(stop_r(n))$ (as depicted by Fig. 7(a)) or in the subtree of $T(start_r(n))$ but not $T(stop_r(n))$ (as depicted by Fig. 7(b)). Without loss of generality, we assume that $start_r(n)$ is the ancestor of $start_r(n')$ for any of such n'.
  Again, as the path from $start_r(n)$ to $end_r(n)$ increases r for agent i by an amount of $e(end_r(n))(i,r) - e(start_r(n))(i,r)$, it is necessary to repeat this path $k = \lceil \frac{|e(n)(i,r)|}{e(end_r(n))(i,r) - e(start_r(n))(i,r)} \rceil$ times, as depicted in Fig. 8(a). Note that this repetition will also repeat nodes n' which are in the subtree of $T(start_r(n))$ but not in $T(stop_r(n))$ and have $e(n')(i,r) < 0$ as $n'_1, \ldots, n'_k$ depicted in Fig. 8(b).

Let T_1 be the obtained tree. Then, the number of nodes n'' in $T_1(n_{k-1})$ with $e(n'')(i,r) < 0$ is strictly less than that in $T(start_r(n))$. Therefore, we can reapply the above construction to obtain a tree T_2 where all nodes n'' in $T_2(n_{k-1})$ have $e(n'')(i,r) \geq 0$. These include node n' as depicted in Fig. 8a and n'_k as depicted in Fig. 8b. Then, we further apply step by step the above construction for nodes $n'_{k-1}, \ldots, n'_1$ and n' in Fig. 8b. Finally, we obtain a tree T_3 where all nodes n'' have $e(n'')(i,r) \geq 0$. This construction can be repeated for other resources $r' \neq r$ and agents $i' \neq i$. Finally, we obtain a tree T_4 where for all nodes n in T_4, $e(n)(i,r) \geq 0$ for all r and i and s_{T_4} is a strategy satisfying ϕ at $s(n_0)$ where n_0 is the root of T_4. In other words, we have $s(n_0), e(n_0) \models \phi$ where it is obvious that $e(n_0) \in compatible(v(n_0))$. Since $n_0 = node_0(s(n), e(n), v(n), A)$, $s(n) = s(n_0)$ and $v(n) = v(n_0)$; therefore $s(n), e(n_0) \models \phi$ where $e(n_0) \in compatible(v(n))$.
($\Leftarrow$): Assume that $s(n), e' \models \phi$ for some $e' \in compatible(v(n))$, then there exists a strategy s_A such that for all $\lambda \in out(q, e', s_A, A) : \exists i_\lambda \geq 0 : \lambda|_Q[i_\lambda], \lambda|_{\mathsf{En}}[i_\lambda] \models \psi_2$ and $\forall 0 \leq j < i_\lambda : \lambda|_Q[j], \lambda|_{\mathsf{En}}[j] \models \psi_1$. We shall now prove that

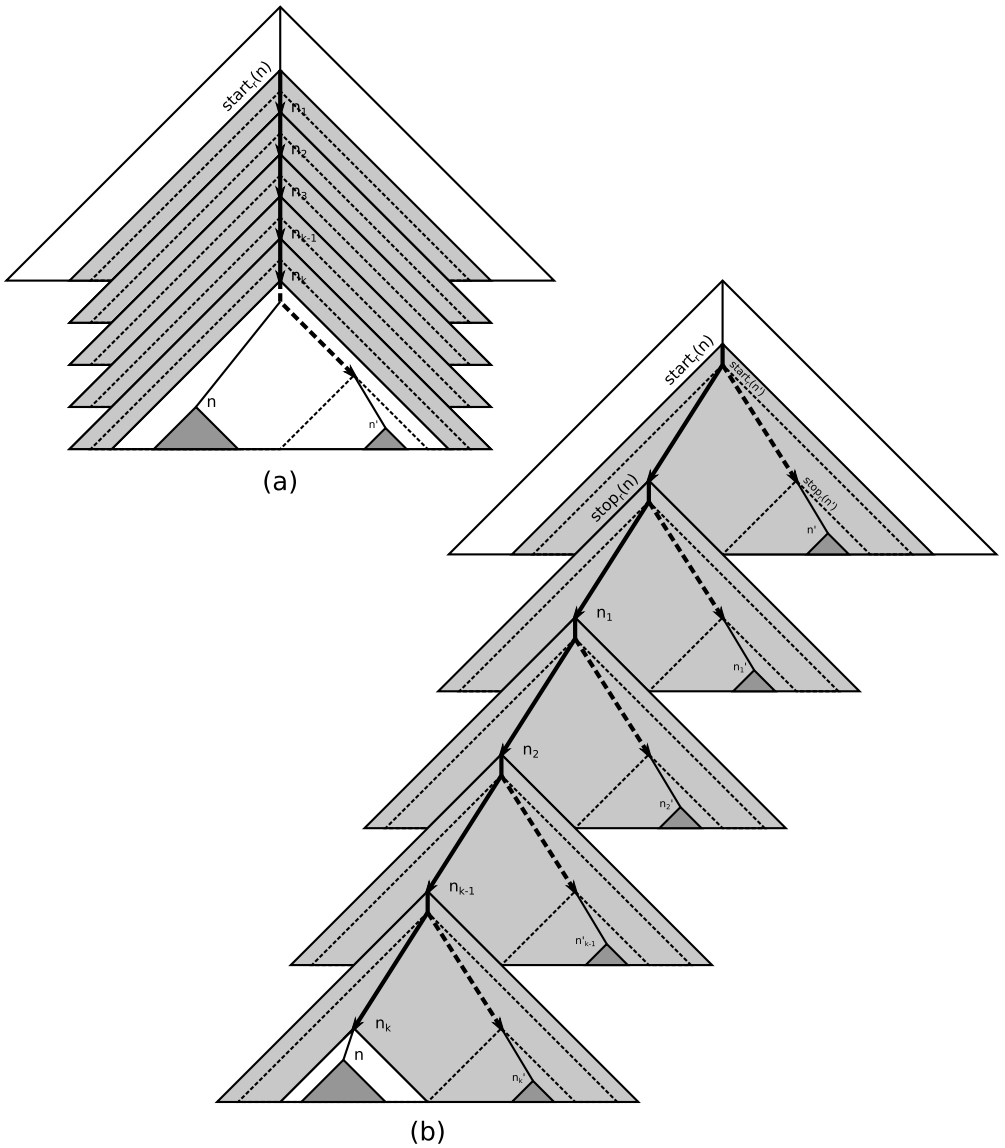

Fig. 8. Repeating $T(start_r(n))$.

STRATEGY(n, ϕ) returns true, i.e., equivalently U-STRATEGY$(node_0(s(n), e(n), v(n), A), \phi)$ returns true. Let $T = (V, E)$ be the tree induced by all runs $\lambda[0, i_\lambda]$ for $\lambda \in out(q, e', s_A, A)$, i.e., $V = \{\lambda[0, i] \mid \lambda \in out(q, e', s_A, A), i \leq i_\lambda\}$ and $E = \{(\lambda[0, i], \lambda[0, i+1]) \mid \lambda \in out(s, e', s_A, A), i < i_\lambda\}$. We first attach to each node $\lambda[0, i]$ of T an element $v(\lambda[0, i])$ where

$$v(\lambda[0,i])(a,r) = \begin{cases} \lambda|_{\mathsf{En}}[i](a,r) & \text{if } v(n)(a,r) \neq arb \\ arb & \text{otherwise} \end{cases}$$

In the following, we show how to convert T into a search tree which shows that U-STRATEGY$(node_0(s(n), e(n), v(n), A), \phi)$ returns true. Note that T must be finite and each edge in E corresponds to a join action of all agents.

Initially, let $T_0 = T$, then T_{l+1} is constructed from T_l as follows.

Case 1a If there is a node $\lambda[0, k]$ in T_l such that $\exists j < k : \lambda|_Q[j] = \lambda|_Q[k] \wedge v(\lambda[0, j]) \geq v(\lambda[0, k])$, then T_{l+1} is constructed from T_l by replacing the subtree $T_l(\lambda[j])$ by $T_l(\lambda[k])$, updating the values for $\lambda[k']|_{\mathsf{En}}$ and $v(\lambda[0, k'])$ for all $k' \geq k$ in the subtree $T_l(\lambda[k])$ according to the values $\lambda[j]|_{\mathsf{En}}$ and $v(\lambda[0, j])$ and the costs of actions in $T_l(\lambda[k])$.

Case 1b If there is a node $\lambda[0, k]$ in T_l such that $\exists j < k : \lambda|_Q[j] = \lambda|_Q[k] \wedge v(\lambda[0, j]) \leq v(\lambda[0, k]) \wedge v(\lambda[0, j]) \neq v(\lambda[0, k])$, then T_{l+1} is constructed from T_l by replacing the value $v(\lambda[0, k'])$ for all nodes $\lambda[0, k']$ in the subtree $T_l(\lambda[k])$ as follows

$$v(\lambda[0,k'])(a,r) = \begin{cases} v(\lambda[0,k'])(a,r) & \text{if } v(\lambda[0,j])(a,r) = v(\lambda[0,k])(a,r) \\ arb & \text{if } v(\lambda[0,j])(a,r) < v(\lambda[0,k])(a,r) \end{cases}$$

Case 2 Otherwise, $T_{l+1} = T_l$, i.e., no more change.

The construction stops when $T_{l+1} = T_l$. Let the resulting tree $T_{l+1} = T'$. For each node $\lambda[0,i]$ in T', we define a node $n_{\lambda[0,i]}$ where: $s(n_{\lambda[0,i]}) = \lambda|_Q[i]$, $p(n_{\lambda[0,i]}) = [\lambda|_Q[0] \cdot \ldots \cdot \lambda|_Q[i-1]]$, $c(n_{\lambda[0,i]}) = A$ and $e(n_{\lambda[0,i]}) = \lambda|_{\mathsf{En}}[i]$ and $v(n_{\lambda[0,i]}) = v(\lambda[0,i])$. In the following, we show by induction on the length of $T'(\lambda[0,i])$ that U-STRATEGY$(n_{\lambda[0,i]}, \phi)$ returns true.

Base case: Assume that $\lambda[0,i]$ is a leaf of T', then it is a leaf from T. Then, the condition of the **if** statement in lines 8–9 of Algorithm 4 is true, therefore, U-STRATEGY$(n_{\lambda[0,i]}, \phi)$ returns true.

Induction step: Assume that $\lambda[0,i]$ is not a leaf of T'. Then the condition of the **if** statement (lines 2–3) is false, since $s(n) \models alt(\phi)$ (where n is from the input of this Lemma). The condition of the next **if** statement (lines 4–5) is also false, since otherwise T' can be cut further by (Case 1a). The condition of the third or fourth **if** statement (lines 8–11) is false since otherwise $\lambda[0,i]$ must have been a leaf of T'. Therefore, the algorithm must enter the second **for** loop. For $\alpha = s_A(\lambda|_Q[0,i]) \in d_A(\lambda[i])$ we have that, for every $s' \in out(\lambda|_Q[i], \alpha)$ with $n' = node(n_{\lambda[0,i]}, s', \alpha, A)$, there must be $\lambda'[0, i+1]$ in T' such that $n' = n_{\lambda'[0,i+1]}$. By the induction hypothesis, U-STRATEGY$(n_{\lambda'[0,i+1]}, \phi)$ returns true. Thus, U-STRATEGY$(n_{\lambda[0,i]}, \phi)$ also returns true.

Obviously, U-STRATEGY$(node_0(s(n), e(n), v(n), A), \phi)$ returns true since $n_{\lambda[0]} = node_0(s(n), e(n), v(n), A)$.

The above proof can be adapted to the case $\phi = \langle\langle A\rangle\rangle^{\zeta}\psi_1 \mathbf{U} \psi_2$ by exchanging the role of $v(n)$ and ζ.

Case $\phi = \langle\langle A\rangle\rangle^{\downarrow}\mathbf{G}\psi$: STRATEGY$(n, \phi)$ returns true iff G-STRATEGY(n_0, ϕ) returns true (by lines 18–19) where $n_0 = node_0(s(n), e(n), v(n), A)$.

($\Rightarrow$): Let T denote the search tree rooted at n_0 when G-STRATEGY(n_0, ϕ) returns true.

For every node n of T, we have that STRATEGY(n, ψ) returns true according to lines 10–11 of Algorithm 5. By induction hypothesis, we also have $s(n), v_n \models \psi$ for some $v_n \in compatible(v(n))$. We first update the value of $e(n)$ for every node in T so that resource availability is enough to satisfy ψ at every node. The update is carried out from the leaves to the root of T as follows:

- For a leaf n, $e(n) := \overline{\max}\{v_n, e(n)\}$;
- For an interior node n with k children $n_1, \ldots, n_k$, if $e(n_i)$ has been updated for all $i \in \{1, \ldots, k\}$, then $e(n) := \overline{\max}\{v_n, e(n), e(n_1) + \mathsf{cons}(a(n)), \ldots, e(n_k) + \mathsf{cons}(a(n))\}$.

Let s_T denote the strategy for A where for each node $n \in T$, with $p(n) = n_0 \ldots n_k$, $s_T(s(n_0) \ldots s(n_k)s(n)) = (a(n))_A$. However, this strategy may not be executable from n_0 if there is some node n in T such that $e(n)(i,r) < 0$ for some resource r and agent i. We can repeat the tree expansions in the previous case to eliminate all such nodes. Let the obtained tree be T_1. By lines 12–13, for all leaves n' of T_1, we have that there exists $n'' \in p(n')$ such that $e_A(n'') \leq e_A(n')$. Then, we construct an infinite tree from T_1 as follows:

- Given T_i, we construct T_{i+1} by replacing all leaves n' of T_i by the tree $T_1(n'')$.
- $T' = \lim_{i\to\infty} T_i$.

Then, the strategy $s_{T'}$ from T' obviously satisfies ϕ. In other words, we have $s(n_0), e(n_0) \models \phi$ where it is obvious that $e(n_0) \in compatible(v(n))$. Again, since $n_0 = node_0(s(n), e(n), v(n), A)$, $s(n) = s(n_0)$ and $v(n) = v(n_0)$; therefore $s(n), e(n_0) \models \phi$ where $e(n_0) \in compatible(v(n))$.

($\Leftarrow$): Assume that $s(n), e' \models \phi$ for some $e' \in compatible(v(n))$, then there exists a strategy s_A such that for all $\lambda \in out(q, e', s_A, A)$ and $i \geq 0 : \lambda|_Q[i], \lambda|_{\mathsf{En}}[i] \models \psi$. We shall now prove that STRATEGY(n, ϕ) returns true, i.e., equivalently G-STRATEGY$(node_0(s(n), e(n), v(n), A), \phi)$ returns true. Let $T = (V, E)$ be the infinite tree induced by all runs $\lambda \in out(q, e', s_A, A)$, i.e., $V = \{\lambda[0,i] \mid \lambda \in out(q, e', s_A, A), i \geq 0\}$ and $E = \{(\lambda[0,i], \lambda[0,i+1]) \mid \lambda \in out(s, \eta, s_A, A)\}$. We also attach to each node $\lambda[0,i]$ of T an element $v(\lambda[0,i])$ where

$$v(\lambda[0,i])(a,r) = \begin{cases} \lambda|_{\mathsf{En}}[i](a,r) & \text{if } v(n)(a,r) \neq arb \\ arb & \text{otherwise} \end{cases}$$

Then, in the following, we cut T into a finite search tree which shows that G-STRATEGY$(node_0(s(n), e(n), v(n), A), \phi)$ returns true. Note that each edge in E corresponds to a join action of all agents. First, we repeatedly apply the following rule to prune the tree:

Case 3 If there is a node $\lambda[0,k]$ in T_l such that $\exists 0 \leq j < k : \lambda|_Q[j] = \lambda|_Q[k] \wedge \lambda|_{\mathsf{En}}[j] \leq \lambda|_{\mathsf{En}}[k]$, then T_{l+1} is constructed from T_l by replacing in the subtree $T_l(\lambda[k])$ by the node $\lambda[k]$.

This step must yield a finite tree T' since for every infinite path in the original tree, there must be a state appearing infinitely often and the corresponding endowments must be non-decreasing. These paths then are always cut by (Case 3).

Then, let $T_0 = T'$, then T_{l+1} is constructed from T_l as follows.

Case 4 If there is a node $\lambda[0,k]$ in T_l such that $\exists j < k : \lambda|_Q[j] = \lambda|_Q[k] \wedge \lambda|_{\mathsf{En}}[j] \geq \lambda|_{\mathsf{En}}[k]$ and $\lambda|_{\mathsf{En}}[j] \neq \lambda|_{\mathsf{En}}[k]$, then T_{l+1} is constructed from T_l by replacing the subtree $T_l(\lambda[j])$ by $T_l(\lambda[k])$, updating the values for $\lambda[k']|_{\mathsf{En}}$ and $v(\lambda[0,k'])$ for all $k' \geq k$ in the subtree $T_l(\lambda[k])$ according to the values $\lambda[j]|_{\mathsf{En}}$ and $v(\lambda[0,j])$ and the costs of actions in $T_l(\lambda[k])$.

Case 5 If there is a node $\lambda[0,k]$ in T_l such that $\exists j < k : \lambda|_Q[j] = \lambda|_Q[k]$ and $(v(\lambda[0,j])(a,r) = v(\lambda[0,k])(a,r) = arb$ or $\lambda|_{\mathsf{En}}[j](a,r) = \lambda|_{\mathsf{En}}[k](a,r)$ for all $a \in A, r \in Res$) and $\lambda|_{\mathsf{En}}[j](a,r) > \lambda|_{\mathsf{En}}[k](a,r)$ for some $a \in A, r \in Res$, then T_{l+1} is constructed from T_l by replacing the subtree $T_l(\lambda[j])$ by $T_l(\lambda[k])$, updating the values for $\lambda[k']|_{\mathsf{En}}$ and $v(\lambda[0,k'])$ for all $k' \geq k$ in the subtree $T_l(\lambda[k])$ according to the values $\lambda[j]|_{\mathsf{En}}$ and $v(\lambda[0,j])$ and the costs of actions in $T_l(\lambda[k])$.

Case 6 If there is a node $\lambda[0,k]$ in T_l such that $\exists j < k : \lambda|_Q[j] = \lambda|_Q[k] \wedge v(\lambda[0,j]) \leq v(\lambda[0,k]) \wedge v(\lambda[0,j]) \neq v(\lambda[0,k])$, then T_{l+1} is constructed from T_l by replacing the value $v(\lambda[0,k'])$ for all nodes $\lambda[0,k']$ in the subtree $T_l(\lambda[k])$ as follows

$$v(\lambda[0,k'])(a,r) = \begin{cases} v(\lambda[0,k'])(a,r) & \text{if } v(\lambda[0,j])(a,r) = v(\lambda[0,k])(a,r) \\ arb & \text{if } v(\lambda[0,j])(a,r) < v(\lambda[0,k])(a,r) \end{cases}$$

Case 7 Otherwise, $T_{l+1} = T_l$, i.e., no more change.

The construction stops when $T_{l+1} = T_l$. Let the resulting tree $T_{l+1} = T''$.

For each node $\lambda[0,i]$ in T'', we define a node $n_{\lambda[0,i]}$ where: $s(n_{\lambda[0,i]}) = \lambda|_Q[i]$, $p(n_{\lambda[0,i]}) = [\lambda|_Q[0] \cdot \ldots \cdot \lambda|_Q[i-1]]$, $c(n_{\lambda[0,i]}) = A$ and $e(n_{\lambda[0,i]}) = \lambda|_{\mathsf{En}}[i]$ and $v(n_{\lambda[0,i]}) = v(\lambda[0,i])$. In the following, we show by induction on the height of $T''(\lambda[0,i])$ that G-STRATEGY$(n_{\lambda[0,i]}, \phi)$ returns true.

Base case: Assume that $\lambda[0,i]$ is a leaf of T'', then it is a leaf from T'' and is the result of applying Case 3. Then, the condition of the **if** statement in lines 12–13 of Algorithm 5 is true, therefore, G-STRATEGY$(n_{\lambda[0,i]}, \phi)$ returns true.

Induction step: Assume that $\lambda[0,i]$ is not a leaf of T''. Then the condition of the **if** statement (lines 2–3) is false, since $s(n) \models alt(\phi)$ (where n is from the input of this Lemma). The condition of the next **if** statement (lines 4–5) is also false, since otherwise T'' can be cut further by (Case 4). The condition of the third or fourth **if** statement (lines 6–7) is false since otherwise T'' can be cut further by (Case 5). Therefore, the algorithm must enter the second **for** loop. For $\alpha = s_A(\lambda|_Q[0,i]) \in d_A(\lambda[i])$ we have that, for every $s' \in out(\lambda|_Q[i], \alpha)$ with $n' = node(n_{\lambda[0,i]}, s', \alpha, A)$, there must be $\lambda'[0, i+1]$ in T' such that $n' = n_{\lambda'[0,i+1]}$. By the induction hypothesis, G-STRATEGY$(n_{\lambda'[0,i+1]}, \phi)$ returns true. Thus, G-STRATEGY$(n_{\lambda[0,i]}, \phi)$ also returns true.

Obviously, G-STRATEGY$(node_0(s(n), e(n), v(n), A), \phi)$ returns true since $n_{\lambda[0]} = node_0(s(n), e(n), v(n), A)$.

The above proof can be adapted to the case $\phi = \langle\langle A \rangle\rangle^{\zeta} \mathbf{G}\psi$ by exchanging the role of $v(n)$ and ζ. □

References

[1] N. Alechina, B. Logan, H.N. Nguyen, A. Rakib, A logic for coalitions with bounded resources, in: C. Boutilier (Ed.), Proceedings of the Twenty First International Joint Conference on Artificial Intelligence, IJCAI 2009, vol. 2, AAAI Press, 2009, pp. 659–664.

[2] N. Bulling, B. Farwer, On the (un-)decidability of model checking resource-bounded agents, in: H. Coelho, R. Studer, M. Wooldridge (Eds.), Proceedings of the 19th European Conference on Artificial Intelligence, ECAI 2010, IOS Press, 2010, pp. 567–572.

[3] N. Alechina, B. Logan, H.N. Nguyen, A. Rakib, Resource-bounded alternating-time temporal logic, in: W. van der Hoek, G. Kaminka, Y. Lespérance, M. Luck, S. Sen (Eds.), Proceedings of the Ninth International Conference on Autonomous Agents and Multiagent Systems, AAMAS 2010, IFAAMAS, 2010, pp. 481–488.

[4] D. Della Monica, M. Napoli, M. Parente, Model checking coalitional games in shortage resource scenarios, in: G. Puppis, T. Villa (Eds.), Proceedings of the Fourth International Symposium on Games, Automata, Logics and Formal Verification, GandALF 2013, in: Electronic Proceedings in Theoretical Computer Science, vol. 119, 2013, pp. 240–255.

[5] N. Bulling, V. Goranko, How to be both rich and happy: combining quantitative and qualitative strategic reasoning about multi-player games (extended abstract), in: F. Mogavero, A. Murano, M.Y. Vardi (Eds.), Proceedings of the 1st International Workshop on Strategic Reasoning, SR 2013, in: Electronic Proceedings in Theoretical Computer Science, vol. 112, 2013, pp. 33–41.

[6] N. Alechina, B. Logan, H.N. Nguyen, F. Raimondi, Decidable model-checking for a resource logic with production of resources, in: T. Schaub, G. Friedrich, B. O'Sullivan (Eds.), Proceedings of the 21st European Conference on Artificial Intelligence, ECAI-2014, IOS Press, 2014, pp. 9–14.

[7] N. Alechina, N. Bulling, B. Logan, H.N. Nguyen, On the boundary of (un)decidability: decidable model-checking for a fragment of resource agent logic, in: Q. Yang (Ed.), Proceedings of the 24th International Joint Conference on Artificial Intelligence, IJCAI 2015, AAAI Press, 2015, pp. 1494–1501.

[8] N. Alechina, B. Logan, H.N. Nguyen, F. Raimondi, Model-checking for resource-bounded ATL with production and consumption of resources, J. Comput. Syst. Sci. (2017), in press.

[9] N. Alechina, B. Logan, H.N. Nguyen, F. Raimondi, Model-Checking for Resource-Bounded ATL with Production and Consumption of Resources, Tech. rep., 2015, arXiv:1504.06766.

[10] M. Ghallab, D.S. Nau, P. Traverso, Automated Planning: Theory and Practice, Morgan Kaufmann, 2004.

[11] M. Pauly, A modal logic for coalitional power in games, J. Log. Comput. 12 (1) (2002) 149–166.

[12] N. Alechina, B. Logan, H.N. Nguyen, A. Rakib, Logic for coalitions with bounded resources, J. Log. Comput. 21 (6) (2011) 907–937.

[13] M.Z. Kwiatkowska, G. Norman, D. Parker, Prism 4.0: verification of probabilistic real-time systems, in: Proceedings of the 23rd International Conference on Computer Aided Verification, CAV 2011, in: Lecture Notes in Computer Science, vol. 6806, Springer, 2011, pp. 585–591.

[14] N. Bulling, B. Farwer, Expressing properties of resource-bounded systems: the logics RBTL and RBTL*, in: Computational Logic in Multi-Agent Systems – 10th International Workshop, CLIMA X, Revised Selected and Invited Papers, in: Lecture Notes in Computer Science, vol. 6214, 2010, pp. 22–45.

[15] D. Della Monica, M. Napoli, M. Parente, On a logic for coalitional games with priced-resource agents, Electron. Notes Theor. Comput. Sci. 278 (2011) 215–228.
[16] D. Della Monica, G. Lenzi, On a priced resource-bounded alternating μ-calculus, in: J. Filipe, A.L.N. Fred (Eds.), Proceedings of the 4th International Conference on Agents and Artificial Intelligence, ICAART 2012, SciTePress, 2012, pp. 222–227.
[17] J. Daniel, A. Cimatti, A. Griggio, S. Tonetta, S. Mover, Infinite-state liveness-to-safety via implicit abstraction and well-founded relations, in: S. Chaudhuri, A. Farzan (Eds.), Proceedings of the 28th International Conference on Computer Aided Verification, CAV 2016, in: Lecture Notes in Computer Science, vol. 9779, Springer, 2016, pp. 271–291.
[18] B. Cook, H. Khlaaf, N. Piterman, On automation of CTL* verification for infinite-state systems, in: D. Kroening, C.S. Pasareanu (Eds.), Proceedings of the 27th International Conference on Computer Aided Verification, CAV 2015, in: Lecture Notes in Computer Science, vol. 9206, Springer, 2015, pp. 13–29.
[19] C. Urban, A. Miné, Proving guarantee and recurrence temporal properties by abstract interpretation, in: D. D'Souza, A. Lal, K.G. Larsen (Eds.), Proceedings of the 16th International Conference on Verification, Model Checking, and Abstract Interpretation, VMCAI 2015, in: Lecture Notes in Computer Science, vol. 8931, Springer, 2014, pp. 190–208.
[20] T.A. Beyene, M. Brockschmidt, A. Rybalchenko, CTL+FO verification as constraint solving, in: N. Rungta, O. Tkachuk (Eds.), Proceedings of the International Symposium on Model Checking of Software, SPIN 2014, ACM, 2014, pp. 101–104.
[21] B. Cook, E. Koskinen, M.Y. Vardi, Temporal property verification as a program analysis task – extended version, Form. Methods Syst. Des. 41 (1) (2012) 66–82.
[22] M. Fisher, L.A. Dennis, M.P. Webster, Verifying autonomous systems, Commun. ACM 56 (9) (2013) 84–93.
[23] P. Kouvaros, A. Lomuscio, Parameterised verification for multi-agent systems, Artif. Intell. 234 (2016) 152–189.
[24] J. Esparza, Decidability and complexity of Petri net problems – an introduction, in: Lectures on Petri Nets I: Basic Models, in: Lecture Notes in Computer Science, vol. 1491, Springer-Verlag, 1998, pp. 374–428.
[25] M. Blockelet, S. Schmitz, Model checking coverability graphs of vector addition systems, in: Proceedings of the 36th International Conference on Mathematical Foundations of Computer Science, in: Lecture Notes in Computer Science, vol. 6907, Springer, 2011, pp. 108–119.
[26] D. Porello, N. Troquard, A resource-sensitive account of the use of artifacts, in: A.L.C. Bazzan, M.N. Huhns, A. Lomuscio, P. Scerri (Eds.), Proceedings of the 13th International Conference on Autonomous Agents and Multi-Agent Systems, AAMAS 2014, IFAAMAS/ACM, 2014, pp. 1549–1550.
[27] D. Porello, N. Troquard, A resource-sensitive logic of agency, in: T. Schaub, G. Friedrich, B. O'Sullivan (Eds.), Proceedings of the 21st European Conference on Artificial Intelligence, ECAI 2014, IOS Press, 2014, pp. 723–728.
[28] P.W. O'Hearn, D.J. Pym, The logic of bunched implications, Bull. Symb. Log. 5 (02) (1999) 215–244.
[29] D.J. Pym, P.W. O'Hearn, H. Yang, Possible worlds and resources: the semantics of BI, Theor. Comput. Sci. 315 (1) (2004) 257–305.
[30] J. Brotherston, C. Calcagno, Classical BI: a logic for reasoning about dualising resources, in: Z. Shao, B.C. Pierce (Eds.), Proceedings of the 36th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2009, ACM, 2009, pp. 328–339.
[31] M. Collinson, B. Monahan, D.J. Pym, A logical and computational theory of located resource, J. Log. Comput. 19 (6) (2009) 1207–1244.
[32] R. Alur, T.A. Henzinger, O. Kupferman, Alternating-time temporal logic, J. ACM 49 (5) (2002) 672–713.
[33] J.E. Hopcroft, J.D. Ullman, Introduction to Automata Theory, Languages, and Computation, Addison–Wesley, 1979.
[34] J.L. Peterson, Petri Net Theory and the Modeling of Systems, Prentice Hall, Englewood Cliffs (NJ), 1981.
[35] N. Alechina, N. Bulling, S. Demri, B. Logan, On the complexity of resource-bounded logics, in: K.G. Larsen, I. Potapov, J. Srba (Eds.), Proceedings of the 10th International Workshop on Reachability Problems, RP 2016, in: Lecture Notes in Computer Science, vol. 9899, Springer, 2016, pp. 36–50.