Model-checking ATL under Imperfect Information and Perfect Recall Semantics is Undecidable

Cătălin Dima[a], Ferucio Laurenţiu Ţiplea[b]

[a]*LACL, Université Paris Est-Créteil, 61 av. du G-ral de Gaulle, 94010 Créteil, France*
[b]*Department of Computer Science, "Al.I.Cuza" University of Iaşi, Iaşi 700506, Romania*

Abstract

We propose a formal proof of the undecidability of the model checking problem for alternating-time temporal logic under imperfect information and perfect recall semantics. This problem was announced to be undecidable according to a personal communication on multi-player games with imperfect information, but no formal proof was ever published. Our proof is based on a direct reduction from the non-halting problem for Turing machines.

Keywords: Alternating-time temporal logic, imperfect information, perfect recall, model checking, decidability

1. Introduction

The Alternating-time Temporal Logic (ATL) have been introduced in [1] as a logic to reason about strategic abilities of agents in multi-agent systems. ATL extends CTL by replacing the path quantifiers $\forall$ and $\exists$ by *cooperation modalities* $\langle\!\langle A \rangle\!\rangle$, where A is a team of agents. A formula $\langle\!\langle A \rangle\!\rangle\varphi$ expresses that the team A has a collective strategy to enforce φ.

The semantics of ATL is defined over *concurrent game structures* (CGS) [1] which are transition systems whose states are labeled by atomic propositions and for which a set of agents is specified. Each agent may have *incomplete/imperfect information* about the state of the system in the sense that the agent may not be able to difference between some states. When the agent is able to observe the entire state labeling, we say that he has *complete/perfect information*. A transition from a state to another one is performed by an action tuple consisting of an action for each agent in the system. The action an agent is allowed to perform at a state is chosen from a given set of actioned allowed to be performed by the agent at that state and may depend on the current state (this is called *imperfect recall*) or on the whole history of events that have happened (this is called *perfect recall*). Combining imperfect or perfect information with imperfect or perfect recall we obtain four types of concurrent game structures and, consequently, four types of semantics for ATL.

A series of papers have been addressed the model-checking problem for ATL [1, 3, 2]. Based on unpublished work of Yannakakis [4], the model checking problem for ATL with imperfect information and perfect recall semantics was announced to be undecidable in [1]. Since then, many authors have mentioned this result but, unfortunately, no formal proof was ever published (see also [2]).

In this paper we propose a formal proof of this problem. Our proof is based on a direct simulation of Turing machines by concurrent game structures under imperfect information and perfect recall, which allows for a reduction of the non-halting problem for Turing machines to the model checking problem for ATL under imperfect information and perfect recall semantics. Moreover, the strategies used by agents to simulate the Turing machine are primitive recursive. This shows that the undecidability of model checking ATL under imperfect information and perfect recall semantics is mainly due to the imperfect information agents have about the system states.

While our proof is given for the *de dicto* strategies from [1], the same construction works also for the *de re* strategies from [3, 5].

2. Alternating-time Temporal Logic

We recall in this section the syntax and semantics of the alternating-time temporal logic. We will mainly follow the approach in [2] and fix first a few notations. $\mathbb{N}$ stands for the set of positive integers (natural numbers) and $\mathcal{P}$ denotes the powerset operator. Given a set V, V^+ denotes the free semi-group and V^* denotes the free monoid generated by V under concatenation. λ stands for the empty word (the unity of V^*). The notation $f : X \rightharpoonup Y$ means that f is a partially defined function from X to Y.

ATL syntax. The syntax of ATL is given by the grammar

$$\varphi ::= p \mid \neg\varphi \mid \varphi \wedge \varphi \mid \langle\!\langle A \rangle\!\rangle \bigcirc \varphi \mid \langle\!\langle A \rangle\!\rangle \square \varphi \mid \langle\!\langle A \rangle\!\rangle \varphi U \varphi$$

where p ranges over a finite non-empty set of *atomic propositions* Π, A is a non-empty subset of a finite set Ag of *agents*, and $\bigcirc$, $\square$, and U are the standard temporal operators *next*, *globally*, and *until*, respectively.

Note that, in order to define combinations of temporal operators inside the coalition operators, the *weak-until* operator should be given as a primitive operator [6], since it cannot be derived from the above operators. However our result holds also for this restricted syntax.

ATL semantics. ATL is interpreted over *concurrent game structures* (CGS) [1]. Such a structure consists of a set of states labeled by atomic propositions and a set of agents. Each agent may perform some actions and at least one action is available to the agent at each state. His decision in choosing which action should be performed at some state may be based on his capability of observing all or some atomic propositions at the current state, usually called *perfect* or *imperfect information*, and on his full or partial history, usually called *perfect* or *imperfect recall*.

2

In what follows we focus on CGS under imperfect information and perfect recall and adopt the formal approach in [2]. A CGS *under imperfect information* is a tuple $\mathcal{G} = (Ag, S, \Pi, \pi, (\sim_i \mid i \in Ag), Act, d, \rightarrow)$, where:

- $Ag = \{1, \ldots, k\}$ is a finite non-empty set of *agents*;

- S is a finite non-empty set of *states*;

- Π is a finite non-empty set of *atomic propositions*;

- $\pi : S \rightarrow \mathcal{P}(\Pi)$ is the *state-labeling function*;

- $\sim_i$ is an equivalence relation on S, for any agent i;

- Act is a finite non-empty set of *actions*;

- $d : Ag \times S \rightarrow \mathcal{P}(Act) - \{\varnothing\}$ gives the set of actions available to agents at each state, satisfying $d(i, s) = d(i, s')$ for any agent i and states s and s' with $s \sim_i s'$;

- $\rightarrow : S \times Act^k \rightharpoonup S$ is the (partially defined) *transition function* satisfying, for any $s \in S$ and $(a_1, \ldots, a_k) \in Act^k$, the following property:

$$\rightarrow (s, (a_1, \ldots, a_k)) \text{ is defined iff } a_i \in d(i, s) \text{ for any agent } i.$$

We will write $s \xrightarrow{(a_1, \ldots, a_k)} s'$, whenever $\rightarrow (s, (a_1, \ldots, a_k)) = s'$.

If s and s' are $\sim_i$-equivalent (i.e., $s \sim_i s'$) then we say that s and s' are *indistinguishable* from the agent i's point of view (due to his partial view on the states). Each $\sim_i$ is component-wise extended to sequences of states. Thus, for $\alpha, \alpha' \in S^+$ we write $\alpha \sim_i \alpha'$ and say that α and α' are *$\sim_i$-equivalent* if $\alpha = s_0 \cdots s_n$ and $\alpha' = s_0' \cdots s_n'$ for some $n \in \mathbb{N}$, and $s_j \sim_i s_j'$ for all $0 \leq j \leq n$.

A *perfect recall strategy* for an agent i in a CGS $\mathcal{G}$ is a function $\sigma : S^+ \rightarrow Act$ which is *compatible* with d and $\sim_i$, i.e.,

- $\sigma(\alpha s) \in d(i, s)$, for any $\alpha \in S^*$ and $s \in S$;

- $\sigma(\alpha) = \sigma(\alpha')$, for any $\alpha, \alpha' \in S^+$ with $\alpha \sim_i \alpha'$.

A *perfect recall strategy for a team A of agents* is a family $\sigma_A = (\sigma_i \mid i \in A)$ of perfect recall strategies for the agents in A. If σ_A is a perfect recall strategy for the agents in A, $\alpha s \in S^* S$, and $a = (a_1, \ldots, a_k) \in Act^k$, then we write $a \in \overline{\sigma}_A(\alpha s)$ if the following properties hold:

- $a_i \in d(i, s)$, for any $i \in Ag - A$;

- $a_i \in \sigma_i(\alpha s)$, for any $i \in A$.

Given a state s of $\mathcal{G}$ and σ_A as above, define $out_{\mathcal{G}}(s, \sigma_A)$ as being the set of all infinite sequences of states $\lambda = s_0 s_1 s_2 \cdots$ such that $s_0 = s$ and, for any $j \geq 0$, there exists $a \in \overline{\sigma}_A(s_0 \cdots s_j)$ with $s_j \xrightarrow{a} s_{j+1}$. For $\lambda = s_0 s_1 s_2 \cdots$ an infinite sequence of states and $j \geq 0$, $\lambda[j]$ denotes the j-th state in the sequence, $\lambda[j] = s_j$

The *imperfect information perfect recall semantics* for ATL, denoted $\vDash_{iR}$, is defined as follows ($\mathcal{G}$ is a CGS under imperfect information and s is a state of $\mathcal{G}$):

- $(\mathcal{G}, s) \vDash_{iR} p$ if $p \in \pi(s)$;

- $(\mathcal{G}, s) \vDash_{iR} \neg\varphi$ if $(\mathcal{G}, s) \nvDash_{iR} \varphi$;

- $(\mathcal{G}, s) \vDash_{iR} \varphi \wedge \psi$ if $(\mathcal{G}, s) \vDash_{iR} \varphi$ and $(\mathcal{G}, s) \vDash_{iR} \psi$;

- $(\mathcal{G}, s) \vDash_{iR} \langle\!\langle A \rangle\!\rangle \bigcirc \varphi$ if there exists a perfect recall strategy σ_A such that $(\mathcal{G}, \lambda[1]) \vDash_{iR} \varphi$, for any $\lambda \in out_{\mathcal{G}}(s, \sigma_A)$;

- $(\mathcal{G}, s) \vDash_{iR} \langle\!\langle A \rangle\!\rangle \square \varphi$ if there exists a perfect recall strategy σ_A such that $(\mathcal{G}, \lambda[j]) \vDash_{iR} \varphi$, for any $\lambda \in out_{\mathcal{G}}(s, \sigma_A)$ and any $j \geq 0$;

- $(\mathcal{G}, s) \vDash_{iR} \langle\!\langle A \rangle\!\rangle \varphi U \psi$ if there exists a perfect recall strategy σ_A such that for any $\lambda \in out_{\mathcal{G}}(s, \sigma_A)$ there exists $j \geq 0$ with $(\mathcal{G}, \lambda[j]) \vDash_{iR} \psi$ and $(\mathcal{G}, \lambda[k]) \vDash_{iR} \varphi$ for all $0 \leq k < j$.

The *model checking problem* for ATL formulas under imperfect information and perfect recall semantics is to decide, given an ATL formula φ, a concurrent game structure $\mathcal{G}$ under imperfect information, and a state s of $\mathcal{G}$, whether $(\mathcal{G}, s) \vDash_{iR} \varphi$.

Computation trees. The proof of our main result in the next section will be based on *computation trees* associated to CGSs. These are special cases of labeled trees, which are structures $\mathcal{T} = (V, E, v_0, l_1, l_2)$, where

- (V, E, v_0) is a tree whose set of nodes is V, whose set of edges is E, and whose root is v_0;

- l_1 is the node-labeling function;

- l_2 is the edge-labeling function.

Paths in a labeled tree $\mathcal{T} = (V, E, v_0, l_1, l_2)$ are defined inductively as usual as sequences of nodes:

- v_0 is a path in $\mathcal{T}$;

- if $v_0 \cdots v_n$ is a path in $\mathcal{T}$ and $(v_n, v) \in E$, then $v_0 \cdots v_n v$ is a path in $\mathcal{T}$.

If v is a node of $\mathcal{T}$, then $path_{\mathcal{T}}(v_0, v)$ stands for the unique path from the root v_0 to v in $\mathcal{T}$. The number of nodes on a path τ is the length of τ, denoted $|\tau|$. The labeling function l_1 is homomorphically extended to paths, that is, $l_1(\tau_1 \tau_2) = l_1(\tau_1) l_1(\tau_2)$.

Levels in a labeled tree $\mathcal{T} = (V, E, v_0, l_1, l_2)$ are sets of nodes of $\mathcal{T}$ defined inductively as follows:

- $level_\mathcal{T}(0) = \{v_0\}$;

- $level_\mathcal{T}(n+1) = \{v \in V | (\exists v' \in level_\mathcal{T}(n))((v', v) \in E)\}$, for any $n \geq 0$.

$level_\mathcal{T}(n)$ is referred to as the *level n in* $\mathcal{T}$.

Given a CGS $\mathcal{G}$, a state s of $\mathcal{G}$, a coalition A of agents, and a perfect recall strategy σ_A for agents in A, define inductively the *s-rooted computation trees of* $\mathcal{G}$ *under* σ_A as follows:

- any tree with exactly one node (its root) labeled by s is an s-rooted computation tree of $\mathcal{G}$ under σ_A;

- if $\mathcal{T} = (V, E, v_0, l_1, l_2)$ is an s-rooted computation tree of $\mathcal{G}$ under σ_A, v is a node of $\mathcal{T}$, and $l_1(v) \xrightarrow{a} s'$ for some action-tuple $a \in \overline{\sigma}_A(l_1(path_\mathcal{T}(v_0, v)))$ and state s' such that no edge from v is labeled by a, then the tree $\mathcal{T}'$ obtained as follows is an s-rooted computation tree of $\mathcal{G}$:

  - $\mathcal{T}'$ is obtained from $\mathcal{T}$ by adding a new node v' labeled by s' and an edge (v, v') labeled by a.

If $\mathcal{T}'$ is obtained from $\mathcal{T}$ as above, we will also write $\mathcal{T} \Rightarrow_{\mathcal{G}, \sigma_A} \mathcal{T}'$ or $\mathcal{T} \xRightarrow{a}_{\mathcal{G}, \sigma_A} \mathcal{T}'$ if we want to specify the action tuple a as well.

Remark 1. *It is easy to see that, for any atomic proposition p, the following property holds true:*

- *$(\mathcal{G}, s) \vDash_{iR} \langle\!\langle A \rangle\!\rangle \square\, p$ if and only if there exists a perfect recall strategy σ_A such that $p \in \pi(l_1(v))$, for any s-rooted computation tree $\mathcal{T}$ of $\mathcal{G}$ under σ_A, and any node v of $\mathcal{T}$.*

3. Undecidability of Model Checking ATL_{iR}

We will prove in this section that the model checking problem for ATL_{iR} is undecidable. The proof technique is by reduction from the non-halting problem for deterministic Turing machines. Given a deterministic Turing machine M, we construct a concurrent game structure under imperfect information $\mathcal{G}$ with three agents $Ag = \{1, 2, 3\}$, a state s_{init} of $\mathcal{G}$, and an ATL formula $\langle\!\langle \{1, 2\} \rangle\!\rangle \square\, ok$, where ok is an atomic proposition, such that M does not halt on the empty word if and only if $(\mathcal{G}, s_{init}) \vDash_{iR} \langle\!\langle \{1, 2\} \rangle\!\rangle \square\, ok$.

The deterministic Turing machines we consider are tuples $M = (Q, \Sigma, q_0, B, \delta)$, where Q is a finite set of states, Σ is a finite tape alphabet, q_0 is the initial state, $B \in \Sigma$ is the blank symbol, and $\delta : Q \times \Sigma \rightharpoonup Q \times \Sigma \times \{L, R\}$ is a partially defined transition function, where "L" specifies a "left move" and "R" specifies a "right move". A configuration of M is a word $a_1 \cdots a_{i-1} q a_i \cdots a_n$, where all a's are from Σ and q is a state. Such a configuration specifies that M is in state q, its read/write head points to the ith cell of the tape, and the jth cell holds a_j if $j \leq n$, and B, otherwise. The initial configuration is $q_0 B$. The transition relation on configurations, denoted $\Rightarrow_M$, is defined as usual. For instance, $a_1 \cdots a_{i-1} q a_i \cdots a_n \Rightarrow_M a_1 \cdots q' a_{i-1} a_i' \cdots a_n$ if $i > 1$ and $\delta(q, a_i) = (q', a_i', L)$.

The Turing machine M *halts* on the empty word if, starting with the initial configuration, the machine reaches a configuration $a_1 \cdots a_{i-1} q a_i \cdots a_n$ for which $\delta(q, a_i)$ is undefined or $i = 1$ and $\delta(q, a_i) = (q', a_i', L)$ for some q' and a_i'.

Intuition first. The main idea of the construction is to encode the configurations of the Turing machine horizontally in the levels of the computation tree. A configuration $a_1 \cdots a_{i-1} q a_i \cdots a_k$ of M will be simulated in $\mathcal{A}$ by some level in some computation tree like in Figure 1 (where $i = 2$ and $k = 3$). The nodes of this tree are represented by circles. The label of a node
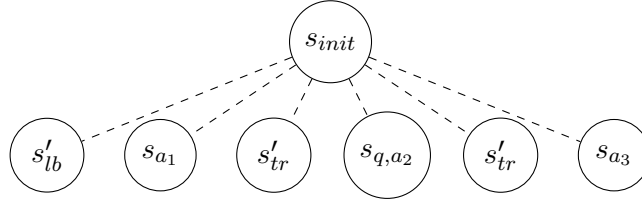

Figure 1: Level corresponding to $a_1 q a_2 a_3$

is carried inside the circle representing the node. The node labeled s'_{lb} specifies the left border of M's tape, the node labeled s'_{tr} is a cell separator also used to transfer information between paths of computation trees, the nodes labeled s_{a_1} and s_{a_3} specify the content of the first and third cell, respectively, and the node labeled s_{q,a_2} specifies both the content of the second cell and the fact that M is in state q and its read/write head points to the second cell.

The generation of the initial configuration $q_0 B$ of M is simulated by the computation tree in Figure 2. All states in this tree ale labeled by *ok*; the node labeled s_{gen} has one more label, namely p_1 (this label is graphically represented because it will be particularly important in defining the agents strategies). As we will see later, the two maximal paths in this tree are $\sim_2$-equivalent. This allows, together with the strategy we will use, for the synchronization in the last computation step of these paths.
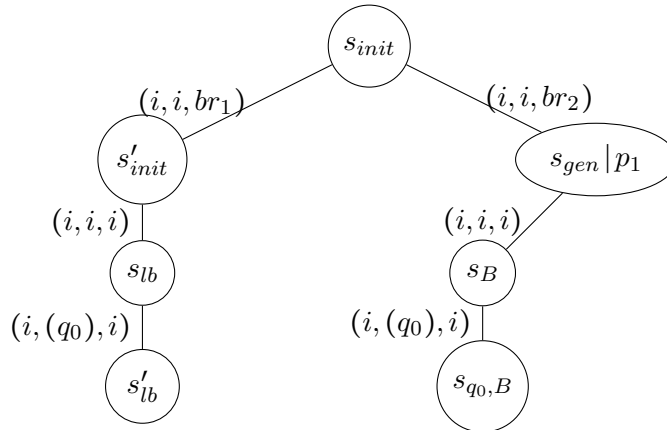

Figure 2: Generating the initial configuration $q_0 B$ of M

6

The levels encoding configurations of the Turing machine will be encoded on the *even* positions in a computation tree, the odd levels being used for correctly representing transitions of the Turing machine. Some nodes in the levels of even index will then encode tape cells, while some other nodes will be used for transferring information between adjacent cells. Some examples presenting this idea are given in the following, before the formal construction and proof.

A computation step $a_1 q a_2 a_3 \Rightarrow_M a_1 a_2' q' a_3$ in the Turing machine is simulated by extending the computation tree in Figure 1 as in Figure 3. The synchronization between the fourth and fifth paths is possible because, as we will see, these paths are $\sim_1$-equivalent. Similarly, the synchronization between the fifth and sixth paths is possible because these paths are $\sim_2$-equivalent.
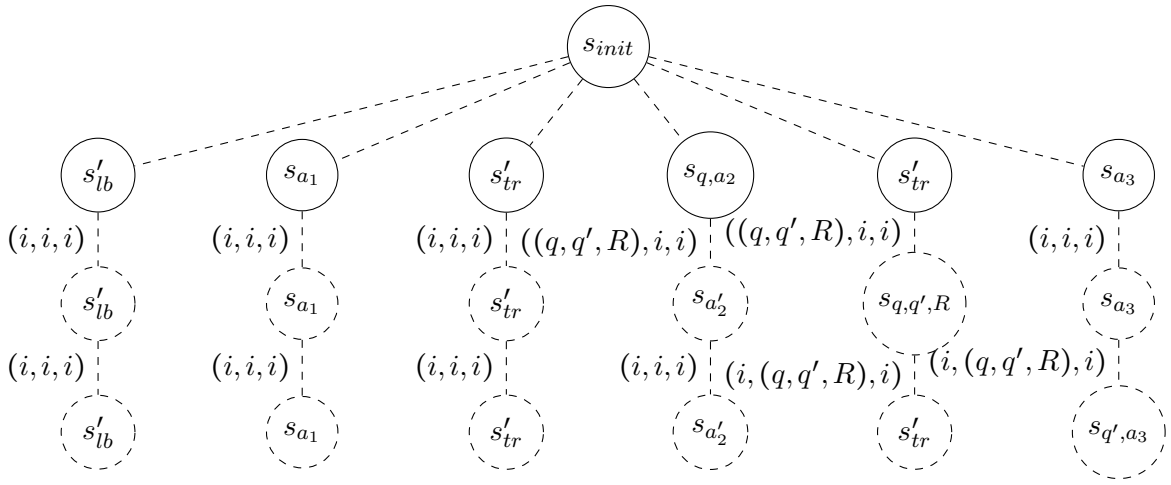

Figure 3: Simulation of $a_1 q a_2 a_3 \Rightarrow_M a_1 a_2' q' a_3$

The simulation represented in these two figures proceeds as follows: in the observable history corresponding to the path ending in s_{q,a_2}, the only possibility for agent 1 to put the system in a state which satisfies ok at the next level is to take action (q, q', R), which corresponds to the transition $\delta(q, a_2) = (q', a_2', R)$ in the Turing machine. Due to identic observability for agent 1, the same action has to be played by agent 1 in the history which ends in state s_{tr}' which is next to the right of state s_{q,a_2}. The effect of this action in state s_{tr}' (combined with an idle action for agent 2) is to bring the system in state $s_{q,q',R}$. In this state, it's upto agent 2 to try to satisfy ok at the next step, and he can only do this by applying the action (q, q', R). The effect of this action in state $s_{q,q',R}$ is to bring the system back in state s_{tr}'. But the same action has to be played by agent 2 in the history which ends in state s_{a_3} on level 3 of the tree, due to identical observability. This play will lead the system to state s_{q',a_3}.

On the other hand, in state s_{a_1}, in order to ensure ok, both agents must play idle, which leaves the system in state s_{a_1}. Identical observability will then ensure that agent 1 has to play idle also in state s_{tr}' which is next to the right of state s_{a_1}, and agent 2 has to play idle in state s_{lb}' on 3rd and 4th levels.

7

The effect of all these is that level 4 on this tree encodes the configuration $a_1 q' a_2' a_3$, which results from applying the transition $\delta(q, a_2) = (q', a_2', R)$ to the configuration $a_1 q a_2 a_3$. States s_{gen} and s_{tr} are used for "creating" all the nodes that simulate tape cells. In a computation tree which satisfies the goal $\square ok$, these are the only states to have two sons.

Figure 4 presents the simulation of the computation step $a_1 q a_2 a_3 \Rightarrow_M q' a_1 a_2' a_3$ Note here that the rôle of agents 1 and 2 are interchanged because it is a left transition.
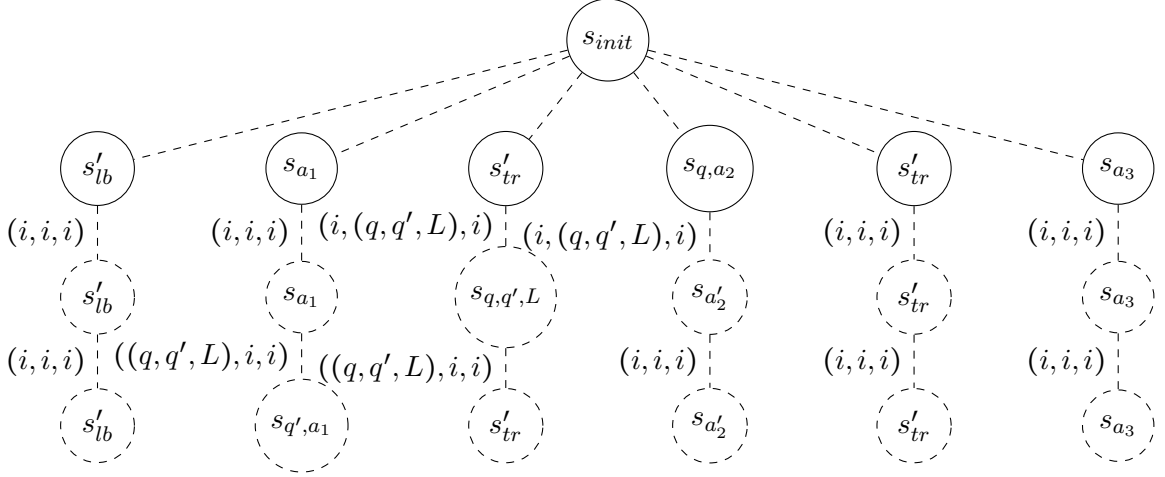

Figure 4: Simulation of $a_1 q a_2 a_3 \Rightarrow_M q' a_1 a_2' a_3$

And in Figure 5, a simulation of the computation $q_0 B \Rightarrow_M a q_1 B \Rightarrow_M q_2 ab$ is shown.

Construction of a game structure associated to M. The concurrent game structure under imperfect information $\mathcal{G} = (Ag, S, \Pi, \pi, Act, (\sim_i \mid i \in Ag), d, \rightarrow)$ that simulates the deterministic Turing machine M is based on three agents, i.e. $Ag = \{1, 2, 3\}$. Its set S of states, together with their meaning, consists of:

- s_{init} (the initial state);

- s'_{init} (copy of s_{init});

- s_{lb} (specifies the left border of M's tape);

- s'_{lb} (copy of s_{lb});

- s_{gen} (initiates the generation of a new blank cell of M's tape);

- s_{tr} (initiates the generation of a new cell separator);

- s'_{tr} (used for transferring information between to equivalent runs);

- s_a, for any $a \in \Sigma$ (specifies that some tape cell holds a);

- $s_{q,a}$, for any state $q \in Q$ and $a \in \Sigma$ (specifies that M is in state q and the read/write head points a cell holding symbol a);
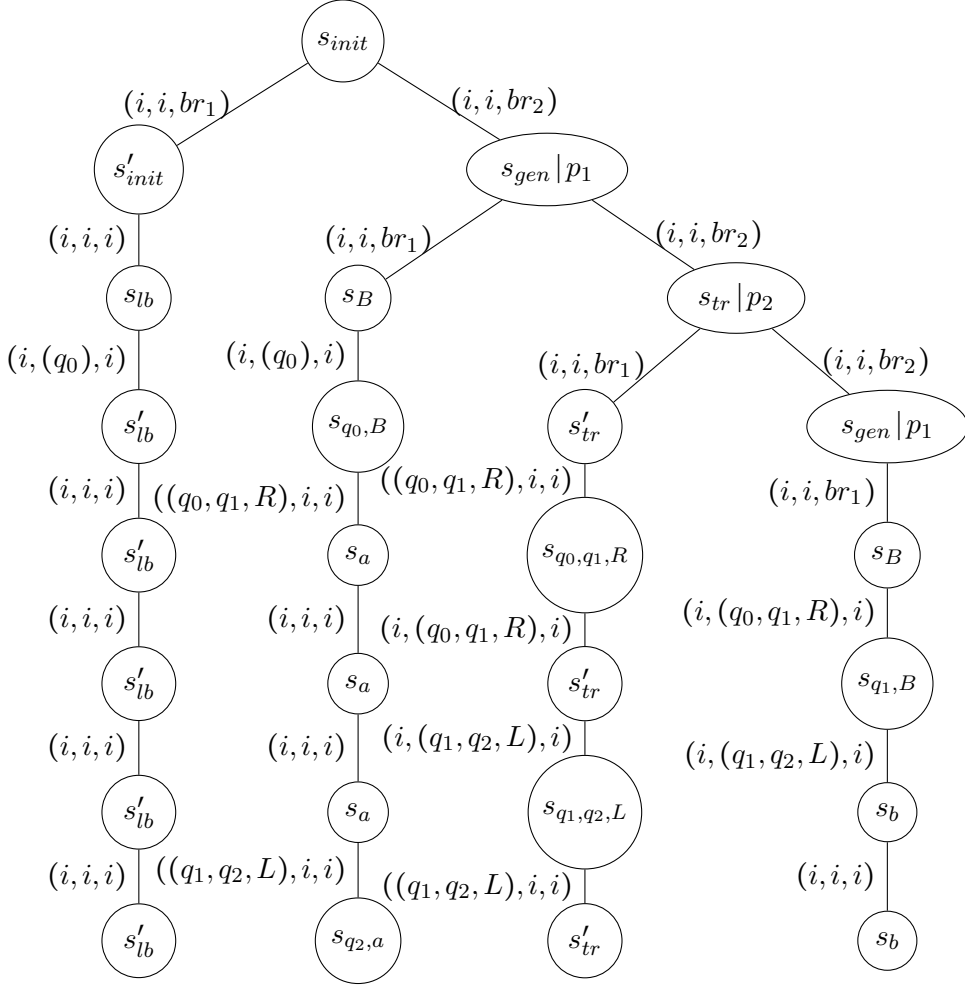
8

Figure 5: Simulation of the computation $q_0B \Rightarrow_M aq_1B \Rightarrow_M q_2ab$.

- $s_{q,q',X}$, for any $q, q' \in Q$ and $X \in \{L, R\}$ such that $\delta(q, a) = (q', a', X)$ for some a and a' (specifies that the machine M enters state q' from state q by an X-move);

- s_{err} ("error" state used to collect all "unwanted" transitions the agents must avoid bringing the system in this state).

The set of atomic propositions is $\Pi = \{p_1, p_2, ok\}$ and the labeling function π is:

$$\pi(s) = \begin{cases} \{ok\}, & \text{if } s \in S - \{s_{gen}, s_{tr}, s_{err}\} \\ \{p_1, ok\}, & \text{if } s = s_{gen} \\ \{p_2, ok\}, & \text{if } s = s_{tr} \\ \varnothing, & \text{if } s = s_{err} \end{cases}$$

For the sake of simplicity, all states but s_{err} will be called *ok-states* (being labeled by *ok*). The relation $\sim_3$ is the identity. The equivalence relations $\sim_1$ and $\sim_2$ are defined by

$$s \sim_i s' \quad \text{iff} \quad (p_i \in \pi(s) \iff p_i \in \pi(s')),$$

9

for any $i = 1, 2$. That is, s and s' are $\sim_i$-equivalent if the agent i observes p_i either in both states s and s' or in none of them.

The set Act of actions consists of:

- *idle*, which is meant to say that the agent doing it is not "in charge of" accomplishing some local objective (this action will be abbreviated by i in our pictures and whenever no confusion may arise);

- (q_0), which is an action meant to set up the initial state of M;

- (q, q', X), for any $q, q' \in Q$ and $X \in \{L, R\}$ with $\delta(q, a) = (q', a', X)$ for some $a, a' \in \Sigma$. Such an action simulates the passing of M from q to q' by an X-move;

- br_1 and br_2, which are two "branching" actions.

The agents 1 and 2 are allowed to perform any action but br_1 and br_2, while the third agent can only perform br_1, br_2, and *idle*. More precisely, $d(i, s) = Act - \{br_1, br_2\}$ for any $i \in \{1, 2\}$ and state s, $d(3, s) = \{br_1, br_2\}$ if $s \in \{s_{init}, s_{gen}, s_{tr}\}$, and $d(3, s) = idle$, otherwise.

Note that the agents' actions are designed such that $d(i, s) = d(i, s')$ for any agent i and states s and s' with $s \sim_i s'$.

The transition relation of the game structure is as follows:

- $s_{init} \xrightarrow{(i,i,br_1)} s'_{init}$ and $s_{init} \xrightarrow{(i,i,br_2)} s_{gen}$ and $s_{init} \xrightarrow{c} s_{err}$, for any c different from the above action tuples;

- $s'_{init} \xrightarrow{(i,i,i)} s_{lb}$ and $s'_{init} \xrightarrow{c} s_{err}$, for any $c \neq (i, i, i)$;

- $s_{lb} \xrightarrow{(i,(q_0),i)} s'_{lb}$ and $s_{lb} \xrightarrow{c} s_{err}$, for any $c \neq (i, (q_0), i)$;

- $s'_{lb} \xrightarrow{(i,i,i)} s'_{lb}$ and $s'_{lb} \xrightarrow{c} s_{err}$, for any $c \neq (i, i, i)$;

- $s_{gen} \xrightarrow{(i,i,br_1)} s_B$ and $s_{gen} \xrightarrow{(i,i,br_2)} s_{tr}$ and $s_{gen} \xrightarrow{c} s_{err}$, for any c different from the above action tuples;

- $s_{tr} \xrightarrow{(i,i,br_1)} s'_{tr}$ and $s_{tr} \xrightarrow{(i,i,br_2)} s_{gen}$ and $s_{tr} \xrightarrow{c} s_{err}$, for any c different from the above action tuples;

- for any $a \in \Sigma$, the transitions at s_a are:

    - $s_a \xrightarrow{(i,i,i)} s_a$;

    - $s_B \xrightarrow{(i,(q_0),i)} s_{q_0,B}$;

    - $s_a \xrightarrow{(i,(q,q',R),i)} s_{q,a}$, for any action (q, q', R);

    - $s_a \xrightarrow{((q,q',L),i,i)} s_{q,a}$, for any action (q, q', L);

$-$ $s_a \xrightarrow{c} s_{err}$, for any c different from any of the above actions;

- for any $q \in Q$ and $a \in \Sigma$, the transitions at $s_{q,a}$ are:

  $-$ $s_{q,a} \xrightarrow{((q,q',R),i,i)} s_{a'}$, if $\delta(q,a) = (q',a',R)$;

  $-$ $s_{q,a} \xrightarrow{((q,q',L),i,i)} s_{a'}$, if $\delta(q,a) = (q',a',L)$;

  $-$ $s_{q,a} \xrightarrow{c} s_{err}$, for any c different from any of the above actions;

- the transitions at s'_{tr} are:

  $-$ $s'_{tr} \xrightarrow{(i,i,i)} s'_{tr}$.

  $-$ $s'_{tr} \xrightarrow{((q,q',R),i,i)} s_{q,q',R}$, for any action (q,q',R);

  $-$ $s'_{tr} \xrightarrow{(i,(q,q',L),i)} s_{q,q',L}$, for any action (q,q',L);

  $-$ $s'_{tr} \xrightarrow{c} s_{err}$, for any c different from any of the above actions;

- $s_{q,q',R} \xrightarrow{(i,(q,q',R),i)} s'_{tr}$ and $s_{q,q',L} \xrightarrow{((q,q',L),i,i)} s'_{tr}$ and $s_{q,q',X} \xrightarrow{c} s_{err}$, for any X and any c different from any of the above actions.

Proof of the correctness of the construction. Let M be a deterministic Turing machine. Without loss of generality we may assume that M, starting in state q_0, will never reach again q_0.

First, we prove that if M does not halt on the empty word then $(\mathcal{G}, s_{init}) \vDash_{iR} \langle\!\langle \{1,2\} \rangle\!\rangle \square$ *ok*. According to Remark 1, it suffices to show that, if M does not halt on the empty word, then there exists a strategy $\sigma = (\sigma_1, \sigma_2)$ for the agents 1 and 2 in $\mathcal{G}$ such that any s_{init}-rooted computation tree of $\mathcal{G}$ under σ has only nodes labeled by *ok*-states.

In order to define σ with the property above, we classify the non-empty sequences of states of $\mathcal{G}$ as follows:

- a sequence $\alpha \in S^+$ is of *type 1* if $\alpha = s_{init} s'_{init} \alpha'$, where $\alpha' \in S^*$;

- a sequence $\alpha \in S^+$ is of *type 2* if $\alpha = s_{init} s_{gen} \alpha'$, where $\alpha' \in S^*$. Type 2 sequences of states can be further classified according to the number of states s_{gen} and s_{tr} they contain:

  $-$ a sequence α is of *type 2(i)(i−1)*, where $i \geq 1$, if $\alpha = s_{init}(s_{gen} s_{tr})^{i-1} s_{gen} \alpha'$, where $\alpha' \in S^*$ does not contain s_{gen} and s_{tr};

  $-$ a sequence α is of *type 2(i)(i)*, where $i \geq 1$, if $\alpha = s_{init}(s_{gen} s_{tr})^i \alpha'$, where $\alpha' \in S^*$ does not contain s_{gen} and s_{tr}.

Of course, there are sequences $\alpha \in S^+$ which are neither of type 1 nor of type 2. A path τ of a computation tree of $\mathcal{G}$ will be called of *type x* if $l_1(\tau)$ is of type x, where x is as above.

The following claim follows easily from definitions.

Claim 1. *Let α and α' be two non-empty sequences of states. Then, the following properties hold:*

1. *If α is of type 1 and α' is of type 2, then $\alpha \not\sim_1 \alpha'$;*
2. *If α is of type 1 and α' is of type 2 and $\alpha \sim_2 \alpha'$, then α' is of type $2(1)(0)$;*
3. *If α and α' are of type 2, have a different number of s_{gen} or s_{tr} states, and $\alpha \sim_1 \alpha'$, then α is of type $2(i)(i-1)$ and α' is of type $2(i)(i)$, or vice-versa;*
4. *If α and α' are of type 2, have a different number of s_{gen} or s_{tr} states, and $\alpha \sim_2 \alpha'$, then α is of type $2(i)(i)$ and α' is of type $2(i+1)(i)$, or vice-versa.*

Now, define a strategy $\sigma = (\sigma_1, \sigma_2)$ as follows:

- $\sigma_1(s_{init}) = \sigma_1(\alpha) = idle$, for any type 1 sequence $\alpha \in S^+$;

- $\sigma_2(s_{init}) = \sigma_2(\alpha) = idle$, for any type 1 sequence $\alpha \in S^+$ different from $s_{init}s'_{init}s_{lb}$, and $\sigma_2(s_{init}s'_{init}s_{lb}) = (q_0)$;

- $\sigma_1(\alpha s_{q,a}) = (q, q', R) = \sigma_1(\alpha's'_{tr})$, for any $\alpha s_{q,a}$ of type $2(i)(i-1)$ and any $\alpha's'_{tr}$ of type $2(i)(i)$ for which $i \geq 1$ and the following property holds:

  - $|\alpha s_{q,a}| = 3 + (2j - 1) = |\alpha's'_{tr}|$ for some $j \geq 1$, and the agent 1 simulating the first j steps of M deduces that the current configuration of M is of the form $uqav$, where $|u| = i - 1$, and $\delta(q, a) = (q', a', R)$, for some q' and a';

- $\sigma_1(\alpha s_a) = (q, q', L) = \sigma_1(\alpha's_{q,q',L})$, for any αs_a of type $2(i)(i-1)$ and any $\alpha's_{q,q',L}$ of type $2(i)(i)$ for which $i \geq 1$ and the following property holds:

  - $|\alpha s_a| = 3 + 2j = |\alpha's_{q,q',L}|$ for some $j \geq 1$, and the agent 1 simulating the first j steps of M deduces that the current configuration of M is of the form $uqbv$, where $|u| = i - 1$, and $\delta(q, b) = (q', b', L)$, for some q' and b';

- $\sigma_2(\alpha s_{q,q',R}) = (q, q', R) = \sigma_2(\alpha's_a)$, for any $\alpha s_{q,q',R}$ of type $2(i)(i)$ and any $\alpha's_a$ of type $2(i+1)(i)$ for which $i \geq 1$ and the following property holds:

  - $|\alpha s_{q,q',R}| = 3 + 2j = |\alpha's_a|$ for some $j \geq 1$, and the agent 2 simulating the first j steps of M deduces that the current configuration of M is of the form $uqav$, where $|u| = i - 1$, and $\delta(q, a) = (q', a', R)$, for some q' and a';

- $\sigma_2(\alpha s'_{tr}) = (q, q', L) = \sigma_2(\alpha's_{q,a})$, for any $\alpha s'_{tr}$ of type $2(i)(i)$ and any $\alpha's_{q,a}$ of type $2(i+1)(i)$ for which $i \geq 1$ and the following property holds:

  - $|\alpha s'_{tr}| = 3 + (2j - 1) = |\alpha's_{q,a}|$ for some $j \geq 1$, and the agent 2 simulating the first j steps of M deduces that the current configuration of M is of the form $uqbv$, where $|u| = i - 1$, and $\delta(q, b) = (q', b', L)$, for some q' and b';

- $\sigma_2(s_{init}s_{gen}s_B) = (q_0)$;

- $\sigma_1(\alpha) = idle$ and $\sigma_2(\alpha') = idle$ for all the other cases.

The strategies σ_1 and σ_2 are both compatible with d, σ_1 is compatible with $\sim_1$, and σ_2 is compatible with $\sim_2$.

Any tree with exactly one node (its root) labeled by s_{init} is an s_{init}-rooted computation tree of $\mathcal{G}$ under σ and its nodes are all labeled by ok-states.

Assume that $\mathcal{T}$ is an s_{init}-rooted computation tree of $\mathcal{G}$ under σ and all its nodes are labeled by ok-states. It is easy to see that $\mathcal{T}$ may only have type 1, type $2(i)(i-1)$, or type $2(i)(i)$ paths, for some $i \geq 1$. Any extension $\mathcal{T}'$ of $\mathcal{T}$ (i.e., $\mathcal{T} \Rightarrow_{\mathcal{G},\sigma} \mathcal{T}'$) adds new nodes to $\mathcal{T}$ which cannot be labeled by s_{err} because M does not halt (see the definition of σ). Therefore, any s_{init}-rooted computation tree of $\mathcal{G}$ under σ has all its nodes labeled by ok-states.

Conversely, we show that M does not halt on the empty word if all s_{init}-rooted computation trees of $\mathcal{G}$ under some strategy σ for $\{1, 2\}$ have only nodes labeled by ok-states.

Let σ be a strategy with the property above and consider an s_{init}-rooted computation tree $\mathcal{T} = (V, E, v_0, l_1, l_2)$ under σ. A node v of $\mathcal{T}$ will be called of *type x* if $l_1(path_{\mathcal{T}}(v_0, v))$ is of type x (x is 1, 2, $2(i)(i-1)$, or $2(i)(i)$, for some $i \geq 1$).

We then define a partial ordering $\prec_{\mathcal{T}}$ on the nodes of $\mathcal{T}$ as the least partial ordering with the following properties:

- if v and v' are nodes on the same level of $\mathcal{T}$ and $l_1(v') \in \{s_{gen}, s_{tr}\}$, then $v \prec_{\mathcal{T}} v'$;

- if v and v' are nodes on the same level of $\mathcal{T}$ and there exist u on the path from root to v and u' on the path from root to v' with $u \prec_{\mathcal{T}} u'$, then $v \prec_{\mathcal{T}} v'$

Some properties of $\mathcal{T}$ and its level sets are listed in the sequel.

Claim 2. *Let $\mathcal{T} = (V, E, v_0, l_1, l_2)$ be an s_{init}-rooted computation tree of $\mathcal{G}$ under σ, and $n \geq 1$. Then:*

1. *$level_{\mathcal{T}}(n)$ has at most $n + 1$ nodes, and each of them is either of type 1, or of type 2, or of type $2(i)(i-1)$, or of type $2(i)(i)$, for some $i \geq 1$;*
2. *$level_{\mathcal{T}}(n)$ contains at most one node of type 1;*
3. *$level_{\mathcal{T}}(n)$ contains at most one node of type $2(i)(i-1)$ and at most one node of type $2(i)(i)$, for each $i \leq \lceil n/2 \rceil$;*
4. *for any $v, v' \in level_{\mathcal{T}}(n)$, $v \prec_{\mathcal{T}} v'$ if and only if one of the following properties hold:*
   (a) *$v = v'$;*
   (b) *v is of type 1;*
   (c) *v is of type $2(i)(i')$, v' is of type $2(j)(j')$, and $i < j$ or, if $i = j$ then $i' < j'$.*
5. *$\prec_{\mathcal{T}}$ is a total ordering on $level_{\mathcal{T}}(n)$.*

PROOF. All the properties in Claim 2 can be proved by induction on $n \geq 1$ and make use of the fact that all nodes of $\mathcal{T}$ are labeled by ok-states. Thus, if v is a node on the level n of $\mathcal{T}$ and it is not label by s_{gen} or s_{tr}, then it may have at most one descendant v' on

13

the level $n+1$ (by σ, each of the agents 1 and 2 has exactly one choice at $l_1(v)$, and by d_3, the agent 3 has exactly one choice as well at $l_1(v)$). Moreover, v' and v have the same type. If v is labeled by s_{gen}, then its type is $2(i)(i-1)$ for some $i \geq 1$, and it may have at most two descendants v' and v'' on the level $n+1$ (by σ, each of the agents 1 and 2 has exactly one choice at $l_1(v)$, but the agent 3 has two choices). One of this descendants is of type $2(i)(i-1)$, while the other is of type $2(i)(i)$ and it is labeled by s_{tr}. Similarly, if v is labeled by s_{tr}, then its type is $2(i)(i)$ for some $i \geq 1$, and it may have at most two descendants v' and v'' on the level $n+1$. One of this descendants is of type $2(i)(i)$, while the other is of type $2(i+1)(i)$ and it is labeled by s_{gen}.

Combining these remarks with the fact that $level_{\mathcal{T}}(1)$ may contain at most two nodes, one of them labeled by s'_{init} (which is of type 1) and the other by s_{gen}, we obtain (1), (2), and (3) in the Claim.

(4) follows from the definition of $\prec_{\mathcal{T}}$ and the above properties, and (5) follows from (4).

$\square$

If $level_{\mathcal{T}}(n) = \{v_1, \ldots, v_{n+1}\}$ of an s_{init}-rooted computation tree $\mathcal{T}$ of $\mathcal{G}$ under σ has exactly $n+1$ nodes, then we say that it is *complete*. Moreover, if we assume that $v_1 \prec_{\mathcal{T}} \cdots \prec_{\mathcal{T}} v_{n+1}$, then we may view $level_{\mathcal{T}}(n)$ as a sequence of nodes, $v_1 \cdots v_{n+1}$.

Claim 3. *Let $\mathcal{T} = (V, E, v_0, l_1, l_2)$ be an s_{init}-rooted computation tree of $\mathcal{G}$ under σ, and $n \geq 1$ such that $level_{\mathcal{T}}(n)$ is complete and its sequence of nodes is $v_1 \cdots v_{n+1}$. Then, the following properties hold:*

1. *$level_{\mathcal{T}}(m)$ is complete, for any $m \leq n$;*
2. *v_1 is of type 1, v_{2i} is of type $2(i)(i-1)$, and v_{2i+1} is of type $2(i)(i)$, for all $i \geq 1$ with $2i \leq n$;*
3. (a) *$l_1(path_{\mathcal{T}}(v_0, v_1)) \sim_2 l_1(path_{\mathcal{T}}(v_0, v_2))$;*
   (b) *$l_1(path_{\mathcal{T}}(v_0, v_{2i})) \sim_1 l_1(path_{\mathcal{T}}(v_0, v_{2i+1}))$, for all $i \geq 1$ with $2i \leq n$;*
   (c) *$l_1(path_{\mathcal{T}}(v_0, v_{2i+1})) \sim_2 l_1(path_{\mathcal{T}}(v_0, v_{2(i+1)}))$, for all $i \geq 1$ with $2i+1 \leq n$;*
4. *$l_1(v_1 \cdots v_{n+1})$ is of the one of the following forms:*
   (a) *$s'_{init} s_{gen}$, if $n = 1$;*
   (b) *$s_{lb} s_B s_{tr}$, if $n = 2$;*
   (c) *$s'_{lb} s_{a_1} s'_{tr} \cdots s_{a_{j-1}} s'_{tr} s_{q, a_j} s_{tr'} s_{a_{j+1}} \cdots s'_{tr} s_{a_m} s'_{tr} s_{gen}$, if $n > 2$ is odd, where $a_1, \ldots, a_m \in \Sigma$, $q \in Q$, $m = (n-1)/2$, and $1 \leq j \leq m$ (for $j = 1$, s_{a_1} becomes s_{q, a_1}, and for $j = m$, a_m becomes s_{q, a_m});*
   (d) *$s'_{lb} s_{a_1} s'_{tr} \cdots s_{a_{j-1}} s'_{tr} s_{a_j} s_{q, q', X} s_{a_{j+1}} \cdots s'_{tr} s_{a_{m-1}} s'_{tr} s_B s_{tr}$, if $n > 2$ is even, where $a_1, \ldots, a_{m-1} \in \Sigma$, $q, q' \in Q$, $X \in \{L, R\}$, $m = n/2$, and $1 \leq j \leq m-1$;*
5. *there exists an s_{init}-rooted computation tree $\mathcal{T}'$ of $\mathcal{G}$ under σ such that $\mathcal{T} \overset{*}{\Rightarrow}_{\mathcal{G}, \sigma} \mathcal{T}'$ and $level_{\mathcal{T}'}(n+1)$ is complete. Moreover, if the sequence of nodes of $level_{\mathcal{T}}(n)$ has the form (4a) ((4b), (4c), (4d)), then $level_{\mathcal{T}'}(n+1)$ has the form (4b) ((4c), (4d), (4c), respectively).*

14

PROOF. (1), (2), and (3) can be proved in a similar way to the statements in Claim 2.

We prove (4) and (5) together. It is easy to show that $l_1(v_1 \cdots v_{n+1})$ has the form (4a) if $n = 1$. As $l_1(path_{\mathcal{T}}(v_0, v_1)) \sim_2 l_1(path_{\mathcal{T}}(v_0, v_2))$ and $\mathcal{T}$ has only ok-states, the strategy σ_2 should select only $idle$ as the only choice for agent 2 at $l_1(v_1)$ and $l_1(v_2)$. σ_1 should select $idle$ for agent 1 at $l_1(v_1)$ and $l_1(v_2)$, while the agent 3 has the only choice $idle$ at $l_1(v_1)$ and two choices, br_1 and br_2, at $l_1(v_2)$. Therefore, we can extend $\mathcal{T}$ by adding a new descendant v_1' of v_1 and two new descendants v_2' and v_2'' of v_2, by the rules

$$l_1(v_1) \xrightarrow{(i,i,i)} l_1(v_1') = s_{lb}, \quad l_1(v_2) \xrightarrow{(i,i,br_1)} l_1(v_2') = s_B, \quad l_1(v_2) \xrightarrow{(i,i,br_2)} l_1(v_2'') = s_{tr}.$$

We obtain a new s_{init}-rooted computation tree $\mathcal{T}'$ of $\mathcal{G}$ under σ whose level 2 satisfies (4) and (5).

Assume $n = 2$ and $l_1(v_1, v_2, v_3) = s_{lb}s_Bs_{tr}$. As $l_1(path_{\mathcal{T}}(v_0, v_1)) \sim_2 l_1(path_{\mathcal{T}}(v_0, v_2))$ and $\mathcal{T}$ has only ok-states, the strategy σ_2 should select only (q_0) as the only choice for agent 2 at $l_1(v_1)$ and $l_1(v_2)$. The agents 1 has the only choice $idle$ at $l_1(v_1)$ and $l_2(v_2)$ (by σ_1), and the agent 3 has the same choice at these states (by d_3). Therefore, we can add a new descendant v_1' of v_1 and a new descendant v_2' of v_2 by the rules

$$l_1(v_1) = s_{lb} \xrightarrow{(i,(q_0),i)} l_1(v_1') = s_{lb}' \quad \text{and} \quad l_1(v_2) = s_B \xrightarrow{(i,(q_0),i)} l_1(v_2') = s_{q_0,B}.$$

There are two choices at $l_1(v_3)$, namely (i, i, br_1) and (i, i, br_2), allowing to add two descendants v_3' and v_3'' of v_3 on the next level. Moreover, $l_1(v_3') = s_{tr}'$ and $l_1(v_3'') = s_{gen}$. As a conclusion, $\mathcal{T}$ can be extended to a new tree $\mathcal{T}'$ whose sequence of nodes on level 3 are $v_1'v_2'v_3'v_3''$ and $l_1(v_1'v_2'v_3'v_3'') = s_{lb}'s_{q_0,B}s_{tr}'s_{gen}$ which is the form (4c). Moreover, (5) holds too.

Assume $n > 2$ odd, $l_1(v_1 \cdots v_{n+1})$ of the form (4c), and $j > 1$ (the case $j = 1$ can be discussed in a similar way). We have that $l_1(v_{2j}) = s_{q,a_j}$ and $l_1(v_{2j-1}) = l_1(v_{2j+1}) = s_{tr}'$. Due to the fact that $l_1(path_{\mathcal{T}}(v_0, v_{2j})) \sim_1 l_1(path_{\mathcal{T}}(v_0, v_{2j+1}))$ and $\mathcal{T}$ has only ok-states, σ_1 should select an action of the form (q, q', R) or (q, q', L) for agent 1 as a choice at $l_1(v_{2j})$ and $l_1(v_{2j+1})$ ($q' \in Q$ and this choice is obtained from the transition function of M). Assume that this choice is (q, q', R) and $\delta(q, a_j) = (q', a_j', R)$ (the other case is similar to this). Each of the agents 2 and 3 has exactly one choice at $l_1(v_{2j})$ and $l_1(v_{2j+1})$, namely $idle$. Therefore, $\mathcal{T}$ can be extended by adding two new descendants v_{2j}' and v_{2j+1}' by the rules

$$l_1(v_{2j}) = s_{q,a_j} \xrightarrow{((q,q',R),i,i)} l_1(v_{2j}') = s_{a_j'} \quad \text{and} \quad l_1(v_{2j+1}) = s_{tr}' \xrightarrow{((q,q',R),i,i)} l_1(v_{2j+1}') = s_{q,q',R}.$$

For the nodes v_i with $i \notin \{2j, 2j+1, n+1\}$, there is exactly one choice for each agent, namely $idle$, and therefore, a new descendant v_i' of v_i can be added by the rule

$$l_1(v_i) \xrightarrow{(i,i,i)} l_1(v_i') = l_1(v_i).$$

For the node v_{n+1} we may reason as in the case $n = 2$ above. Two descendants v_{n+1}' and v_{n+1}'' can be added, with $l_1(v_{n+1}') = s_B$ and $l_1(v_{n+1}'') = s_{tr}$.

In this way, we obtain a new tree $\mathcal{T}'$ whose level $n + 1$ satisfies (4) and (5).

The case "$n > 2$ even and $l_1(v_1 \cdots v_{n+1})$ of the form (4d)" can be treated analogously to the above one. □

Consider further the homomorphism $h : S \to (Q \cup \Sigma)^*$ given by:

$$h(s) = \begin{cases} a, & \text{if } s = s_a \\ qa, & \text{if } s = s_{q,a} \\ \lambda, & \text{otherwise} \end{cases}$$

We shall write $h(level_{\mathcal{T}}(n))$ for $h(v_1 \cdots v_{n+1})$, where $v_1 \cdots v_{n+1}$ is the sequence of nodes associated to complete level $level_{\mathcal{T}}(n)$ of some s_{init}-rooted computation tree $\mathcal{T}$ of $\mathcal{G}$ under σ.

Claim 4. *Let $\mathcal{T} = (V, E, v_0, l_1, l_2)$ be an s_{init}-rooted computation tree of $\mathcal{G}$ under σ, and $n \geq 3$ odd such that $level_{\mathcal{T}}(n)$ is complete. Then:*

1. *$h(level_{\mathcal{T}}(n)) \in \Sigma^* Q \Sigma \Sigma^*$;*
2. *there exists an s_{init}-rooted computation tree $\mathcal{T}'$ of $\mathcal{G}$ under σ such that $\mathcal{T} \overset{*}{\Rightarrow}_{\mathcal{G},\sigma} \mathcal{T}'$, $level_{\mathcal{T}'}(n+2)$ is complete, and $h(level_{\mathcal{T}}(n)) \Rightarrow_M h(level_{\mathcal{T}'}(n+2))$.*

PROOF. From the definition of h, Claim 3, and by inspecting the proof of Claim 3. □

It is straightforward to see that there exists an s_{init}-rooted computation tree $\mathcal{T}$ of $\mathcal{G}$ under σ whose $level_{\mathcal{T}}(3)$ is complete. Moreover, by Claim 3, we have $h(level_{\mathcal{T}}(3)) = q_0 B$ (that is, the initial configuration of M). Then, combining with Claim 4, we obtain that M does not halt on the empty word if all s_{init}-rooted computation trees of $\mathcal{G}$ under some strategy σ for $\{1,2\}$ have only nodes labeled by ok-states.

Our discussion above leads to:

Theorem 1. *The model checking problem for ATL_{iR} is undecidable.*

4. Conclusions

The proof above shows that the strategies used by the agents 1 and 2 to simulate the deterministic Turing machine M are primitive recursive. Therefore, the crucial elements which allow to simulate M are the equivalence relations $\sim_1$ and $\sim_2$. These equivalence relations are "inter-related" and are used to transfer information from one computation path can be transferred to another computation path.

A deeper analysis of the nature of the observational equivalence relations associated to agents in a CGS would be interesting.

References

[1] R. Alur, Th. A. Henzinger, O. Kupferman. *Alternating-time Temporal Logic*, Journal of the ACM 49, 2002, 672–713. Preliminary version appeared in the Proc. of the 38th IEEE Symposium on Foundations of Computer Science (FOCS '97), 1997, 100-109.

[2] N. Bulling, J. Dix, W. Jamroga. *Model Checking Logics of Strategic Ability: Complexity*, in "Specification and Verification of Multi-Agent Systems" (M. Dastani, K. Hindriks, J.-J. Meyer, eds.), Springer-Verlag, 2010 (to appear).

[3] P.-Y. Schobbens. *Alternating-time Logic with Imperfect Recall*, Electronic Notes in theoretical Computer Science 85(2), 2004.

[4] M. Yannakakis. *Synchronous Multi-player Games with Incomplete Information are Undecidable*, personal communication, 1997.

[5] W. Jamroga and Th. Agotnes. *Constructive Knowledge: What Agents Can Achieve under Imperfect Information*, Journal of Applied Non-Classical Logics, 17, (4), p. 423–475, 2007.

[6] Fr. Laroussinie, N. Markey and Gh. Oreiby, *On the Expressiveness and Complexity of ATL*, Logical Methods in Computer Science, 4(2), 2008.