Improving the Model Checking of Strategies under Partial Observability and Fairness Constraints

Simon Busard[1]*, Charles Pecheur[1], Hongyang Qu[2], and Franco Raimondi[3]

[1] ICTEAM Institute, Université catholique de Louvain, Louvain-la-Neuve, Belgium,
{simon.busard,charles.pecheur}@uclouvain.be
[2] Dept. of Automatic Control and Systems Engineering, University of Sheffield,
Sheffield, United Kingdom, h.qu@sheffield.ac.uk
[3] Dept. of Computer Science, Middlesex University, London, United Kingdom,
f.raimondi@mdx.ac.uk

Abstract. Reasoning about strategies has been a concern for several years, and many extensions of Alternating-time Temporal Logic have been proposed. One extension, $ATLK_{irF}$, allows the user to reason about the strategies of the agents of a system under partial observability and unconditional fairness constraints. However, the existing model-checking algorithm for $ATLK_{irF}$ is inefficient when the user is only interested in the satisfaction of a formula in a small subset of states, such as the set of initial states of the system. We propose to generate fewer strategies by only focusing on partial strategies reachable from this subset of states, reducing the time needed to perform the verification. We also describe several practical improvements to further reduce the verification time and present experiments showing the practical impact of the approach.

1 Introduction

Logics to reason about the strategies of a group of agents have been studied for years and they have a number of practical applications, from security to synthesis of plans to achieve a certain goal. Starting with Alternating-time Temporal Logic (ATL), reasoning about all strategies of the agents [1], many extensions have been developed. For example, ATL_{ir} restricts the strategies of interest to those that the players can actually play, based on their local knowledge of the system [2]. $ALTK_{irF}$ [3] is another extension that combines strategies under partial observability and unconditional fairness constraints, with branching-time and epistemic operators. This logic can be used, for example, to verify strategic properties of multi-agent programs in the presence of a fair scheduler [4]. However, the basic algorithm proposed in [3] is inefficient when the user is interested in the existence of a winning strategy in a small subset of the states of the system, such as the initial states, instead of all the states of the system.

* This work is supported by the European Fund for Regional Development and by the Walloon Region.

The objective of this paper is to improve the practical efficiency of the algorithm presented in [3] by checking fewer strategies. Let us consider the following simple motivational example of a 3-card poker, inspired by the card game of [5]: the system is a card game played between two agents, a player and a dealer. The game is composed of three cards: the ace A, the king K and the queen Q; the ace wins over the two others and the king wins over the queen. The game is played in three steps: 1) the dealer gives a card to the player; he also takes one for himself and keeps it secret; 2) the player can abandon the game or continue; 3) the player can choose to keep his card or to swap it with the third one. If the third step is reached—the player did not abandon after the first step—the winner is the one with the winning card, and the game restarts from the beginning. The graph of the system is illustrated in Figure 1.

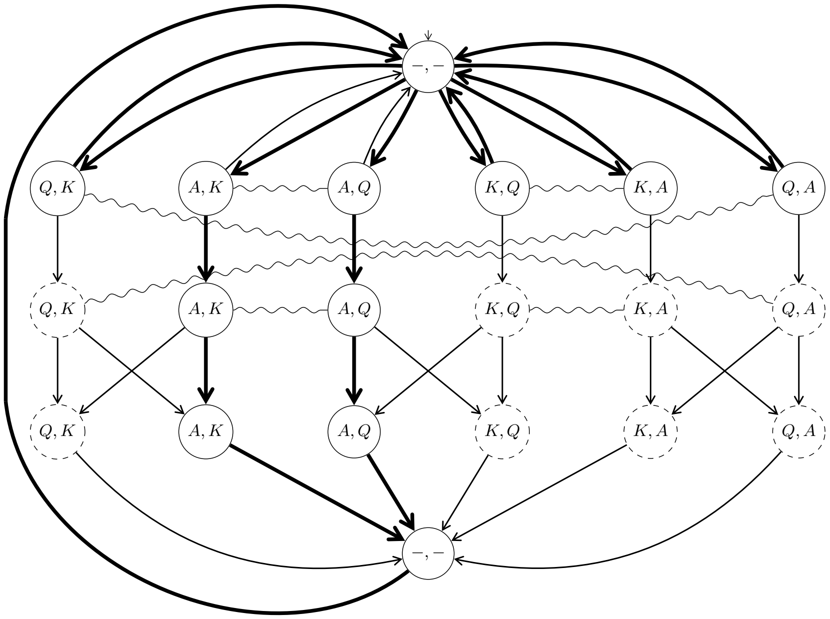

Fig. 1. The graph of the card game. Circles are states, (K, A means the player has K, the dealer has A). Arrows are transitions (actions of the agents are easily inferred). Waved edges link together the states that are indistinguishable for the player.

We are interested in whether the player has a strategy to eventually win the game before the dealer. Intuitively, to consider all strategies of the player in the initial state, we have to consider his choices at the second step—abandoning or continuing the game—and, if he chooses to continue the game, his choices at the third step—keeping or swapping his card. But if he chooses to abandon when he does not receive the ace, we do not need to consider his choice at third step when his card is the king or the queen. This amounts to considering 27 strategies. Figure 1 shows such a strategy in bold and the states in which we do not need to consider the player's choices with dashed borders.

On the other hand, the algorithm presented in [3] blindly makes a choice for all possible sets of indistinguishable states of the system. In the present case, it considers whether the player keeps or swaps his king (or queen) at the third step, even if the considered strategy is to abandon the game at step 2; that is, the algorithm enumerates the strategies in the dashed states of Figure 1, too, considering 64 strategies. The contributions of this paper are:

- an algorithm to generate only strategies that are relevant for given states;
- an new model-checking algorithm for $ATLK_{irF}$ based on these strategies;
- further practical improvements of this algorithm—for example, stopping when a winning strategy has been found instead of checking them all;
- an implementation of these algorithms (to the best of our knowledge, this is the only implementation currently available);
- experiments showing the benefits of the approach.

The paper is organized as follows: Section 2 briefly describes $ATLK_{irF}$ and the original model-checking algorithm; Section 3 proposes an algorithm to generate fewer strategies and Section 4 presents the new model-checking algorithm; Section 5 describes further improvements to the approach, and Section 6 presents experiments made on the implementation of the approach. The proof of theorems are omitted due to space constraints.

2 Background

This section presents the syntax and the semantics of $ATLK_{irF}$. This logic has been presented in [3], where it is called $ATLK^{F}_{po}$. This section also describes the original model-checking algorithm for $ATLK_{irF}$ proposed in [3] and gives an intuition of how to use partial strategies—that is, strategies that give actions for a subset of the state space—to improve the algorithm.

Syntax and semantics. Formulas of $ATLK_{irF}$ are built from a set of atomic propositions AP, standard Boolean connectives, CTL operators [6], epistemic operators [7] and strategic operators [1]. They follow this grammar:

$$\begin{aligned}\phi &::= true \mid p \mid \neg\phi \mid \phi \vee \phi \mid E\psi \mid \langle\Gamma\rangle\psi \mid K_i\phi \mid E_\Gamma\phi \mid D_\Gamma\phi \mid C_\Gamma\phi\\ \psi &::= X\phi \mid \phi\ U\ \phi \mid \phi\ W\ \phi\end{aligned}$$

where $p \in AP$, Γ is a subset of a set of agents Ag, i is an agent of Ag.

Models and notations. $ATLK_{irF}$ formulas are interpreted over states of models $M = \langle Ag, S, Act, T, I, \{\sim_i\}_{i\in Ag}, V, FC\rangle$ where (1) Ag is a set of n agents; (2) S is a set of states; (3) $Act \subseteq Act_1 \times ... \times Act_n$ is a set of joint actions (one action for each agent); (4) $T \subseteq S \times Act \times S$ is a transition relation giving at least one successor for each state (we write $s \xrightarrow{a} s'$ for $(s, a, s') \in T$); (5) $I \subseteq S$ is the set of initial states; (6) $\{\sim_i\}_{i\in Ag}$ is a set of equivalence relations on $S \times S$, one for each agent (we write $\sim_\Gamma$ for $\bigcap_{i\in\Gamma} \sim_i$, the *distributed knowledge* relation of agents in $\Gamma \subseteq Ag$); (7) $V : S \to 2^{AP}$ is a function labeling the states of M with atomic propositions of AP; (8) $FC \subseteq 2^S$ is a set of fairness constraints.

The function $img : S \times Act \to 2^S$ returning the set of states accessible from a given state through a given action is defined as $img(s, a) = \{s' \in S | s \xrightarrow{a} s'\}$. Furthermore, the set of states that are indistinguishable for Γ from the states of Z is defined as $[Z]_\Gamma = \{s' | \exists s \in Z \text{ s.t. } s' \sim_\Gamma s\}$.

A partially joint action is an element a_Γ of $Act_\Gamma = \prod_{i \in \Gamma} Act_i$; we say that action $a \in Act$ completes a_Γ, written $a_\Gamma \sqsubseteq a$, if the actions of agents of Γ in a_Γ correspond to the actions of Γ in a. The function $enabled : S \times Ag \to 2^{Act}$ returning the actions a group of agents can perform in a state is defined as

$$enabled(s, \Gamma) = \{a_\Gamma \in Act_\Gamma | \exists s' \in S, a \in Act \text{ s.t. } a_\Gamma \sqsubseteq a \wedge s \xrightarrow{a} s'\}. \quad (1)$$

Two additional constraints are set on the models:

$$\forall s, s' \in S, s \sim_i s' \implies enabled(s, i) = enabled(s', i), \quad (2)$$

$$\forall s \in S, enabled(s, Ag) = \prod_{i \in Ag} enabled(s, i). \quad (3)$$

They ensure that an agent only needs its own information about the current state to make a choice (2), and that nobody can prevent him from choosing an enabled action (3).

A path is a sequence $\pi = s_0 \xrightarrow{a_1} s_1 \xrightarrow{a_2} ...$ such that $(s_i, a_{i+1}, s_{i+1}) \in T$ for all $i \geq 0$. We write $\pi(d)$ for s_d. A path π is fair according to FC if for each fairness constraint $fc \in FC$, there exist infinitely many d such that $\pi(d) \in fc$.

A memoryless strategy for Γ is a function $f_\Gamma : S \to Act_\Gamma$ such that $\forall s, f_\Gamma(s) \in enabled(s, \Gamma)$, specifying, for each state of the model, which action group Γ has to choose in each state. A strategy f_Γ is uniform iff $\forall s, s' \in S, s \sim_\Gamma s' \implies f_\Gamma(s) = f_\Gamma(s')$. In the sequel, we only speak about memoryless uniform strategies, and simply call them strategies. The outcomes $out(s, f_\Gamma)$ of a strategy f_Γ from state s is the set of paths reached by f_Γ from s and is defined as

$$out(s,f_\Gamma) = \left\{\pi = s_0 \xrightarrow{a_1} s_1 ... \middle| \begin{array}{l} s_0 = s \wedge \\ \forall d \geq 0, s_{d+1} \in img(s_d, a_{d+1}) \wedge f_\Gamma(s_d) \sqsubseteq a_{d+1} \end{array}\right\}. \quad (4)$$

Finally, a move of Γ is a state/action pair, that is, an element of $S \times Act_\Gamma$. A strategy f_Γ can be represented as a set of moves as

$$\{\langle s, a_\Gamma \rangle | s \in dom(f_\Gamma) \wedge a_\Gamma = f_\Gamma(s)\}, \quad (5)$$

that is, the set of moves $\langle s, a_\Gamma \rangle$ such that s is a state for which f_Γ is defined and a_Γ is the action that f_Γ chooses.

Semantics. The semantics of $ATLK_{irF}$ is defined over states of a model M as the relation $M, s \models \phi$, where s is a state of M and ϕ is a formula of the logic. This relation is defined in the standard way for atomic propositions, Boolean connectors, branching-time and epistemic operators. The semantics of strategic operators is defined as

$$M, s \models \langle \Gamma \rangle \psi \iff \begin{array}{l} \text{there exists a strategy } f_\Gamma \text{ s.t.} \\ \forall s' \sim_\Gamma s, \forall \textbf{ fair} \text{ paths } \pi \in out(s', f_\Gamma), M, \pi \models \psi, \end{array} \quad (6)$$

where the relation $M, \pi \models \psi$ is defined as

$$M, \pi \models X\phi \iff M, \pi(1) \models \phi; \tag{7}$$

$$M, \pi \models \phi_1 U \phi_2 \iff \exists d \geq 0 \text{ s.t. } M, \pi(d) \models \phi_2 \wedge \forall e < d, M, \pi(e) \models \phi_1; \tag{8}$$

$$M, \pi \models \phi_1 W \phi_2 \iff \forall d \geq 0, M, \pi(d) \models \phi_1 \vee \exists e \leq d \text{ s.t. } M, \pi(e) \models \phi_2. \tag{9}$$

Note that the remaining strategic operators can be expressed in terms of the previous three operators: $[\mathit{\Gamma}]\psi = \neg\langle\mathit{\Gamma}\rangle\neg\psi$, $G\phi = \phi\ W\ \mathit{false}$ and $F\phi = \mathit{true}\ U\ \phi$.

Due to space constraints, we only focus on strategic operators in this paper, but our approach can be employed for the remaining operators (our implementation has all the operators).

Standard model-checking algorithm. The original algorithm consists in enumerating all the strategies of the model and accumulating, for each of them, the set of states for which the strategy is winning. Algorithm 1 is the original algorithm for evaluating the set of states satisfying a strategic operator; it uses Algorithm 2 for computing the set of strategies of the model as sets of moves and the function $\mathit{eval}_{IrF}(\langle\mathit{\Gamma}\rangle\psi, f_{\mathit{\Gamma}})$ for computing the set of states for which strategy $f_{\mathit{\Gamma}}$ is winning on ψ for $\mathit{\Gamma}$. The function $\mathit{eval}_{IrF}(\langle\mathit{\Gamma}\rangle\psi, f_{\mathit{\Gamma}})$ relies on the function $\mathit{Pre}_{\langle\mathit{\Gamma}\rangle}(Z, f_{\mathit{\Gamma}})$, defined as, given $f_{\mathit{\Gamma}} \subseteq S \times \mathit{Act}_{\mathit{\Gamma}}$ and $Z \subseteq S$,

$$\mathit{Pre}_{\langle\mathit{\Gamma}\rangle}(Z, f_{\mathit{\Gamma}}) = \{s | \forall a, f_{\mathit{\Gamma}}(s) \sqsubseteq a \implies \mathit{img}(s, a) \subseteq Z\}. \tag{10}$$

$\mathit{Pre}_{\langle\mathit{\Gamma}\rangle}(Z, f_{\mathit{\Gamma}})$ computes the set of states for which $\mathit{\Gamma}$ can force to reach states of Z in one step, by using the actions provided by $f_{\mathit{\Gamma}}$. eval_{IrF} is defined using fix-point operations as

$$\mathit{eval}_{IrF}(\langle\mathit{\Gamma}\rangle X\phi, f_{\mathit{\Gamma}}) = \mathit{Pre}_{\langle\mathit{\Gamma}\rangle}\big(\mathit{eval}_{irF}(\phi) \cup \mathit{NFair}_{\langle\mathit{\Gamma}\rangle}(f_{\mathit{\Gamma}}), f_{\mathit{\Gamma}}\big) \tag{11}$$

$$\begin{aligned} &\mathit{eval}_{IrF}(\langle\mathit{\Gamma}\rangle\phi_1 U \phi_2, f_{\mathit{\Gamma}}) = \\ &\mu Z.\mathit{\Phi} \cap \left(\mathit{\Phi}_2 \ \cup \bigcup_{fc \in FC} \mathit{Pre}_{\langle\mathit{\Gamma}\rangle}\left(\nu Y. \begin{matrix} \mathit{\Phi} \cap (Z \cup \overline{fc}) \cap \\ \big(\mathit{\Phi}_2 \ \cup \ \mathit{Pre}_{\langle\mathit{\Gamma}\rangle}(Y, f_{\mathit{\Gamma}})\big) \end{matrix}, f_{\mathit{\Gamma}}\right)\right) \end{aligned} \tag{12}$$

$$\mathit{eval}_{IrF}(\langle\mathit{\Gamma}\rangle\phi_1 W \phi_2, f_{\mathit{\Gamma}}) = \nu Z.\mathit{\Phi} \cap \big(\mathit{\Phi}_2 \cup \mathit{Pre}_{\langle\mathit{\Gamma}\rangle}(Z, f_{\mathit{\Gamma}})\big) \tag{13}$$

where

$$\overline{fc} = S \backslash fc, \tag{14}$$

$$\mathit{\Phi} = \mathit{eval}_{irF}(\phi_1) \cup \mathit{eval}_{irF}(\phi_2) \cup \mathit{NFair}_{\langle\mathit{\Gamma}\rangle}(f_{\mathit{\Gamma}}), \tag{15}$$

$$\mathit{\Phi}_2 = \mathit{eval}_{irF}(\phi_2), \tag{16}$$

$$\mathit{NFair}_{\langle\mathit{\Gamma}\rangle}(f_{\mathit{\Gamma}}) = \mu Z. \bigcup_{fc \in FC} \mathit{Pre}_{\langle\mathit{\Gamma}\rangle}(\nu Y.(Z \cup \overline{fc}) \cap \mathit{Pre}_{\langle\mathit{\Gamma}\rangle}(Y, f_{\mathit{\Gamma}}), f_{\mathit{\Gamma}}). \tag{17}$$

Given a set of moves SA, Algorithm 2 produces all the strategies only composed of moves of SA. When Algorithm 1 uses $\mathit{Split}(S \times \mathit{Act}_{\mathit{\Gamma}})$ at Line 2, it gets all the strategies of the whole model.

Algorithm 1: $eval_{irF}(\langle\Gamma\rangle\psi)$

```
Data: $\Gamma$ a set of agents of a model $M$, $\psi$ an $ATLK_{irF}$ path formula.
Result: The set of states of $M$ satisfying $\langle\Gamma\rangle\psi$.
$sat = \{\}$
for $f_\Gamma \in Split(S \times Act_\Gamma)$ do
    $winning = eval_{IrF}(\langle\Gamma\rangle\psi, f_\Gamma)$
    $sat = sat \cup \{s \in winning | \forall s' \sim_\Gamma s, s' \in winning\}$
return $sat$
```

The goal of $eval_{irF}(\langle\Gamma\rangle\psi)$ is to compute the set of states of the system that satisfy $\langle\Gamma\rangle\psi$, that is, the set of states for which there exists a winning strategy. For this, the algorithm has to produce and check all strategies of the entire model. But when we only need to know if some states satisfy the formula—for example, when we want to know if the initial states of the model satisfy $\langle\Gamma\rangle\psi$—we can improve this algorithm by only checking the partial strategies reachable from these states. We say that a strategy is partial if it provides moves for a subset of the states of the model.

Algorithm 2: $Split(SA)$

```
Data: $\Gamma$ a given (implicit) subset of agents, $SA \subseteq S \times Act_\Gamma$.
Result: The set of all the strategies $f_\Gamma$ composed only of moves of $SA$.
$conflicting = \{\langle s, a_\Gamma\rangle \in SA | \exists \langle s', a'_\Gamma\rangle \in SA \ s.t.\ s' \sim_\Gamma s \wedge a_\Gamma \neq a'_\Gamma\}$
if $conflicting = \emptyset$ then return $\{SA\}$

else
    $\langle s, a_\Gamma\rangle$ = pick one element in $conflicting$
    $equivalent = \{\langle s', a'_\Gamma\rangle \in SA | s' \sim_\Gamma s\}$
    $actions = \{a'_\Gamma \in Act_\Gamma | \exists \langle s, a'_\Gamma\rangle \in equivalent\}$
    $substrats = Split(SA \backslash equivalent)$
    $strats = \{\}$
    for $a_\Gamma \in actions$ do
        $equivStrat = \{\langle s', a'_\Gamma\rangle \in equivalent | a'_\Gamma = a_\Gamma\}$
        $strats = strats \cup \{equivStrat \cup substrat | substrat \in substrats\}$
    return $strats$
```

If a state in not reachable from the initial states through a given strategy, then it is useless to consider all the possible choices in this state, since no particular choice will modify the fact that the strategy is winning or not in the initial states. This is illustrated with the example in Figure 2. There are eight possible strategies, choosing one action per state; but if a strategy chooses action (1) in s_0, then the choice made in s_2 is irrelevant regarding the fact that the strategy is winning or not for s_0 because s_2 is not reachable from s_0 in this strategy. In fact, there are only four (partial) strategies to check to know if the initial state satisfies a given $\langle\Gamma\rangle\psi$ formula:

1. $\langle s_0, (1)\rangle, \langle s_1, (1)\rangle, \langle s_3, (1)\rangle$;
2. $\langle s_0, (1)\rangle, \langle s_1, (2)\rangle, \langle s_4, (1)\rangle$;
3. $\langle s_0, (2)\rangle, \langle s_2, (1)\rangle, \langle s_5, (1)\rangle$;
4. $\langle s_0, (2)\rangle, \langle s_2, (2)\rangle, \langle s_6, (1)\rangle$.

These partial strategies cover all the ways the agent can act from the initial state and are sufficient to know whether the initial state satisfies a strategic formula.

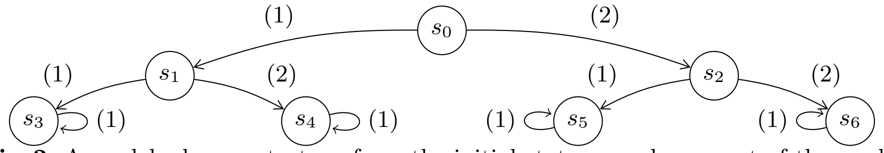

Fig. 2. A model where a strategy from the initial state s_0 makes a part of the model unreachable.

3 Generating partial strategies

This section presents the notions of partial strategies and maximal partial strategies, and shows how to generate them. A partial strategy is a strategy that is defined for a subset of the states of the model.

Given a set of states $S' \subseteq S$, a partial strategy that contains a move for all $s \in S'$, and that contains a move for all states reachable from the moves it defines is called *maximal*. More formally, a partial strategy f_Γ is a maximal partial strategy reachable from S' iff

$$S' \subseteq dom(f_\Gamma) \wedge \forall \langle s, a_\Gamma \rangle \in f_\Gamma, \forall a \in Act, a_\Gamma \sqsubseteq a \implies img(s, a) \subseteq dom(f_\Gamma). \quad (18)$$

Such a strategy is uniform iff $\forall s, s' \in dom(f_\Gamma), s \sim_\Gamma s' \implies f_\Gamma(s) = f_\Gamma(s')$.

The main advantage of maximal partial strategies reachable from S' is that they can be used to check if there is a winning strategy for the states of S'.

Theorem 1. *Given a model $M = \langle Ag, S, Act, T, I, \{\sim_i\}_{i \in Ag}, V, FC\rangle$, a set of states $S' \subseteq S$ and a group of agents $\Gamma \subseteq Ag$, we have that for all $s \in S'$, $M, s \models \langle \Gamma \rangle \psi$ iff there exists a uniform maximal partial strategy f_Γ reachable from $[S']_\Gamma$ such that for all $s' \in [S']_\Gamma$, for all fair paths π of $out(s', f_\Gamma), M, \pi \models \psi$.*

Proof (Proof sketch). We can proof this theorem by showing that there exists a winning uniform strategy in $s' \in S'$ iff there exists a winning uniform maximal partial strategy reachable from $[S']_\Gamma$. Indeed, a strategy can be reduced to a partial one by removing unreachable moves, and a partial strategy can be augmented with moves in unreachable states, producing a complete strategy.

By Theorem 1, it is sufficient to check all maximal partial strategies reachable from $[S']_\Gamma$ to know whether there exists a winning strategy for the states of S'. Thus, if we are interested in the satisfaction of a strategic operator for only a subset of states S', it is sufficient to check the maximal partial strategies reachable from $[S']_\Gamma$. Before focusing on such a model-checking algorithm, we propose an algorithm to produce these maximal partial strategies.

Algorithm 3 can be used to generate the set of maximal partial strategies reachable from a set of states. It uses functions *Post*, *Compatible* and *Split*.

$\mathit{Post}(Z, f_\Gamma)$ is a version of the post-image computation modified to take actions present in f_Γ and states of Z into account. More formally, given $Z \subseteq S$ and a strategy $f_\Gamma \subseteq S \times \mathit{Act}_\Gamma$,

$$\mathit{Post}(Z, f_\Gamma) = \left\{ s \left| \begin{array}{l} \exists \langle s', a'_\Gamma \rangle \in f_\Gamma, a' \in \mathit{Act} \text{ s.t.} \\ s' \in Z \wedge a'_\Gamma \sqsubseteq a' \wedge s \in \mathit{img}(s', a') \end{array} \right. \right\}. \tag{19}$$

is the set of states reachable through a move of f_Γ from states of Z. *Compatible* is defined by

$$\mathit{Compatible}(Z, f_\Gamma) = \left\{ \langle s, a_\Gamma \rangle \left| \begin{array}{l} s \in Z \wedge a_\Gamma \in \mathit{enabled}(s, \Gamma) \wedge \\ \nexists \langle s', a'_\Gamma \rangle \in f_\Gamma \text{ s.t. } s \sim_\Gamma s' \wedge a_\Gamma \neq a'_\Gamma \end{array} \right. \right\}. \tag{20}$$

It returns the set of moves m, composed of states of Z and actions enabled in these states, such that m are not conflicting with any move of f_Γ. We say that two moves $\langle s, a_\Gamma \rangle$ and $\langle s', a'_\Gamma \rangle$ are conflicting iff $s \sim_\Gamma s'$ and $a_\Gamma \neq a'_\Gamma$, that is, if they propose different actions for states that are indistinguishable by Γ.

Given a partial strategy represented by a set of moves, Algorithm 3 returns the set of maximal partial strategies extending the given one. A partial strategy f'_Γ extends another partial strategy f_Γ if the choices made in f'_Γ match the choices in f_Γ, that is, if $f_\Gamma \subseteq f'_\Gamma$.

Algorithm 3: $\mathit{ReachSplit}_\Gamma(f_\Gamma)$

```
  Data: $\Gamma$ a subset of agents, $f_\Gamma \subseteq S \times \mathit{Act}_\Gamma$ a partial strategy.
  Result: The set of maximal strategies extending $f_\Gamma$.
1 $\mathit{new} = \mathit{Post}(\mathit{dom}(f_\Gamma), \mathit{strat}) \backslash (\mathit{dom}(f_\Gamma))$
  if $\mathit{new} = \emptyset$ then return $\{f_\Gamma\}$

  else
5     $\mathit{compatible} = \mathit{Compatible}(\mathit{new}, f_\Gamma)$
      $\mathit{newstrats} = \mathit{Split}(\mathit{compatible})$
      $\mathit{strats} = \{\}$
      for $f'_\Gamma \in \mathit{newstrats}$ do $\mathit{strats} = \mathit{strats} \cup \mathit{ReachSplit}_\Gamma(f_\Gamma \cup f'_\Gamma)$

      return $\mathit{strats}$
```

Algorithm 3 first gets the states reachable in one step from f_Γ that are not yet included in f_Γ (Line 1). These states are the states reachable in one step from f_Γ for which an action is not already chosen. If there are no such states, f_Γ is already maximal since a choice has already been made for each reachable state. Otherwise, we have to make some choices for *new* states. First, some uniform choices may have already been made through choices of f_Γ: if a new state s is indistinguishable from a state s' in f_Γ, the choice in s must follow the one in s'. Thus, we can remove from the choices possible in states of *new* all the choices that are conflicting with the ones in f_Γ (Line 5). After that, *compatible* can still contain conflicts, which are resolved by splitting *compatible* into strategies with *Split*. These strategies are compatible with f_Γ because all the potentially conflicting choices are removed at Line 5. Thus, any splitting f'_Γ of *compatible* combined with f_Γ is a partial strategy extending f_Γ and we can recursively call *ReachSplit* until all reachable states are encountered.

The correctness of Algorithm 3 is given by the following theorem.

Theorem 2. *Given a subset Γ of the agents of a model $M = \langle Ag, S, Act, T, I, \{\sim_i\}_{i \in Ag}, V, FC\rangle$ and a partial strategy represented by a set of moves f_Γ, the result of $ReachSplit_\Gamma(f_\Gamma)$ is the set of maximal strategies extending f_Γ.*

Finally, we can compute the set of maximal partial strategies reachable from S' by using *ReachSplit*: let *PartialStrats* be the function defined as

$$PartialStrats(S') = \bigcup\{ReachSplit_\Gamma(st) | st \in Split(Moves_\Gamma(S'))\}, \qquad (21)$$

where $Moves_\Gamma(Z) = \{\langle s, a_\Gamma\rangle | s \in Z \wedge a_\Gamma \in enabled(s, \Gamma)\}$ is the set of moves that Γ can play from states of Z. $PartialStrats(S')$ computes the set of maximal partial strategies reachable from S'.

Theorem 3. *Given a model $M = \langle Ag, S, Act, T, I, \{\sim_i\}_{i \in Ag}, V, FC\rangle$, a subset Γ of the agents of M and a subset S' of states of M, $PartialStrats(S')$ is the set of maximal partial strategies reachable from S'.*

4 Model checking $ATLK_{irF}$ with partial strategies

The number of partial strategies to consider to determine whether a group of agents Γ has a winning strategy in a subset of states S' can be substantially smaller than the overall number of strategies of the model (see Section 2). We can thus improve the model-checking algorithm for $ATLK_{irF}$ presented in Section 2 by using partial strategies. The idea is to only get the satisfaction of the formula in the states that matter, instead of getting it in all states of the system. For example, when checking whether a model satisfies $\langle\Gamma\rangle Fp$, we only need to know whether all the states indistinguishable from the initial states satisfy the formula, instead of knowing all states satisfying the formula. On the other hand, when checking $AG\langle\Gamma\rangle Fp$, we need to know whether all reachable states satisfy $\langle\Gamma\rangle Fp$ to say whether the formula is satisfied or not by all initial states.

Our algorithm keeps track of the set of states for which the satisfaction of the formula has to be known. Whenever an operator is evaluated, the algorithm is recursively called on the set of states in which the satisfaction of the top-level subformulas have to be known before evaluating the current operator. Given the initial states, the algorithm returns all the initial states satisfying the formula.

Given a set of states Z and a formula ϕ, Algorithm 4 returns the states of Z that satisfy ϕ. It works recursively on the structure of ϕ, and evaluates, on each step, the set of states in which it is necessary to know the satisfaction of the top-level subformulas. Due to space constraints, only the cases for strategic operators are presented. In these cases, the algorithm goes through all partial strategies reachable from Z and their indistinguishable states, and needs to know the satisfaction for the top-level subformulas in the states reachable by each partial strategy before computing the states of Z satisfying the main formula.

The goal of Algorithm 4 is to evaluate the satisfaction of the formula in as few states as possible. When dealing with strategic operators, the generation of partial strategies allows the algorithm to avoid a potentially large number of strategies. Note that, while it computes the partial strategies through a forward traversal of the model (see Section 3), it performs the evaluation of the states satisfying a given strategic operator with a backward traversal of the strategy.

```
Algorithm 4: $eval_{irF}^{Partial}(Z, \langle\Gamma\rangle\psi)$

Data: $Z \subseteq S$ a subset of states, $\langle\Gamma\rangle\psi$ an $ATLK_{irF}$ formula.
Result: The set of states of $Z$ satisfying $\langle\Gamma\rangle\psi$.

sat = {}
for $f_\Gamma \in PartialStrats([Z]_\Gamma)$ do
    case $\psi = X\phi'$
        $\Phi' = eval_{irF}^{Partial}(Post([Z]_\Gamma, f_\Gamma), \phi')$
        $win = Pre_{\langle\Gamma\rangle}(\Phi' \cup NFair_{\langle\Gamma\rangle}(f_\Gamma), f_\Gamma)$
    case $\psi = \phi_1 U \phi_2$
        $\Phi_1 = eval_{irF}^{Partial}(dom(f_\Gamma), \phi_1)$; $\Phi_2 = eval_{irF}^{Partial}(dom(f_\Gamma), \phi_2)$
        $win =$
        $\mu X.(\Phi_1 \cup \Phi_2 \cup NFair_{\langle\Gamma\rangle}(f_\Gamma)) \cap$
        $\left(\Phi_2 \cup \bigcup_{fc \in FC} Pre_{\langle\Gamma\rangle}\left(\nu Y. \begin{matrix}(\Phi_1 \cup \Phi_2 \cup NFair_{\langle\Gamma\rangle}(f_\Gamma)) \\ \cap (X \cup \overline{fc}) \cap \big(\Phi_2 \cup Pre_{\langle\Gamma\rangle}(Y, f_\Gamma)\big)\end{matrix}, f_\Gamma\right)\right)$
    case $\psi = \phi_1 W \phi_2$
        $\Phi_1 = eval_{irF}^{Partial}(dom(f_\Gamma), \phi_1)$; $\Phi_2 = eval_{irF}^{Partial}(dom(f_\Gamma), \phi_2)$
        $win = \nu X.(\Phi_1 \cup \Phi_2 \cup NFair_{\langle\Gamma\rangle}(f_\Gamma)) \cap \big(\Phi_2 \cup Pre_{\langle\Gamma\rangle}(X, f_\Gamma)\big)$
    $sat = sat \cup \{s \in win \cap Z | \forall s' \sim_\Gamma s, s' \in win\}$
return sat
```

Finally, to get the set of initial states satisfying an $ATLK_{irF}$ formula ϕ, we can simply use Algorithm 4 on these initial states. The following theorem proves the correctness of Algorithm 4:

Theorem 4. *Given a model $M = \langle Ag, S, Act, T, I, \{\sim_i\}_{i \in Ag}, V, FC\rangle$, a set of states $Z \subseteq S$ and an $ATLK_{irF}$ formula $\langle\Gamma\rangle\psi$, $eval_{irF}^{Partial}(Z, \langle\Gamma\rangle\psi)$ is the subset of states of Z that satisfy $\langle\Gamma\rangle\psi$.*

The strategies considered by $ATLK_{irF}$ are slightly different from the strategies of ATL_{ir} [2]. $ATLK_{irF}$ considers the agents of Γ under supervision of a *virtual supervisor*, as in the case of ATL_{iR}^{D}, a variant of ATL_{ir} using distributed knowledge, perfect recall and partial observability [8]. On the other hand, ATL_{ir} considers that each agent acts independently and does not share his knowledge with the other agents of Γ. Nevertheless, the approach of this paper can be easily adapted to fit ATL_{ir} strategies: the notion of conflicting moves needs to be changed to take into account the knowledge of each agent individually instead of as a group, and the *Compatible* and *Split* algorithms must be adapted accordingly.

5 Further optimisations

Several improvements can be added to Algorithm 4 to make it more efficient in common cases.

Checking fewer strategies by early termination. When dealing with a strategic operator $\langle \Gamma \rangle \psi$, $eval_{irF}^{Partial}$ goes through all partial strategies generated from $[Z]_\Gamma$ and accumulates in *sat* the subset of Z for which the current strategy is winning. The **for** loop at Line 2 could be terminated as soon as all states of Z are winning, that is, when $sat = Z$. In this case, we know that we found winning partial strategies for all states of Z and it is not necessary to check the remaining strategies. But if a state of Z does not satisfy $\langle \Gamma \rangle \psi$, all partial strategies must be checked. In the sequel, we call this improvement *full early termination*.

Following this idea, we could reconsider smaller strategies when *sat* grows. Indeed, when checking the strategies computed by $PartialStrats([Z]_\Gamma)$, we could recompute the smaller strategies reachable from $[Z]_\Gamma \backslash sat$ when states are added to *sat*, ignoring the part of these strategies taking *sat* states into account. This can be done by recomputing a new set of strategies whenever *sat* grows—we call this approach *partial early termination*.

We can also perform fewer recomputations of the strategies by recomputing them when the number of states of Z that are not in *sat* decreases under a certain threshold; the value given in the following is the threshold under which the part of the remaining states must be to trigger the recomputation of strategies. We call this approach *threshold-based early termination*.

The main drawback of the two last approaches is that parts of some strategies will be checked again, while we know they are not winning for the remaining states: indeed, when recomputing partial strategies for the remaining states, some of the new partial strategies will be parts of a partial strategy that has already been checked, and thus they cannot be winning for the remaining states.

Another approach to tackle this drawback would be to avoid recomputing the strategies and simply reduce the remaining ones to the moves reachable from the remaining states. In this case, we would need a mechanism to filter out the reduced strategies that are met multiple times. The current implementation (see next Section) only uses the approach of recomputing strategies.

Avoiding recomputation of subformulas with caching. When the model-checking algorithm deals with a strategic operator $\langle \Gamma \rangle \psi$, it enumerates all partial strategies reachable from $[Z]_\Gamma$ and, for each of them, first computes the set of states of the strategy satisfying the top-level subformula(s) of ψ. This can perform a lot of redundant work since several strategies can share the same subpart of the model. We can improve this by accumulating, for each subformula of ψ, their satisfaction value in encountered states.

Note that there is a difference between this approach and the standard caching techniques for BDD-based CTL model checking. BDD-based CTL model checking keeps track of BDDs representing states satisfying a property; these BDDs do not change for different occurrences of a subformula since they represent *all* the states satisfying it. This mechanism can not be used here because subsets of states of interest change for different strategies, thus BDDs change, and these new BDDs must be completely recomputed. The caching mechanism we propose is to only recompute satisfaction for new states, and keep the results in two accumulated BDDs, avoiding to recompute strategies for states for which it has already been done.

Pre-filtering out losing moves. A move is losing if it does not belong to a winning strategy. Experiments showed that pre-filtering out moves that are not winning under full observability can decrease the time needed to check a strategic operator [3]. We can include this improvement in $eval_{irF}^{Partial}$ by pre-filtering the state space reachable from $[Z]_\Gamma$ before building the partial strategies, and only consider the remaining submodel. This can lead to ignoring a large part of the system if this part cannot be winning, reducing the number of choices to make and the number of strategies to consider.

6 Experiments

The algorithm generating partial strategies shown in Section 3, the model-checking algorithm presented in Section 4 and the improvements discussed in Section 5 have been implemented with PyNuSMV, a Python framework for prototyping and experimenting with BDD-based model-checking algorithms based on NuSMV [9]. The implementation has been tested on two different models and several $ATLK_{irF}$ formulas.

The first model is another variant of the card game from [5]. The game is composed of two players—the player and the dealer—and n cards. The n cards c_1, ..., c_n are such that c_i wins over c_j if $i > j$ or $i = 1$ and $j = n$. The game is played in four steps: 1) the dealer gives one card to himself; 2) he gives one card to the player; 3) the player can choose to keep his card or to ask for another, but cannot get back a card he discarded before; 4) the game stops when the player chooses to keep his card or when the stack of cards is empty. The winner of the game, known during the last step, is the one with the winning card. The game can then be repeated infinitely many times and the dealer is fair, that is, if the game is repeated infinitely many times, the dealer gives the cards in each possible order infinitely many times.

The second model is inspired from the ancient tale of Tian Ji. It is composed of two agents: Tian Ji and the king. Both agents have n horses h_1, ..., h_n and horse h_i wins over h_j if $i > j$; if $i = j$, the winner is chosen non-deterministically. Their game is as follows: Tian Ji and the king go for n races, with n different horses. They can choose their own horses in the order they want, but do not know the horse the opponent chose. The winner is the one with the most won races. The game can then be repeated infinitely many times and the king is fair, that is, if the game is repeated infinitely many times, the king will choose his horses in each possible order infinitely many times.

Several $ATLK_{irF}$ formulas have been checked on each model to assess the impact of partial strategies and the improvements presented in Section 5. These formulas are listed in Table 1. They use different atomic propositions; for example, $playerWins$ is true when the game is done (at the fourth step) and the card of the player wins over the dealer's card; $playerHasFirst$ is true when the player has card c_1. Similarly, $tianjiWins$ is true when the game is done (all horses have been used) and Tian Ji won more races than the king; $tianjiLostUpToNow$ is true when Tian Ji has lost all races since the beginning of the game.

Table 1. Formulas checked over the models of the card game and Tian Ji's race.

Card game formulas	**Tian Ji's race formulas**
$\langle player\rangle F\ playerWins$	$\langle tianji\rangle F\ tianjiWins$
$\langle player\rangle F\ (playerWins \wedge playerHasFirst)$	
$\langle player\rangle G\ \langle player\rangle F\ playerWins$	$\langle tianji\rangle G\ \langle tianji\rangle F\ tianjiWins$
$\langle player\rangle F \langle player\rangle [\neg dealerWins\ U\ playerWins]$	$\langle tianji\rangle F\ \langle tianji\rangle [\neg kingWins\ U\ tianjiWins]$
$AF\langle player\rangle X\ playerHasFirst$	$\langle tianji\rangle X\ tianjiLostUpToNow$
$AG(FirstStep \implies \neg\langle player\rangle X\ playerWins)$	$\langle tianji\rangle G\ \langle tianji\rangle X\ tianjiWon < 2Races$

These formulas are intended to test the proposed algorithms under different circumstances. For example, the formula $\langle tianji\rangle F\ tianjiWins$ must only be checked over the initial states. In this case, partial strategies should help since the number of strategies to check substantially decreases. On the other hand, for the formula $\langle tianji\rangle G\ \langle tianji\rangle F\ tianjiWins$, the $\langle tianji\rangle F\ tianjiWins$ subformula must be evaluated on all states, and partial strategies do not help. Other formulas, like $AF\langle player\rangle X\ playerHasFirst$, are used to test the improvements presented in Section 5. In this case, the $\langle player\rangle X\ playerHasFirst$ subformula is true in a significantly small subset of the states, thus pre-filtering out losing moves before generating partial strategies should help.

A first set of tests have been performed to assess the efficiency of the model-checking algorithm of Section 4 compared to the original algorithm presented in Section 2. For each formula of Table 1, the size of the model to check has been increased and both model-checking algorithms have been run with a limit of 15 minutes. Some of the results are shown in Table 2. They show that using partial strategies can improve the efficiency of the process: for example, for the specification $\langle player\rangle F\ playerWins$, and even more for $\langle tianji\rangle F\ tianjiWins$, the time needed for the verification is significantly decreased. This is due to the fact that in these cases, we are interested in the existence of winning strategies in the initial states, and thus the number of strategies to consider is smaller. On the other hand, for the specification $AF\ \langle player\rangle X\ playerHasFirst$, we need to know the satisfaction of the inner strategic operator in all reachable states, and thus the verification does not run faster.

A second set of tests have been performed to assess the impact of the proposed improvements. Each formula of Table 1 has been checked with partial strategies on models of increasing sizes, with all combinations of improvements of Section 5 and with the same limit of 15 minutes. This resulted in a huge set of time results that have been analyzed with box plots. More precisely, for each formula, each improvement type and each size of the model, a box plot has been drawn showing the time results for each possible value of the improvement. For example, Figure 3 shows the box plots for times to check the formula $\langle tianji\rangle F\ tianjiWins$ for 3 to 5 horses, grouped by value of early termination. The box plots show, for a given parameter, the time needed for model checking the property when the parameter takes a particular value. This means that, in a box plot, a single box

Table 2. Execution times of the original algorithm and the algorithm based on partial strategies, for some formulas checked over the card game and the problem of Tian Ji.

Formula	**Size**	**# States**	**Original algorithm**	**Partial strategies**
$\langle player\rangle F\ playerWins$	3	28	$0m2.527s$	$0m2.603s$
	4	101	$0m8.035s$	$0m8.205s$
	5	326	$0m34.937s$	$0m30.885s$
	6	967	$2m14.461s$	$1m39.931s$
	7	2696	$> 15m$	$9m46.126s$
$\langle tianji\rangle F\ tianjiWins$	3	61	$0m5.388s$	$0m2.489s$
	4	409	$> 15m$	$0m25.172s$
$AF\langle player\rangle X\ playerHasFirst$	3	28	$0m1.506s$	$0m1.444s$
	6	967	$0m39.285s$	$0m38.535s$
	8	7177	$11m37.270s$	$12m9.149s$
	9	18442	$> 15m$	$> 15m$

represent the model-checking time when the other parameters vary. Thus, if a box is much lower than another, this means that whatever the other parameters are, the first parameter value gives better performances than the other.

From these box plots, we analyzed the effect of each improvement value on the time needed to model check a formula. For example, the box plots of Figure 3 show that when checking the formula $\langle tianji\rangle F\ tianjiWins$, early termination really decreases the time needed, but the kind of early termination used has no significant impact; this is expected since the formula is satisfied in all states with all strategies, and the model checking algorithm stops at the first strategy (instead of having to check all of them if early termination is deactivated).

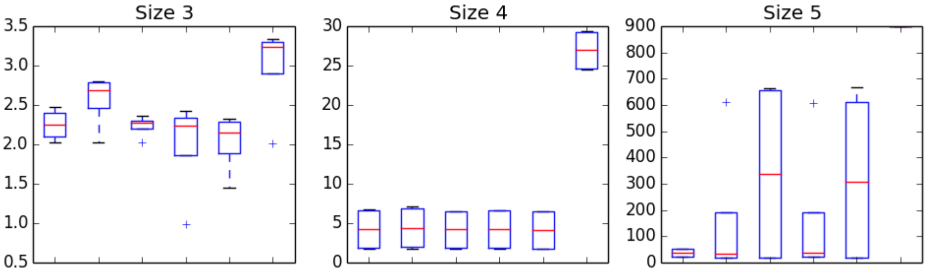

Fig. 3. Box plot of time (in seconds) needed for checking the formula $\langle tianji\rangle F\ tianjiWins$ with 3 to 5 horses. In each plot, boxes represent measures with different type of early termination (from left to right): threshold (trigger value: 0.9), partial, threshold (trigger value: 0.1), threshold (trigger value: 0.5), full, no early termination. On the third box plot, model checking exceeded the limit of 900 seconds for all checks without early termination.

The box plots of Figure 4 show that, when model checking the formula $\langle tianji\rangle G\ \langle tianji\rangle X\ tianjiWon < 2Races$, filtering can really decrease the time needed for the verification. This is expected since the $\langle tianji\rangle X\ tianjiWon < 2Races$ subformula is true in a small subset of the states, reducing the number of strategies to consider.

Finally, the box plots of Figure 5 show that, when checking the formula $\langle player\rangle G\ \langle player\rangle F\ playerWins$ without early termination, caching really helps. This is expected because without early termination, all strategies must be checked;

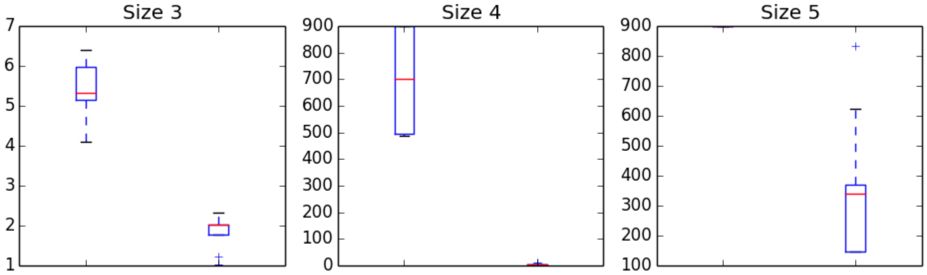

Fig. 4. Box plot of time (in seconds) needed for checking the formula $\langle tianji\rangle G\ \langle tianji\rangle X\ tianjiWon < 2Races$ with 3 to 5 horses. In each plot, boxes represent measures without filtering and with filtering (from left to right). On the third box plot, model checking exceeded the limit of 900 seconds for all checks without filtering.

thus, a lot of redundant work is performed to get the states satisfying the subformula and this redundant work is avoided with caching.

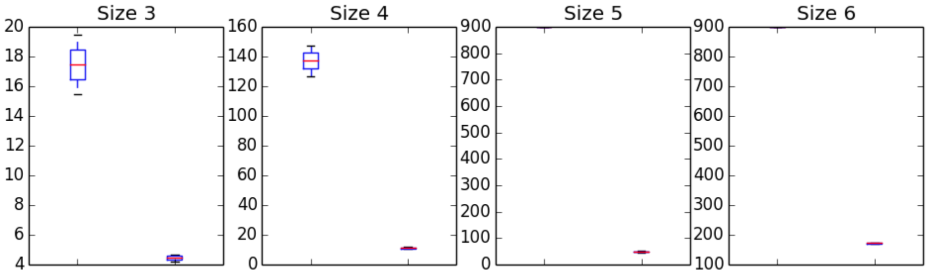

Fig. 5. Box plot of time (in seconds) needed for checking the formula $\langle player\rangle G\ \langle player\rangle F\ playerWins$ with 3 to 6 cards. In each plot, boxes represent measures without caching and with caching (from left to right), when early termination is not activated. On the last two box plots, model checking exceeded the limit of 900 seconds for all checks without caching.

The conclusions we can make based on the test results are the following.

- Activating pre-filtering can greatly improve the process—when there is only a small part of the model that satisfies the formula under full observability—but can also generate unnecessary work when this is not the case, and this can become significant when the number of strategies is small—for example, when early termination is activated. This improvement can thus be helpful in certain cases, but decreases performances for some other cases.
- Activating early termination has, at worst, no impact. It should always be activated since it can greatly improve the verification time when most of the strategies are winning. However, the performed tests did not show a type of early termination better than the others;
- Caching increases performances in some of the tests above; for the others, it has no impact. It should thus be always activated.

7 Conclusion

The model-checking algorithm for $ATLK_{irF}$ presented in [3] is in many cases inefficient because it blindly enumerates all possible strategies to check whether there exists a winning strategy in some states. This paper has presented an approach to generate fewer strategies when we are interested in whether there exists a winning strategy in a small subset of the states of the model. More precisely, we proposed to generate partial strategies reachable from a subset of states of the model that are sufficient to determine the satisfaction of a strategic formula in these states. Based on the generation of these partial strategies, a new algorithm has been designed and the experimental results showed that in a number of cases, the new approach is more efficient than the original one.

While the presented model-checking algorithm clearly improves the efficiency of the verification, it may still be improved along different directions. For example, given the $ATLK_{irF}$ formula $\langle player \rangle F\ playerWins$, it is in theory not necessary to generate and check any strategy if the initial states satisfy $playerWins$. The approach of on-the-fly model checking consists in exploring only the part of the system that is necessary to know whether a particular state satisfies or not a given property. It has been studied by several authors and many techniques have been developed in this direction [10,11,12]. One possible extension of our work involves the possibility of applying such techniques to our setting.

References

1. Alur, R., Henzinger, T.A., Kupferman, O.: Alternating-time temporal logic. J. ACM **49**(5) (September 2002) 672–713
2. Schobbens, P.Y.: Alternating-time logic with imperfect recall. Electronic Notes in Theoretical Computer Science **85**(2) (2004) 82 – 93
3. Busard, S., Pecheur, C., Qu, H., Raimondi, F.: Reasoning about strategies under partial observability and fairness constraints. In: SR. (2013) 71–79
4. Dastani, M., Jamroga, W.: Reasoning about strategies of multi-agent programs. In: Proceedings of AAMAS 10. (2010) 997–1004
5. Jamroga, W., van der Hoek, W.: Agents that know how to play. Fundamenta Informaticae **Volume 63**(2) (2004) 185–219
6. Clarke, E.M., Grumberg, O., Peled, D.: Model Checking. MIT Press (1999)
7. Fagin, R., Halpern, J.Y., Moses, Y., Vardi, M.Y.: Reasoning about Knowledge. MIT Press, Cambridge (1995)
8. Dima, C., Enea, C., Guelev, D.: Model-checking an alternating-time temporal logic with knowledge, imperfect information, perfect recall and communicating coalitions. In: GANDALF 2010. 103–117
9. Busard, S., Pecheur, C.: Pynusmv: Nusmv as a python library. In: NASA Formal Methods. Springer (2013) 453–458
10. Stirling, C., Walker, D.: Local model checking in the modal mu-calculus. In: TAPSOFT'89, Springer (1989) 369–383
11. Bhat, G., Cleaveland, R., Grumberg, O.: Efficient on-the-fly model checking for ctl. In: LICS'95. 388–397
12. Mateescu, Sighireanu: Efficient on-the-fly model-checking for regular alternation-free mu-calculus. Science of Computer Programming **46**(3) (2003) 255–281