

# Model-Checking Iterated Games<sup>\*</sup>

Chung-Hao Huang<sup>1</sup>   Sven Schewe<sup>2</sup>   Farn Wang<sup>1,3</sup>

1: Graduate Institute of Electronic Engineering, National Taiwan University

2: Department of Computer Sciences, University of Liverpool

3: Department of Electrical Engineering, National Taiwan University

**Abstract.** We propose a logic for the definition of the collaborative power of groups of agents to enforce different temporal objectives. The resulting *temporal cooperation logic (TCL)* extends ATL by allowing for successive definition of strategies for agents and agencies. Different to previous logics with similar aims, our extension cuts a fine line between extending the power and maintaining a low complexity: model-checking TCL sentences is EXPTIME complete in the logic, and fixed parameter tractable for specifications of bounded size. This advancement over non-elementary logics is bought by disallowing a too close entanglement between cooperation and competition. We show how allowing such an entanglement immediately leads to a non-elementary complexity. We have implemented a model-checker for the logic and shown the feasibility of model-checking on a few benchmarks.

## 1 Introduction

While the verification of traditional linear and branching time logics like LTL, CTL, and CTL\* [17, 8] has been reduced to (repeated) reachability [11, 13], the satisfiability checking and synthesis problem has been tightly linked with game theory ever since the seminal works of Büchi and Landweber [5, 4]. With the introduction of *alternating time logic (ATL)* by Alur, Henzinger, and Kupferman [1] and in automata based  $\mu$ -calculus model-checking (e.g., [22]), games have entered into the verification of the correctness of reactive systems. With game theoretic challenges moving into the focus of researchers who study the specification and design of reactive systems, traditional problems of multi-player games are replacing the former distinction between an adversarial environment and a supportive system. Instead, we have groups of players that cooperate on some objectives while competing on others.

For particular properties, the intuition that some players represent the system while other players represent the environment is, however, still useful. Following this intuition, the system wins the game in an execution (or a *play* in the jargon of game theory) if the system specification is fulfilled along it, and it wins the game if it can force a winning play. System design as a whole for specifications in game logics can rather be compared to designing a game board and to show that the respective group of players (or: agency) has the coalition power required by the system specification.

---

<sup>\*</sup> The research was supported by the National Science Council grant 97-2221-E-002-129-MY3 and by the Engineering and Physical Sciences Research Council grant EP/H046623/1.

There are various established game-based specification languages, including *ATL*, *ATL\**, the *alternating  $\mu$ -calculus (AMC)*, and *game logic (GL)* [1], *strategy logics* [7, 9, 15, 14], *coordination logic* [10], *stochastic game logic* [3], and *basic strategy interaction logic (BSIL)* [21] for the specification of the interplay in open systems. Each language also comes with a verification algorithm that determines whether a winning strategy for the system exists. However, there is a gap between the available techniques and the scalability required for industrial applications. Frankly speaking, none of the languages above represents, in our view, a proper combination of expressiveness for close interaction among agent strategies and efficiency for the verification or refutation of compliance with a specification. On one hand, logics like *ATL*, *ATL\**, *AMC*, and *GL* [1] allow us to specify the collaborative power of groups of players to enforce a common objective. This falls short from specifying even the simple properties in a typical game. For example, it was shown in [21] that *ATL*, *ATL\**, *AMC*, and *GL* [1] cannot express that the same strategy of a banking system must allow the clients both, to withdraw and to deposit money: a strategy quantifier in these logics always refers to the strategies of all agents, whereas this property requires to bind first the strategy of the bank, and then refer to different strategies of the clients. This is arguably a severe restriction when reasoning about real-world problems.

To solve the expressiveness problem in the above example, *strategy logics (SL)* were proposed in [3, 7, 15, 14]. They allow for the flexible quantification over strategies in logic formulas. However, their verification complexity is prohibitively high and has inhibited practical application.

A previous attempt to tame the complexity of strategy interaction [21], on the other hand, results in a full temporalisation. This leads to severe restrictions in the entanglement between temporal operators and strategy binding and thus prevents, for example, reasoning about Nash equilibria.

We thus propose to adapt the logic introduced in [21] to a new temporal logic called *temporal cooperation logic (TCL)* for this purpose. Let us introduce *TCL* informally on a game among three prisoners.

**Example: Iterated Prisoners' Dilemma.** Inspired by the famous prisoners' dilemma, we consider a model where three suspects, who are initially in custody, are interrogated. In our simplified version, they play in turns (rather than concurrently), and have the choices to either admit or deny the charges made against them. If all deny, they will be released based on lack of evidence.

However, a suspect may decide to collaborate with the police and betray her peers. A sole collaborator will be acquitted as a crown witness, while her peers will be sentenced. But if two or more suspects collaborate with the police, all will be sentenced.

In an iterated prisoners' dilemma, the interplay can continue up to an unbounded number of times. Such a game is very useful in modelling collaboration and competition in networks. For example, a strategy in prisoners' dilemma is *nice* if it does not suggest betrayal initially and only suggests betrayal if, in the previous round, another prisoner betrayed [2]. The following *TCL* sentence states that Prisoner 1 has a nice strategy.

$$\langle 1 \rangle \Box ((\langle + \rangle \circ \neg \text{betray}_1) \vee \bigvee_{a \neq 1} \text{betray}_a) \quad (\text{A})$$

$\langle 1 \rangle$  is a *strategy quantifier* (SQ), which states that there exists a strategy of Prisoner 1 to achieve her temporal goal.  $\langle + \rangle$  is a *strategy interaction quantifier* (SIQ) that inherits the strategy from its parent formula. Proposition  $\text{betray}_i$  is an atomic proposition for the betrayal of prisoner  $i$  at the present state. Similarly, we can reflect more involved strategies, such as ‘Prisoner 2 will always betray when she does not have the power to force Player 1 to always play nice.’

$$\langle 2 \rangle (\langle + \rangle \Box \text{betray}_2) \vee \langle + \rangle \Box (\langle + \rangle \bigcirc \neg \text{betray}_1) \vee \bigvee_{a \neq 1} \text{betray}_a \quad (\text{B})$$

Similar properties can be used to specify *forgiving*<sup>1</sup> or other related strategies [2]. A forgiving strategy of Prisoner 1 is reflected by the following TCL property.

$$\langle 1 \rangle \Diamond (\langle + \rangle \bigcirc \neg \text{betray}_1) \wedge \bigvee_{a \neq 1} \text{betray}_a \quad (\text{C})$$

We can also reason about the existence of Prisoner 2’s strategy that avoid betrayal if Prisoner 1 can be unforgiving under this strategy.

$$\langle 2 \rangle (\langle + \rangle \Box \neg \text{betray}_2) \vee \langle +1 \rangle \Diamond (\langle + \rangle \bigcirc \neg \text{betray}_1) \wedge \bigvee_{a \neq 1} \text{betray}_a \quad (\text{D})$$

As can be seen, properties like (B) and (D) are relevant in network environments where plays can be extended round by round without termination. Every agent may track each others’ records to decide whether or not to cooperate. Such a property cannot be expressed in ATL\*, GL, AMC, or BSIL. While it can be expressed with SL, the verification complexity of SL is prohibitive.

In [21], SIQs can neither override nor revoke strategies assigned by the SQ or SIQs in whose scope they are. Consequently, BSIL cannot express deterministic Nash equilibria. To overcome this restriction, we introduce a strategy reset operator that revokes previous strategy assignments.

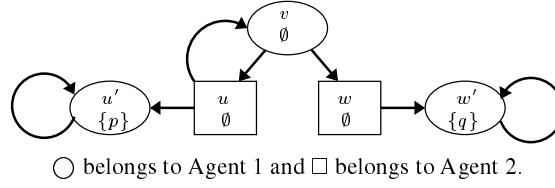
Let  $\text{jail}_a$  be a proposition, which states that ‘‘Prisoner  $a$  is in jail’’. In TCL,

$$\langle 1, 2, 3 \rangle \bigwedge_{a \in [1,3]} (\langle + \rangle \Diamond \neg \text{jail}_a) \vee \langle -a \rangle \Box \text{jail}_a \quad (\text{E})$$

requires that the tree agents can cooperate such that every agent either eventually leaves prism, or stays for ever in prism regardless of her own strategy under the current strategies of the remaining prisoners. The SIQ  $\langle -a \rangle \psi$  revokes the binding of agent  $a$  to her strategy.

In this work, we establish that TCL is incomparable with ATL\*, GL, and AMC in expressiveness. Although the strategy logics proposed in [3, 7, 9, 15] subsume TCL with their flexible quantification of strategies and binding to strategy variables, their model-checking complexities are all doubly exponential time hard. In contrast, TCL enjoys an EXPTIME-complete model-checking complexity and fixed parameter tractability when using the length of the formula as parameter, as well as 2EXPTIME completeness of the TCL satisfiability problem for turn-based game graphs. TCL thus provides a better balance between expressiveness and complexity / efficiency considerations than ATL\*, GL [1], and SL [7, 15, 14]. Given the expressive power as exemplified by the specifications from above, TCL can be viewed as an expressive yet inexpensive subclass of SL [15, 14].

<sup>1</sup> A strategy is forgiving if it does not always punish betrayal in the previous round.



**Fig. 1.** A turn-based game graph

**Organisation of the Paper.** Section 2 explains turn-based game graphs for the description of multi-agent systems and presents the syntax and semantics of TCL. Section 3 discusses the expressiveness of TCL, establishing that CTL, ATL, LTL, and CTL\* can be viewed as syntactic fragments of TCL. We show that TCL is more expressive than any of these logics while incomparable with ATL\*, AMC, and GL [1] in expressiveness, and discuss the effect of a mild extension of TCL. In the following sections, we develop an automata based model-checking algorithm and establish the EXPTIME-completeness and 2EXPTIME-completeness of the TCL model-checking and satisfiability problem, respectively. Finally, we have implemented a model-checker and validated the feasibility of using TCL on a set of benchmarks.

## 2 System Models and TCL

### 2.1 Turn-based game graphs

A *turn-based* game is played by a finite number  $m$  of agents, indexed 1 through  $m$ . A game is a tuple  $\mathcal{G} = \langle m, \mathcal{Q}, r, \omega, P, \lambda, E \rangle$ , where

- Parameter  $m$  is the number of agents in the game.
- $\mathcal{Q}$  is the set of states and  $r \in \mathcal{Q}$  is the *initial state* (or root) of  $\mathcal{G}$ .
- $\omega : \mathcal{Q} \mapsto [1, m]$  is a function that specifies the owner of each state. Only the owner of a state makes choices at the state.
- $P$  is a finite set of atomic propositions.
- $\lambda : \mathcal{Q} \mapsto 2^P$  is a proposition labelling function.
- $E \subseteq \mathcal{Q} \times \mathcal{Q}$  is the set of transitions.

For ease of notation, we denote with  $\mathcal{Q}_a = \{q \in \mathcal{Q} \mid \omega(q) = a\}$  the states owned by an agent  $a$ .

In Figure 1, we have the graphical representation of a turn-based game graph. The ovals and squares represent states while the arcs represent state transitions. We also put down the  $\lambda$  values inside the corresponding states.

For convenience, in the remaining part of the manuscript, we assume that we are always in the context of a given game graph  $\mathcal{G} = \langle m, \mathcal{Q}, r, \omega, P, \lambda, \mathcal{E} \rangle$ . Thus, when we write  $\mathcal{Q}, r, \omega, P, \lambda$ , and  $\mathcal{E}$ , we respectively refer to the components  $\mathcal{Q}, r, \omega, P, \lambda$ , and  $E$  of this  $\mathcal{G}$ .

A *play*  $\rho$  is an infinite path  $q_0 q_1 \dots$  in  $\mathcal{G}$  such that, for every  $k \in \mathbb{N}$ ,  $(q_k, q_{k+1}) \in \mathcal{E}$ .  $\rho$  is *initial* if  $q_0 = r$ . For every  $k \geq 0$ , we let  $\rho(k)$  denote  $q_k$ . Also, given  $h \leq k$ , we let  $\rho[h, k]$  denote  $\rho(h) \dots \rho(k)$  and  $\rho[h, \infty)$  denote the infinite tail of  $\rho$  from  $\rho(h)$ .

A *play prefix* is a finite segment of a play from the beginning of the play. Given a play prefix  $\pi = q_0 q_1 \dots q_n$ ,  $|\pi| = n + 1$  denotes the length of the prefix. Given a  $k \in [0, |\pi| - 1]$ , we let  $\pi(k) = q_k$ . For convenience, we use  $last(\pi)$  to denote the last state in  $\pi$ , i.e.,  $\pi(|\pi| - 1)$ .

For an agent  $a \in [1, m]$ , a *strategy*  $\sigma$  for  $a$  is a function from  $\mathcal{Q}^* \mathcal{Q}_a$  to  $\mathcal{Q}$  such that for every  $\pi \in \mathcal{Q}^* \mathcal{Q}_a$ ,  $\sigma(\pi) \in \mathcal{Q}$  with  $(last(\pi), \sigma(\pi)) \in \mathcal{E}$ .

An *agency*  $A$  of  $[1, m]$  is a subset of  $[1, m]$ . In a short hand notation, we often drop the curly brackets in the set notation, in particular for singleton and empty sets. For example, “1, 3, 4” is a short hand for  $\{1, 3, 4\}$ .

A play  $\rho$  is *compatible* with a strategy  $\sigma_a$  of an agent  $a \in [1, m]$  iff, for every  $k \in \mathbb{N}$ ,  $\omega(\rho(k)) = a$  implies  $\rho(k + 1) = \sigma(\rho[0..k])$ .

## 2.2 TCL Syntax

A TCL formula  $\phi$  is constructed with the following three syntax rules.

$$\begin{aligned} \phi &::= p \mid \neg\phi_1 \mid \phi_1 \vee \phi_2 \mid \langle A \rangle \psi \\ \psi &::= \phi \mid \eta \mid \psi_1 \vee \psi_2 \mid \psi_1 \wedge \psi_2 \mid \langle +A \rangle \psi_1 \mid \langle +A \rangle \bigcirc \psi_1 \mid \langle +A \rangle \eta_1 \mathbf{U} \psi_1 \mid \langle +A \rangle \psi_1 \mathbf{R} \eta_1 \\ &\quad \mid \langle -A \rangle \psi_1 \mid \langle -A \rangle \bigcirc \psi_1 \mid \langle -A \rangle \eta_1 \mathbf{U} \psi_1 \mid \langle -A \rangle \psi_1 \mathbf{R} \eta_1 \\ \eta &::= \phi \mid \eta_1 \vee \eta_2 \mid \eta_1 \wedge \eta_2 \mid \langle + \rangle \bigcirc \eta_1 \mid \langle + \rangle \eta_1 \mathbf{U} \eta_2 \mid \langle + \rangle \eta_1 \mathbf{R} \eta_2 \\ &\quad \mid \langle -A \rangle \bigcirc \eta_1 \mid \langle -A \rangle \eta_1 \mathbf{U} \eta_2 \mid \langle -A \rangle \eta_1 \mathbf{R} \eta_2 \end{aligned}$$

Here,  $p$  is an atomic proposition in  $\mathcal{P}$  and  $A \subseteq \{1, \dots, m\}$  is an agency. Property  $\langle A \rangle \psi_1$  is an (existential) strategy quantification (SQ) specifying that there exist strategies of the agents in  $A$  that make all plays consistent with these strategies satisfy  $\psi_1$ . Property  $\langle +A \rangle \psi_1$  is an (existential) strategy interaction quantification (SIQ) and can only occur bound by an SQ. Intuitively,  $\langle +A \rangle \psi_1$  means that there exist strategies of the agents in  $A$  that work with the strategies introduced by the ancestor formulas. Likewise,  $\langle -A \rangle$  indicates a revocation of the strategy binding for the agents in  $A$ .  $\langle + \rangle$  is an abbreviation for  $\langle +\emptyset \rangle$  or, equivalently  $\langle -\emptyset \rangle$ . Thus, it neither binds nor revokes the binding of the strategy of any agent. Yet, it provides a temporalisation in that it provides a tree formula that can be interpreted at a particular point.

‘U’ is the *until* operator. The property  $\psi_1 \mathbf{U} \psi_2$  specifies a play along which  $\psi_1$  is true until  $\psi_2$  becomes true. Moreover, along the play,  $\psi_2$  must eventually be fulfilled. ‘R’ is the *release* operator. Property  $\psi_1 \mathbf{R} \psi_2$  specifies a play along which either  $\psi_2$  is always true or  $\psi_2 \mathbf{U} (\psi_1 \wedge \psi_2)$  is satisfied. (Release is dual to until:  $\neg(\phi_1 \mathbf{U} \phi_2) \Leftrightarrow \neg\phi_2 \mathbf{R} \neg\phi_1$ .)

In the following we may use  $\langle ?A \rangle \psi$  to conveniently denote an SQ or SIQ formula with ‘?’ is empty, ‘+’, or ‘-’. An SIQ  $\langle \pm A \rangle \psi$  is called non-trivial if  $A$  is not empty, and trivial otherwise.

Formulas  $\phi$  are called *TCL formulas*, *sentences*, or *state formulas*. Formulas  $\psi$  and  $\eta$  are called *tree formulas*. Note that we strictly require that non-trivial strategy interaction cannot cross path modal operators. This restriction is important because it offers a sufficient level of locality to efficiently model-check a system against a TCL property. To illustrate this and to provide a simple extension that offers more expressive power to the cost of a much higher complexity, we informally discuss a small extension, *extended TCL* (ETCL), where the production rule of  $\psi$  also contains  $\neg\psi$  and show that it can be

used to encode ATL\*, and the realisability problem of prenex QPTL can be reduced to ETCL model-checking.

For convenience, we also have the following shorthand notations.

$$\begin{array}{ll}
true \equiv p \vee (\neg p) & false \equiv \neg true \\
\phi_1 \wedge \phi_2 \equiv \neg((\neg\phi_1) \vee (\neg\phi_2)) & \phi_1 \Rightarrow \phi_2 \equiv (\neg\phi_1) \vee \phi_2 \\
\Diamond\phi_1 \equiv true \mathbf{U}\phi_1 & \Box\phi_1 \equiv false \mathbf{R}\phi_1 \\
\neg \bigcirc \phi_1 \equiv \bigcirc \neg\phi_1 & \langle A \rangle \bigcirc \psi_1 \equiv \langle A \rangle \langle + \rangle \bigcirc \psi_1 \\
\langle A \rangle \psi_1 \mathbf{U} \psi_2 \equiv \langle A \rangle \langle + \rangle \psi_1 \mathbf{U} \psi_2 & \langle A \rangle \psi_1 \mathbf{R} \psi_2 \equiv \langle A \rangle \langle + \rangle \psi_1 \mathbf{R} \psi_2
\end{array}$$

In general, it would also be nice to have the universal SQs and SIQs as duals of existential SQs and SIQs, respectively. Couldn't we add, or encode by pushing negations to state formulas, a property of the form  $[+A]\psi_1$ , meaning that, for all strategies of agency  $A$ ,  $\psi_1$  will be fulfilled? In principle, this is indeed no problem, and extending the semantics would be simple. This logic would be equivalent to allowing for negations in the production rule of  $\psi$ . The problem with this logic is that it is too succinct. We will briefly discuss in the following section that model-checking becomes non-elementary if we allow for such negations.

From now on, we assume that we are always in the context of a given TCL sentence.

### 2.3 TCL Semantics

In order to prepare the definition of a semantics for TCL formulas, we start with the definition of a semantics for sentences of the form  $\langle A \rangle \psi$ , where  $\psi$  does not contain any SQs. We call these formulas *primitive TCL* formulas.

Due to the design of TCL, strategy bindings can only effectively happen at non-trivial SQs  $\langle A \rangle$  and when a non-trivial SIQ  $\langle +B \rangle$  is interpreted. To ease referring to these strategies, we first define the *bound agency* of a subformulas  $\phi$  of a TCL sentence  $\chi$ , denoted  $bind(\phi)$ , as follows.

- For state formulas  $\phi$ ,  $bind(\phi) = \emptyset$ .
- For state formulas  $\langle A \rangle \psi$ ,  $bind(\psi) = A$  (unless  $\psi$  is a state formula).
- For tree formulas  $\psi_1 = \langle +A \rangle \psi_2$ ,  $bind(\psi_2) = bind(\psi_1) \cup A$ .
- For tree formulas  $\psi_1 = \langle -A \rangle \psi_2$ ,  $bind(\psi_2) = bind(\psi_1) \setminus A$ .
- For all other tree formulas  $\psi_1$  or  $\psi_2$  with  $\psi = \psi_1 \mathbf{OP} \psi_2$ , with  $\mathbf{OP} \in \{\wedge, \vee, \mathbf{U}, \mathbf{R}\}$ , we have  $bind(\psi_1) = bind(\psi)$  or  $bind(\psi_2) = bind(\psi)$ , respectively.

$bind$  shows, which agents have strategies assigned to them by an SIQ or SQ. Note that this leaves the  $bind$  undefined for all state formulas not in the scope of an SQ formulas. For completeness, we could define  $bind$  as empty in these cases, but a definition will not be required in the definition of the semantics.

As the introduction of additional strategies through non-trivial SIQ  $\langle +B \rangle$  is governed by a *positive* Boolean combination, all strategy selections can be performed concurrently. Such a design leads us to the concept of strategy schemes.

A *strategy scheme*  $\sigma$  is the set of strategies introduced by any non-trivial SQ  $\langle A \rangle$  or SIQ  $\langle +A \rangle$ . By abuse of notation, we use  $\sigma[\phi, a]$  to identify such a strategy. Read in this way,  $\sigma$  can be viewed as a partial function from subformulas and their bound agencies to strategies. Thus,  $\sigma[\phi, a]$  is defined if  $a \in bind(\phi)$  is in the bound agency of  $\phi$ .

For example, given a strategy scheme  $\sigma$  for a TCL sentence  $\langle 1 \rangle \diamond (\langle +2 \rangle \circ p) \wedge \langle 2 \rangle \Box q$ , the strategy used in  $\sigma$  by Agent 1 to enforce the whole formula can be referred to by

$$\sigma[\langle 1 \rangle \diamond (\langle +2 \rangle \circ p) \wedge \langle 2 \rangle \Box q, 1],$$

but also by  $\sigma[\langle +2 \rangle \circ p, 1]$ , while  $\sigma[\langle 2 \rangle \Box q, 1]$  is undefined.

We use a simple tree semantics for TCL formulas. A (computation) tree  $T_r$  is obtained by unravelling  $\mathcal{G}$  from  $r$  and expand the ownership and labelling functions from  $\mathcal{G}$  to  $T_r$  in the natural way. Technically, we have the following definition.

**Definition: Computation Tree.** A *computation tree* for a turn based game  $\mathcal{G}$  from a state  $q$ , denoted  $T_q$ , is the smallest set of play prefixes that contains  $q$  and, for all  $\pi \in T$  and  $(\text{last}(\pi), q') \in \mathcal{E}$ ,  $\pi q' \in T$ . ■

The *strategy-pruned tree* for a tree node  $\pi$ , a strategy scheme  $\sigma$ , and a subformula  $\psi_1$  of  $\chi$  from a state  $q$ , in symbols  $T_q\langle \pi, \sigma, \psi_1 \rangle$ , is the smallest subset of  $T_q$  such that:

- $\pi \in T_q\langle \pi, \sigma, \psi_1 \rangle$ ;
- for all  $\pi' \in T_q\langle \pi, \sigma, \psi_1 \rangle$  with  $\omega((\text{last}(\pi'))) \notin \text{bnd}(\psi_1)$  and  $(\text{last}(\pi'), q') \in \mathcal{E}$ ,  $\pi' q' \in T_q\langle \pi, \sigma, \psi_1 \rangle$ ;
- for all  $\pi' \in T_q\langle \pi, \sigma, \psi_1 \rangle$ ,  $a = \omega((\text{last}(\pi')))$ , and  $q' = \sigma[\psi_1, a](\pi')$  with  $a \in \text{bnd}(\psi_1)$ ,  $\pi' q' \in T_q\langle \pi, \sigma, \psi_1 \rangle$ .

Given a computation tree or a strategy-pruned tree  $T$  and a node  $\pi \in T$ , for every  $\pi q \in T$ , we say that  $\pi q$  is a successor of  $\pi$  in  $T$ . A play  $\rho$  is a *limit of  $T$*  (or an infinite path in  $T$ ), in symbols  $\rho \overset{\infty}{\in} T$ , if there are infinitely many prefixes of  $\rho$  in  $T$ .

We now define the semantics of subformulas of primitive TCL formulas inductively as follows. Given the computation tree  $T_q$  of  $\mathcal{G}$ , a tree node  $\pi \in T_q$ , and a strategy scheme  $\sigma$ , we write  $T_q, \pi, \sigma \models \psi_1$  to denote that  $T_q$  satisfies  $\psi_1$  at node  $\pi$  with strategy scheme  $\sigma$ .

While the notation might seem heavy on first glance, note that the truth for state formulas merely depends on the state  $\text{last}(\pi)$  in which they are interpreted, and the tree formulas are simply interpreted on a strategy pruned tree rooted in  $\pi$  and defined by the strategy scheme.

- For state formulas  $\phi$  other than SQ formulas, we use the state formula semantics:  
 $T_q, \pi, \sigma \models \phi$  iff  $\mathcal{G}, \text{last}(\pi) \models \phi$ , with the usual definition.
  - $\mathcal{G}, q \models p$  if, and only if,  $p \in \lambda(q)$ ,
  - $\mathcal{G}, q \models \neg \phi$  if, and only if,  $\mathcal{G}, q \not\models \phi$ ,
  - $\mathcal{G}, q \models \phi_1 \vee \phi_2$  if, and only if,  $\mathcal{G}, q \models \phi_1$  or  $\mathcal{G}, q \models \phi_2$ , and
  - $\mathcal{G}, q \models \phi_1 \wedge \phi_2$  if, and only if,  $\mathcal{G}, q \models \phi_1$  and  $\mathcal{G}, q \models \phi_2$ .

(Note that this allows for using negation for state formulas.)

- $T_q, \pi, \sigma \models \psi_1 \vee \psi_2$  iff  $T_q, \pi, \sigma \models \psi_1$  or  $T_q, \pi, \sigma \models \psi_2$ . (The  $\psi_i$  are no state formulas.)
- $T_q, \pi, \sigma \models \psi_1 \wedge \psi_2$  iff  $T_q, \pi, \sigma \models \psi_1$  and  $T_q, \pi, \sigma \models \psi_2$  hold.
- $T_q, \pi, \sigma \models \langle \pm A \rangle \circ \psi$  iff, for all successors  $\pi q'$  of  $\pi$  in  $T_q\langle \pi, \sigma, \langle \pm A \rangle \circ \psi_1 \rangle$ ,  $T_q, \pi q', \sigma \models \psi$  holds.
- $T_q, \pi, \sigma \models \langle \pm A \rangle \psi_1 \text{U} \psi_2$  iff, for all limits  $\rho \overset{\infty}{\in} T_q\langle \pi, \sigma, \langle \pm A \rangle \psi_1 \text{U} \psi_2 \rangle$ , there is a  $k \geq |\pi| - 1$  such that  $T_q, \rho[0, k], \sigma \models \psi_2$  and, for all  $h \in [|\pi| - 1, k - 1]$ ,  $T_q, \rho[0, h], \sigma \models \psi_1$  hold.

- $T_q, \pi, \sigma \models \langle \pm A \rangle \psi_1 R \psi_2$  iff, for all limits  $\rho \in T_q \langle \pi, \sigma, \langle \pm A \rangle \psi_1 R \psi_2 \rangle$ , one of the following two restrictions are satisfied.
    - For all  $k \geq |\pi| - 1$ ,  $T_q, \rho[0, k], \sigma \models \psi_2$ .
    - There is a  $k \geq |\pi| - 1$  such that  $T_q, \rho[0, k], \sigma \models \psi_1 \wedge \psi_2$ , and, for all  $h \in [|\pi| - 1, k]$ ,  $T_q, \rho[0, h], \sigma \models \psi_2$ .
  - $T_q, \pi, \sigma \models \langle \pm A \rangle \psi_1$  iff  $T_q, \pi, \sigma \models \psi_1$ .
  - $\mathcal{G}, q \models \langle A \rangle \psi_1$  iff there is a strategy scheme  $\sigma$  such that  $T_q, q, \sigma \models \psi_1$ .
- If  $\phi_1$  is a TCL sentence then we write  $\mathcal{G} \models \phi_1$  for  $\mathcal{G}, r \models \phi_1$ .

Note that, while asking for the existence of a strategy scheme refers to all strategies introduced by some SQ or SIQ in the TCL sentence, only the strategies introduced by the respective SQ and the SIQs in its scope are relevant.

The simplicity of the semantics is owed to the fact that it suffices to introduce new strategies at the points where eventualities become true for the first time. Thus, they do not really depend on the position in which they are invoked and we can guess them up-front. (Or, similarly, together with the points on the unravelling where they are invoked.) This is possible, simply because the validity of state formulas (and hence of TCL sentences) cannot depend on the validity of the left hand side of an until (or the right hand side of a release) *after* the first time it has been satisfied.

### 3 Expressiveness of TCL

Note that TCL is not a superclass of BSIL since BSIL allows for negation in front of SIQs while TCL does not. However, by examining the proofs in [21] for the inexpressibility of BSIL properties by ATL\*, GL, and AMC, we find that the BSIL sentence used in the proofs is also a TCL sentence. This leads to the conclusion that there are properties expressible in TCL but cannot be expressed in ATL\*, GL, and AMC.

**Lemma 1.** *There are TCL sentences that cannot be expressed in any of ATL\*, GL, or AMC.* ■

TCL is, in fact, not only a powerful logic, but also contains important logics either as syntactical fragments or can embed them in a straight forward way. ATL and CTL can be viewed as syntactic fragments of TCL.

But it is also simple to embed LTL and even CTL\*. We start with  $\exists LTL$ , the less used variant where one is content if one path satisfies the formula. We then translate an LTL formula, which we assume w.l.o.g. to be in negative normal form (negations only in front of atomic propositions). Then “there is a path that satisfies  $\phi$ ” is equivalent to  $\langle 1, \dots, m \rangle \hat{\phi}$ , where  $\hat{\phi}$  is derived from  $\phi$  by replacing every occurrence of  $\bigcirc$ ,  $U$ , and  $R$  by  $\langle + \rangle \bigcirc$ ,  $\langle + \rangle U$ , and  $\langle + \rangle R$ , respectively. The simple translation is possible because the formula  $\hat{\psi}$  is de-facto interpreted over a path, the path formed by the joint strategy of the agency  $[1, m]$ . The  $\langle + \rangle$  operators we have added have no effect on the semantics in such a case, just as a CTL formula can be interpreted as the LTL formula obtained by deleting all path quantifiers when interpreted over a word.

Consequently, we have the expected semantics for  $\forall LTL$ : “all paths satisfy  $\phi$ ” is equivalent to  $\neg \langle A \rangle \neg \hat{\phi}$ , where  $\neg \phi$  is assumed to be re-written in negative normal form. The encoding of  $\exists LTL$  and  $\forall LTL$  can easily be extended to the encoding of CTL\*.



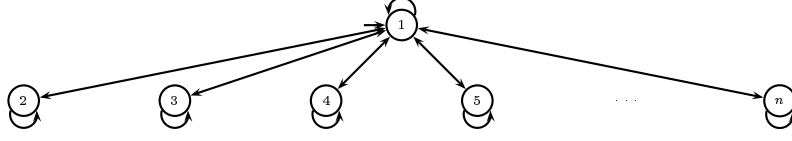


Fig. 2. The turn-based game graph from the non-elementary hardness proof of extended TCL.

**Lemma 2.** *TCL is more expressive than CTL\* and LTL.* ■

This encoding does not extend to ATL\*.  $\langle 1 \rangle (\Box p \vee \Box q)$  is an ATL\* property that cannot be expressed with TCL.

This is different from the ATL property  $\langle 1 \rangle \Box p \vee \langle 1 \rangle \Box q$  or the TCL property  $\langle 1 \rangle (\langle + \rangle \Box p \vee \langle + \rangle \Box q)$ . In fact, the proofs and examples in [21] can also be applied in this work to show that there are properties of ATL\* (or GL, or AMC) that cannot be expressed with TCL. This leads to the following lemma.

**Lemma 3.** *TCL is incomparable in expressiveness with ATL\*, GL, and AMC.* ■

Note, however, that allowing for a negation in the definition of  $\psi$  would change the situation. Then an ATL\* formula  $\langle A \rangle \psi$  (assuming for the sake of simplicity that  $\psi$  is an LTL formula), would become  $\langle A \rangle \neg \langle +[1, m] \setminus A \rangle \widehat{\psi}$  in the extended version of TCL. The translation extends to full ATL\*, but this example also demonstrates why negation is banned: even without nesting, we can, by encoding ATL\*, encode a 2EXPTIME complete model-checking problem, losing the appealing tractability of our logic.

In fact, it is easy to reduce the realisability problem of prenex QPTL, and hence a non-elementary problem, to the model-checking problem of extended TCL. Using the game structure from Figure 2, we can encode the realisability of a prenex QPTL formula with  $n - 1$  variables, for simplicity of the form  $\forall p_2 \exists p_3 \forall p_4 \dots \exists p_n \phi$ , where  $p_2, \dots, p_n$  are all propositions occurring in  $\phi$ . We reduce this to model-checking the formula

$$\phi' = \langle 1 \rangle \neg \langle +2 \rangle \neg \langle +3 \rangle \neg \langle +4 \rangle \neg \dots \neg \langle +n \rangle (\psi_\phi \wedge \langle + \rangle \Box p_1),$$

where  $\psi_\phi$  can be obtained from  $\widehat{\phi}$  by replacing

- every literal  $p_i$  by  $\langle -1 \rangle \langle +1 \rangle \bigcirc (p_i \wedge \langle + \rangle \bigcirc p_i)$ , and
- every literal  $\neg p_i$  by  $\langle -1 \rangle \langle +1 \rangle \bigcirc (p_i \wedge \langle + \rangle \bigcirc \neg p_i)$ .

These formulas are technically not extended TCL formulas as  $\langle +i \rangle \psi_1$  is not part of the production rule of  $\psi$ , but  $\langle +i \rangle \psi_1$  can be used as an abbreviation for  $\langle +i \rangle \text{false} U \psi_1$ .

Checking satisfiability of  $\phi$  is equivalent to model-checking  $\phi'$  on the game shown in Figure 2. The game has  $n + 1$  nodes, agents, and atomic propositions. The nodes in Figure 2 are labeled with the agent that owned the nodes, and the atomic proposition  $p_i$  is true exactly in node  $i$ . From his state, Agent 1 can move to any other state, while all other agents can either stay in their state or return to the state owned by Agent 1.

The game starts in the node owned by Agent 1, and in order to comply with the specification, the outermost strategy profile chosen by Agent 1 must be to stay in the initial state for ever.  $\psi_\phi$  is chosen to align the truth of  $p_i$  at position  $j \in \mathbb{N}$  with the decision that Agent  $i$  makes on the history  $1^j i$ : *true* corresponds to staying in  $i$  and *false* with returning to 1.

It is not hard to establish a matching upper bound for model-checking extended TCL.

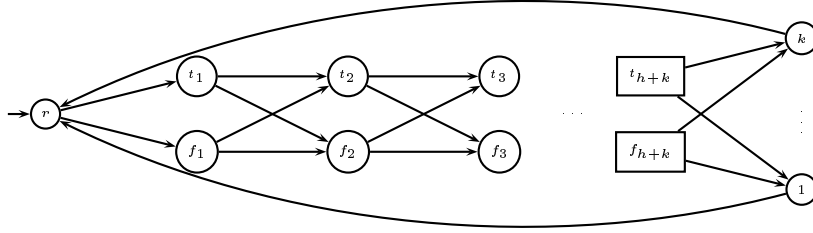


Fig. 3. The turn-based game graph from the EXPTIME hardness proof.

## 4 Complexity of TCL

In this section, we show that model-checking TCL formulas is EXPTIME-complete in the formula and P-complete in the model (and for fixed formulas), while the satisfiability problem is 2EXPTIME-complete. As the proof of inclusion of the satisfiability problem in 2EXPTIME builds on the proof of the inclusion of model-checking in EXPTIME, we start with an outline of the EXPTIME hardness argument for the TCL model-checking problem and then continue with describing EXPTIME and 2EXPTIME decision procedures for the TCL model and satisfiability checking problem, respectively. 2EXPTIME hardness for TCL satisfiability is implied by the inclusion of CTL\* as a de-facto sub-language [20].

We show EXPTIME hardness by a reduction from the PEEK- $G_6$  [19] game. An instance of PEEK- $G_6$  consists of two disjoint sets of boolean variables,  $P_1 = \{p_1, \dots, p_h\}$  (owned by a safety agent) and  $P_2 = \{p_{h+1}, \dots, p_{h+k}\}$  (owned by a reachability agent), a subset  $I \subseteq P_1 \cup P_2$  of them that are initially *true*, and a boolean formula  $\gamma$  in CNF over  $P_1 \cup P_2$  that the reachability agent wants to become *true* eventually. The game is played in turns between the safety and the reachability agent (say, with the safety agent moving first), and each player can change the truth value of one of his or her variables in his/her turn.

**Lemma 4.** *TCL model-checking is EXPTIME hard for primitive TCL formulas.*

*Proof.* To reduce determining the winner of an instance of a PEEK- $G_6$  game to TCL model-checking, we introduce a 2-agent game  $\mathcal{G} = \langle 2, \mathcal{Q}, r, \omega, \mathcal{P}, \lambda, \mathcal{E} \rangle$  as shown in Figure 3, where Agent 1 (he, for convenience) represents the safety agent while Agent 2 (she, for convenience) represents the reachability agent.  $t_{h+k}$  and  $f_{h+k}$  are the only states owned by Agent 2.

The game is played in rounds, and a round starts each time the game is at state  $r$ . If the game goes through  $t_i$  this is identified with the variable  $p_i$  to be true. Likewise, going through  $f_i$  is identified with the variable being false.

It is simple to write a TCL specification that forces the safety player to toggle the value of exactly one of his variables in each round, and to toggle the value of the variable  $p_{h+i}$  of the reachability player defined by the state  $i$  she has previously moved to, while maintaining all other variable values. Requiring additionally that the safety agent can guarantee that the boolean formula is never satisfied provides the reduction. ■

The details of the construction are moved to Appendix ???. It is interesting that a game with only two agents suffices for the proof. Two agents are also sufficient to show P hardness for fixed formulas, as solving a reachability problem for AND-OR graphs [12] naturally reduces to showing  $\langle 1 \rangle \diamond p$ .

**Lemma 5.** *TCL model-checking for fixed formulas is P hard for primitive TCL formulas.* ■

In order to establish inclusion in EXPTIME and P, respectively, we use an automata based argument.

**Theorem 1.** *The model-checking problem of TCL formulas against turn-based game graphs is EXPTIME-complete, and P-complete for fixed formulas.*

*Proof.* We first show the claim for primitive TCL formulas  $\phi = \langle A \rangle \psi$ .

To keep the proof simple, we first consider a tree automaton  $\mathcal{U}$  that checks the acceptance of  $\psi$  for a given strategy scheme  $\sigma$ . That is,  $\mathcal{U}$  checks if  $T_q^+, q, \sigma \models \psi$  under the assumption that both  $\sigma$  and the truth values for the subformulas starting with a  $\langle \pm B \rangle$  are encoded in the nodes of  $T_q^+$ .

Such an automaton would merely have to run simple consistency checks, and it is simple to construct a suitable universal weak tree automaton  $\mathcal{U}$ , which is polynomial in the size of  $\phi$ . From there it is simple to infer a deterministic Büchi tree automaton  $\mathcal{D}$ , which is exponential in the weak universal tree automaton [16].

It is then a trivial step (projection) to *guess*  $\sigma$  and the truth annotation of the subformulas on the fly, turning the deterministic Büchi tree automaton  $\mathcal{D}$  that requires a correct annotation into a nondeterministic Büchi automaton  $\mathcal{N}$  of the same size that checks  $\mathcal{G}, q \models \phi$ . Acceptance can be checked in time quadratic in the size of the product of  $\mathcal{N}$  and  $\mathcal{G}$  [6].

To take the step to full TCL, we can model-check the truth of primitive TCL formulas and then use the result of this model-checking instead of the respective subformula.

Hardness is inherited from Lemmata 4 and 5. ■

This argument shows more: the complexity of TCL model-checking for fixed formulas does not depend on the formula. It suffices to solve a number of Büchi games, where both the size of the game and the number of games to be played is linear in  $\mathcal{G}$ .

**Corollary 1.** *Viewing the size of a TCL sentence as a parameter, TCL model-checking is fixed parameter tractable.*

The automata construction from the proof of Theorem 1 extends to a construction for satisfiability checking.

**Theorem 2.** *The TCL satisfiability problem is 2EXPTIME-complete.*

*Proof.* As usual, it is convenient to construct an enriched model that contains the truth of all subformulas for a TCL sentence  $\phi$  that start with an SQ.

In a first step, we construct an alternating tree automaton  $\mathcal{A}$  that recognises the enriched models of a specification. This is quite simple:  $\mathcal{A}$  merely has to check that the

boolean combination of SQ formulas that forms the TCL sentence  $\phi$  is satisfied and that the truth assignment of each SQ is consistent. But this is simple, as we can use the tree automaton  $\mathcal{N}_{\phi'}$  from the proof for Theorem 1 to validate the claim that a subformula  $\phi'$  of  $\phi$  that starts with an SQ is true, and its dual to validate that it is false. Hence, such an automaton has only two states more than the sum of the states of the individual  $\mathcal{N}_{\phi'}$ . In particular, it is exponential in  $\phi$ .

For the resulting alternating automaton, we can again invoke the simulation theorem [16] to construct an equivalent nondeterministic parity automaton, which has doubly exponentially many states in  $\phi$  (and whose transition table is doubly exponential in  $\phi$ ) and whose colours are exponential in  $\psi$ . Solving the emptiness game of this automaton reduces to solving a parity game, which can be done in time doubly exponential in  $\psi$ , e.g., using [18].

Hardness is inherited from CTL\* satisfiability checking [20]. ■

## 5 Implementation and Experiment

As a proof of concept, we have implemented a model-checker, `tcl`, in C++. `tcl` accepts models composed of extended automata that communicate with synchronisers and shared variables, with an explicit shared variable `turn` that specifies the turn of agents at a state. A turn-based game graph is then constructed as the product of the extended automata. Such an input format facilitates modular description of the interaction among the agents.

The implementation builds on a prototype for a PSPACE logic [21]. The extension is possible because we can reduce the complexity of TCL to PSPACE by simply restricting the number of operators in the  $\eta$  production rules in the scope of any SQ to be logarithmic in the size of the TCL sentence. We show this for primitive TCL sentences.

**Lemma 6.** *Model-checking can be done in space bilinear in the size of the turn based game structure and the state and tree formulas that are produced using the  $\psi$  production rules and exponentially only in the number of  $\eta$  produced tree formulas.*

*Proof.* We have seen that, for a primitive TCL sentence  $\phi$ , we can use a single strategy scheme and only have to refer to the *first* position that the right hand side of an until or the left hand side of a release operator is true. Moreover, it suffices to guess just a minimal set of positions where tree formulas are true. In particular, the left hand side of a release, the right hand side of an until, and a next formula are then marked true exactly once, and the respective release and until formulas never need to be marked as true after such an event.

We can therefore use an alternating algorithm that guesses such minimal truth claims. The algorithm alternates between a verifier who guesses a truth assignment and the current decisions of the strategy scheme, and a falsifier, who guesses the direction into which to expand the path.

It is now easy to see that they will produce an infinite path in this way, and on this path each obligation that refers to a tree subformula from a  $\psi$  production rule can appear only on a continuous interval. The points where these obligations change is therefore linear in the size of  $\phi$ . However, it also needs to track the truth value of tree

formulas produced by the  $\eta$  production rule. (If there are multiple utilities introduced by  $\eta$  production rules, this also includes a marker that distinguishes a leading until, which is changed in a round robin fashion when the leading utility is fulfilled.)

The number of possible assignments is then exponential in the number of tree sub-formulas from  $\eta$  production rules. Note that  $\Box$  formulas can be exempt from this rule: they are monotonous and hence incur a small impact similar to the formulas introduced using the  $\psi$  production rule.

Hence, if  $|\mathcal{G}|$  denotes the size of the turn based game and  $k$  the number of temporal operators (different to  $\Box$ ) introduced by  $\eta$  production rules, we end up in a cycle if there is no change in the truth assignment temporal operators that are introduced by  $\psi$  production rules or  $\Box$  operators we reach a cycle within  $|\mathcal{G}| \cdot k \cdot 2^k$  steps. Hence, we reach a cycle in a number of steps that is linear in  $|\mathcal{G}|$  and the size of  $\phi$ , and exponential only in the size of  $\eta$ -produced temporal operators (different to  $\Box$ ).

Upon reaching a cycle, it suffices to check if the cycle is accepting. (No standing obligation by an until.) ■

The model-checker uses a stack to explicitly enumerate all paths of all tree tops with depth prescribed by Lemma 6. The tool can be downloaded from Sourceforge at project REDLIB at: <http://sourceforge.net/projects/redlib/>.

We use the parametrised models of the iterated prisoners' dilemma as our benchmarks to check the performance of our implementation. A brief explanation of the models can be found in the introduction. The unique parameter to the models are the number of prisoners  $m$ . There is also a policeman in the models. We build a turn-based game graph for each value of  $m$  in the experiments. The parametrisation helps us to observe how our algorithm and implementation scale to model and formula sizes. To simplify the construction of the state-space representation, we assume that, in each iteration, the prisoners make their decisions in a fixed order. After all prisoners have made their decisions, the policeman makes his decision. Subsequently, the whole game moves to the next iteration. We use seven benchmark formulas on these models in our experiments. The first five benchmarks are taken from the examples (A) through (E) from the introduction. Benchmarks (F) and (G) are the following two properties, taken from [21].

- Property (F) specifies that all prisoners except Prisoner 1 can collaborate to release Prisoner 1 and let Prisoner 1 decide their fate.  
 $\langle 2, \dots, m \rangle (\langle + \rangle \Diamond \neg \text{jail}_1) \wedge \bigwedge_{i \in \{2, \dots, m\}} (\langle +1 \rangle \Diamond \neg \text{jail}_i) \wedge (\langle +1 \rangle \Box \text{jail}_i)$  (F)
- Property (G) specifies that Prisoner 1 has a strategy to put all other prisoners in jail while leaving her fate to them.  
 $\langle 1 \rangle (\bigwedge_{i \in \{2, \dots, m\}} \langle + \rangle \Box \text{jail}_i) \wedge (\langle 2, \dots, m \rangle \Diamond \neg \text{jail}_1) \wedge \langle 2, \dots, m \rangle \Box \text{jail}_1$  (G)

For these benchmarks, we have collected the performance data for various parameter values in Table 1. For small models, the memory usage is dominated by the normal overhead, such as the representation of variable tables, state-transition tables, formula structures, etc. The data shows that our prototype can handle the various benchmarks, and scales well on five of the seven benchmarks. Ignoring the overhead, it also shows the exponential growth. The models, however, are growing exponentially, too. We assume that this growth is the main cause of the exponential growth of the response time.

**Table 1.** Performance data of model-checking the TCL fragment

properties \ $m$	2	3	4	5	6	7	8	9	10
(A)	0.71s	0.94s	5.41s	66.3s	945s	>1000s			
	163M	165M	185M	350M	1307M				
(B)	0.50s	0.52s	0.61s	0.71s	1.11s	1.62s	5.77s	20.9s	68.1s
	163M	163M	164M	165M	168M	176M	214M	270M	376M
(C)	0.51s	0.51s	0.6s	0.82s	1.01s	1.81s	5.54s	18.2s	48.3s
	163M	163M	164M	165M	168M	176M	200M	241M	318M
(D)	0.5s	0.51s	0.57s	0.74s	1.01s	1.79s	7.41s	33.8s	141s
	163M	163M	164M	165M	168M	175M	232M	312M	430M
(E)	0.51s	0.66s	19.1s	>1000s					
	163M	164M	194M						
(F)	0.51s	0.53s	0.61s	0.71s	1.01s	1.70s	5.38s	15.2s	53.7s
	163M	163M	163M	165M	168M	175M	202M	243M	295M
(G)	0.52s	0.52s	0.65s	0.72s	1.03s	1.85s	4.86s	16.1s	93.5s
	163M	163M	164M	165M	169M	177M	189M	208M	235M

s: seconds; M: megabytes.

The models are with 1 policeman and  $m$  prisoners. The experiment was carried out on an Intel i5 2.4G notebook with 2 cores and 4G memory, running ubuntu Linux version 11.10.

## 6 Conclusion

TCL is a promising logic for the specification of groups of agents who balance their strategies in order to cooperate with different partners to achieve different objectives. It is an inexpensive logic in many ways. First and foremost, it is fixed parameter tractable. Following folklore, specifications are tiny while models are huge. In this situation, fixed parameter tractability is a very important property, in particular as it is achieved by a natural and simple decision procedure, which is merely exponential in the formula.

This appealing property is not bought with inexpressiveness. In particular, the popular temporal logics LTL, CTL, ATL, and CTL\* are contained as de-facto sublogics. Consequently, it can be excellently used to extend existing specifications in these languages, without the need to develop competitive models.

The applicability is underlined by compelling data from our benchmarks. This is in spite of the fact that our implementation is rather based on an ad hoc extension of an existing algorithm for a different logic, and neither fully exploit the low complexity, nor is a fully symbolic implementation. It will be interesting to see by which extent symbolic representation like BDDs will enhance the performance and how an automata based tool would fare.

## References

1. R. Alur, T. A. Henzinger, and O. Kupferman. Alternating-time temporal logic. *Journal of the ACM (JACM)*, 49(5):672–713, September 2002.

2. R. Axelrod. Effective choice in the prisoner's dilemma. *Journal of conflict resolution*, 24(1):3–25, 1980.
3. C. Baier, T. Brázdil, M. Gröser, and A. Kucera. Stochastic game logic. In *QEST*, pages 227–236. IEEE Computer Society, 2007.
4. J. Büchi and L. Landweber. Definability in the monadic second-order theory of successor. *Journal of Symbolic Logic*, 34(2):166–170, 1969.
5. J. Büchi and L. Landweber. Solving sequential conditions by finite-state strategies. *Trans. AMS*, 138(4):295–311, 1969.
6. K. Chatterjee and M. Henzinger. An  $O(n^2)$  time algorithm for alternating Büchi games. In *Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2012), Kyoto, Japan, January 17-19, 2012*, pages 1386–1399. SIAM, 2012.
7. K. Chatterjee, T. A. Henzinger, and N. Piterman. Strategy logic. *Information and Computation*, 208:677–693, 2010.
8. E. M. Clarke and E. A. Emerson. Design and synthesis of synchronization skeletons using branching-time temporal logic. In *Workshop on Logic of Programs*, volume LNCS 131. Springer-Verlag, 1981.
9. A. D. Costa, F. Laroussinie, and N. Markey. Atl with strategy contexts: Expressiveness and model checking. In *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2010)*, volume 8 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 120–132. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2010.
10. B. Finkbeiner and S. Schewe. Coordination logic. In *CSL*, pages 305–319, 2010.
11. G. J. Holzmann. The model checker spin. *IEEE Trans. Software Eng.*, 23(5), 1997.
12. N. Immerman. Number of quantifiers is better than number of tape cells. *Journal of Computer and System Sciences*, 22(3):65–72, 1981.
13. O. Kupferman, M. Y. Vardi, and P. Wolper. An automata-theoretic approach to branching-time model checking. *Journal of ACM*, 47(2):312–360, 2000.
14. F. Mogavero, A. Murano, G. Perelli, and M. Y. Vardi. What makes  $\text{atl}^*$  decidable? a decidable fragment of strategy logic. In *Concurrency theory (CONCUR 2012)*, volume LNCS 7454, pages 193–208. Springer-Verlag, 2012.
15. F. Mogavero, A. Murano, and M. Y. Vardi. Reasoning about strategies. In *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2010)*, LIPIcs 8, pages 133–144, 2010.
16. D. E. Muller and P. E. Schupp. Simulating alternating tree automata by nondeterministic automata: new results and new proofs of the theorems of Rabin, McNaughton and Safra. *Theoretical Computer Science*, 141(1-2):69–107, 1995.
17. A. Pnueli. The temporal logic of programs. In *18th annual IEEE-CS Symposium on Foundations of Computer Science*, pages 45–57, 1977.
18. S. Schewe. Solving parity games in big steps. In *Proceedings of the 27th Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2007), 12–14 December, New Delhi, India*, volume 4805 of *Lecture Notes in Computer Science*, pages 449–460. Springer-Verlag, 2007.
19. L. J. Stockmeyer and A. K. Chandra. Provably difficult combinatorial games. *SIAM Journal on Computing (SICOMP)*, 8(2):151–174, 1979.
20. M. Vardi and L. Stockmeyer. Improved upper and lower bounds for modal logics of programs: Preliminary report. In *Proceedings of the 17th Annual ACM Symposium on Theory of Computing (STOC '85), May 6-8, Providence, Rhode Island, USA*, pages 240–251, 1985.
21. F. Wang, C.-H. Huang, and F. Yu. A temporal logic for the interaction of strategies. In *22nd Concurrency Theory (CONCUR)*, volume LNCS 6901. Springer-Verlag, Sept. 2011.
22. T. Wilke. Alternating tree automata, parity games, and modal  $\mu$ -calculus. *Bulletin of the Belgian Mathematical Society*, 8(2), May 2001.