Model-Checking Iterated Games

Chung-Hao Huang[1] Sven Schewe[2] Farn Wang[1,3]

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 nonelementary logics is bought by disallowing a too close entanglement between cooperation and competition. We show how allowing such an entanglement immediately leads to a nonelementary 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 μ-calculus model-checking (e.g., [22]), games have entered into the verification the correctness of reactive systems. With game theoretic challenges moving into the focus of researchers studying 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.

There are various established game-based specification languages, including *ATL*, *ATL**, the *alternating μ-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. 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 temporalization. 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 in [21] to a new temporal logic called *temporal cooperation logic* (*TCL*) for this purpose. Let us introduce TCL informaly on a game among three prisoners.

Example 1. **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 acquited 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 modeling 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 refers to nice strategies of Prisoner 1.

$$\langle 1\rangle\Box((\langle +\rangle \bigcirc \neg\mathtt{betray}_1) \vee \bigvee_{a\neq 1} \mathtt{betray}_a) \tag{A}$$

$\langle a\rangle$ 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 $\mathtt{betray}_i$ 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.'

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

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

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

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

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

As can be seen, properties like (B) and (D) are relevant in network environment 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 or 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 $\mathtt{jail}_a$ be a proposition, which says that for prisoner a in jail. In TCL,

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

requires that no agent stays in jail indefinitely, if she can avoid it under the current strategies of the remaining prisoners. The SIQ $\langle -a\rangle\psi$ revokes the binding of the (singleton agency that contains only) agent a to her strategy.

In this work, we establish that TCL is incomparable with ATL*, GL, and AMC in expressiveness. Although strategy logics proposed in [3, 7, 9, 15] subsume TCL with their flexible quantification of strategies and binding to strategy variables, their model-checking complexity 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].

Organization or 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 and show that TCL is more expressive than any of these logics while in comparable 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 implement a model checker and validated the feasibility of using TCL on a set of benchmarks.

[1] A strategy is forgiving if it does not always punish betrayal in the previous round.

2 System Models and TCL

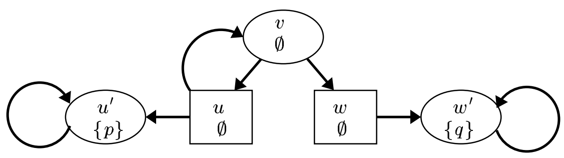
2.1 Turn-based game graphs

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 labeling 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



○ belongs to Agent 1 and □ belongs to Agent 2.

Fig. 1. A turn-based game graph

ovals and squares represent states while the arcs represent state transitions. We also put down the λ 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 $\mathcal{G} = \langle m, \mathcal{Q}, r, \omega, \mathcal{P}, \lambda, \mathcal{E} \rangle$. Thus, when we write $\mathcal{Q}, r, \omega, \mathcal{P}, \lambda$, and $\mathcal{E}$, we respectively refer to the components $\mathcal{Q}$, r, ω, P, λ, and E of this $\mathcal{G}$.

A *play* ρ is an infinite path $q_0 q_1 \ldots$ in $\mathcal{G}$ such that, for every $k \in \mathbb{N}$, $(q_k, q_{k+1}) \in \mathcal{E}$. ρ 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 $\rho(h) \ldots \rho(k)$ and $\rho[h, \infty)$ denote the infinite tail of ρ from $\rho(h)$.

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

For an agent $a \in [1, m]$, a *strategy* σ 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 $\big(\mathit{last}(\pi), \sigma(\pi)\big) \in \mathcal{E}$.

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 ρ is *compatible* with a strategy σ_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 ϕ is constructed with the following three syntax rules.

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

Here p is an atomic proposition in $\mathcal{P}$ and $A \subseteq \{1, \ldots, m\}$ 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 satisfy ψ_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 declared by the ancestor formulas.

'U' is the *until* operator. The property $\psi_1 \mathrm{U}\psi_2$ specifies a play along which ψ_1 is true until ψ_2 becomes true. Moreover, along the play, ψ_2 must eventually be fulfilled. 'R' is the *release* operator. Property $\psi_1 \mathrm{R}\psi_2$ specifies a play along which either ψ_2 is always true or $\psi_2 \mathrm{U}(\psi_1 \wedge \psi_2)$ is satisfied. (Release is dual to until: $\neg(\phi_1 \mathrm{U}\phi_2) \Leftrightarrow \neg\phi_2 \mathrm{R}\neg\phi_1$.)

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 ϕ are called *TCL formulas*, *sentences*, or *state formulas*. Formulas ψ and η 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 ψ also contains $\neg\psi$ and show that it can be used to encode ATL*, and the realizability problem of prenex QPTL can be reduced to ETCL model-checking.

For convenience, we also have the following shorthands.

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

In general, it would also be nice to have the universal SQs and SIQs respectively as duals of existential SQs and SIQs. 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, ψ_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 ψ. The problem with this logic is that it is too succinct. We will briefly discuss in the following section that model checking becomes nonelementary if we allow for such negagtions.

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 ψ does not contain an 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 ϕ of a TCL sentence χ, denoted $\mathit{bnd}(\phi)$, as follows.

- For state formulas ϕ, $\mathit{bnd}(\phi) = \emptyset$.
- For state formulas $\langle A\rangle\psi$, $\mathit{bnd}(\psi) = A$.
- For tree formulas $\psi_1 = \langle +A\rangle\psi_2$, $\mathit{bnd}(\psi_2) = \mathit{bnd}(\psi_1) \cup A$.
- For tree formulas $\psi_1 = \langle -A\rangle\psi_2$, $\mathit{bnd}(\psi_2) = \mathit{bnd}(\psi_1) \smallsetminus A$.
- For all other tree formulas ψ_1 or ψ_2 with $\psi = \psi_1 \mathsf{OP} \psi_2$, with $\mathsf{OP} \in \{\wedge, \vee, \mathcal{U}, \mathcal{R}\}$, we have $\mathit{bnd}(\psi_1) = \mathit{bnd}(\psi)$ or $\mathit{bnd}(\psi_2) = \mathit{bnd}(\psi)$, respectively.

bnd shows, which agents have strategies assigned to them by an SIQ or SQ. Note that this leaves the *bnd* undefined for all state formulas not in the scope of an SQ formulas. For completeness, we could define *bnd* 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* σ 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, σ can be viewed as a partial function from subformulas and their bound agencies to strategies. Thus, $\sigma[\phi, a]$ is defined if $a \in \mathit{bnd}(\phi)$ is in the bound agency of ϕ.

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

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

but also by $\sigma[\langle +2\rangle \bigcirc p, 1]$, 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 1. **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 $(\mathit{last}(\pi), q') \in \mathcal{E}$, $\pi q' \in T$. ∎

The *strategy-pruned* tree for a tree node π, a strategy scheme σ, and a subformula ψ_1 of χ 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 $\omega\big((last(\pi')\big) \notin bnd(\psi_1)$ and $(last(\pi'), q') \in \mathcal{E}$, $\pi' q' \in T_q\langle \pi, \sigma, \psi_1 \rangle$;
- for all $\pi' \in T_q\langle \pi, \sigma, \psi_1 \rangle$, $a = \omega\big((last(\pi')\big)$, and $q' = \sigma[\psi_1, a](\pi')$ with $a \in bnd(\psi_1)$, $\pi' q' \in T_q\langle \pi, \sigma, \psi_1 \rangle$.

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 πq is a successor of π in T. A play ρ is a limit of 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 σ, we write $T_q, \pi, \sigma \models \psi_1$ to denote that T_q satisfies ψ_1 at node π with strategy scheme σ.

- For state formulas ϕ other than SQ formulas, we use the state formula semantics: $T_q, \pi, \sigma \models \phi$ iff $\mathcal{G}, last(\pi) \models \phi$, 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 $T_q, \pi, \sigma \models \psi_2$. (These ψ_i are no state formulas.)
- $T_q, \pi, \sigma \models \psi_1 \wedge \psi_2$ iff $T_q, \pi, \sigma \models \psi_1$ and $T_q, \pi, \sigma \models \psi_2$.
- $T_q, \pi, \sigma \models \langle \pm A \rangle \bigcirc \psi$ iff, for all successors $\pi q'$ of π in $T_q\langle \pi, \sigma, \langle \pm A \rangle \bigcirc \psi_1 \rangle$, $T_q, \pi q', \sigma \models \psi$.
- $T_q, \pi, \sigma \models \langle \pm A \rangle \psi_1 \mathrm{U} \psi_2$ iff, for all limits $\rho \overset{\infty}{\in} T_q\langle \pi, \sigma, \langle \pm A \rangle \psi_1 \mathrm{U} \psi_2 \rangle$, 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.
- $T_q, \pi, \sigma \models \langle \pm A \rangle \psi_1 \mathrm{R} \psi_2$ iff, for all limits $\rho \overset{\infty}{\in} T_q\langle \pi, \sigma, \langle \pm A \rangle \psi_1 \mathrm{R} \psi_2 \rangle$, 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 σ such that $T_q, q, \sigma \models \psi_1$.

If ϕ_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 properties used in the proof is in fact also a property of TCL. This observation leads us to the conclusion that there are properties expressible in TCL but cannot be expressed in ATL*, GL, and AMC.

Lemma 1. *There are TCL properties that cannot be expressed in any of ATL*, GL, and 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 ∃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 ϕ" is equivalent to $\langle 1,\ldots,m\rangle\widehat{\phi}$, where $\widehat{\phi}$ is derived from ϕ 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 $\widehat{\psi}$ 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 ϕ" is equivalent to $\neg\langle A\rangle\widehat{\neg\phi}$, where $\neg\phi$ is assumed to be re-written in negative normal form. The encoding of ∃LTL and ∀LTL can easily be extended to the encoding of CTL*.

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

Note that this encoding does not extend to ATL*. The following example shows an ATL* property that cannot be expressed with TCL.

$$\langle 1\rangle((\Box p) \vee \Box q)$$

Note that this is different from ATL property $(\langle 1\rangle\Box p)\vee\langle 1\rangle\Box q$ or 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 any of ATL*, GL, and AMC.* ■

Note, however, that allowing for a negation in the definition of ψ would change the situation. Then an ATL* formula $\langle A\rangle\psi$ (assuming for the sake of simplicity that ψ is an LTL formula), would become $\langle A\rangle\neg\langle +[1,m]\smallsetminus A\rangle\widehat{\neg\psi}$ 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 this extended version of TCL: using the following game structure, we can encode the realisability of a prenex QPTL formula with $n-1$ variables, for the sake of simplicity of the form $\forall p_2 \exists p_3 \forall p_4 \ldots \exists p_n \phi$, where $p_2, \ldots, p_n$ are all propositions occurring in ϕ. We reduce this to the model-checking of the formula

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

where ψ_ϕ 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 $\langle -1\rangle\langle +1\rangle \bigcirc (p_i \wedge \langle +\rangle \bigcirc \neg p_i)$.

(While these formulas are technically not extended TCL formulas because $\langle +i\rangle\psi_1$ is not part of the production rule of ψ, $\langle +i\rangle\psi_1$ can be used as an abbreviation for $\langle +i\rangle \mathit{false}\mathrm{U}\psi_1$.)

Checking satisfiability of ϕ is is equivalent to model-checking ϕ' 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. ψ_ϕ 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.

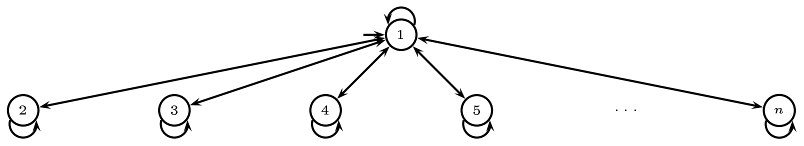


Fig. 2. The turn-based game graph from the non-elementary hardness proof of extended TCL.

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

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 satifiability 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, $P_1 = \{p_1, \ldots, p_h\}$ (owned by a safety agent) and $P_2 = \{p_{h+1}, \ldots, p_{h+k}\}$ (owned by a reachability agent), a subset $I \subseteq P_1 \cup P_2$ of them that are initially *true*, and a boolean formula γ 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.

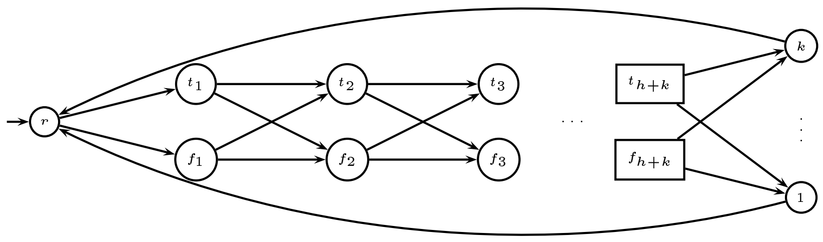


Fig. 3. The turn-based game graph from the EXPTIME hardness proof.

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 variable 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 A. 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 ψ for a given strategy scheme σ. That is, $\mathcal{U}$ checks if ${T_q}^{+}, q, \sigma \models \psi$ under the assumption that both σ 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 ϕ. 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* σ 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. ■

Note that 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 ϕ 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 ϕ 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 ϕ' of ϕ 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 ϕ.

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 ϕ (and whose transition table is doubly exponential in ϕ) and whose colours are exponential in ψ. Solving the emptiness game of this automaton reduces to solving a parity game, which can be done in time doubly exponential in ψ, 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 synchronizers 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 η 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 ψ production rules and exponentially only in the number of η produced tree formulas.*

Proof. We have seen that, for a primitive TCL sentence ϕ, 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 therefor 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 referring to a tree subformula from a ψ production rule can appear only on a continuous interval. In particular, the points where these obligations change, is linear in the size of ϕ. However, it also needs to track the truth value of tree formulas produced by the η production rule. (If there are multiple untilities introduced by η production rules, this also includes a marker that distinguishes a leading until, which is changed in a round robin fashion when the leading untility is fulfilled.)

The number of possible assignments is then exponential in the number of tree subformulas from η 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 ψ 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 η production rules, we end up in a cycle if there is no change in the truth assignment temporal operators that are introduced by ψ 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 $|G|$ and the size of ϕ, and exponential only in the size of η-produced temporal operators (different to $\Box$).

Upon reaching a cycle, is 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 parameterized 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 built a turn-based game graph for each values of m in the experiment. The parameterization helps us in observing 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 in an iteration, the policeman make his decision and then the whole game moves to the next iteration.

We have used seven benchmarks in our experiments. The first five benchmarks are taken from the examples (A) through (E) used in the introduction to introduce TCL. 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.
$$\langle 2,\ldots,m\rangle\big((\langle +\rangle\Diamond\neg\mathtt{jail}_1) \wedge \bigwedge_{i\in\{2,\ldots m\}}(\langle +1\rangle\Diamond\neg\mathtt{jail}_i) \wedge (\langle +1\rangle\Box\mathtt{jail}_i\big) \tag{F}$$
- Property (G) specifies that Prisoner 1 has a strategy to put all other prisoners in jail while leaving her fate to them.
$$\langle 1\rangle\big((\bigwedge_{i\in\{2,\ldots m\}}\langle +\rangle\Box\mathtt{jail}_i) \wedge (\langle 2,\ldots,m\rangle\Diamond\neg\mathtt{jail}_1) \wedge \langle 2,\ldots,m\rangle\Box\mathtt{jail}_1\big) \tag{G}$$

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. Note, however, that the models are growing exponentially, too, and we assume to be the main cause of the exponential growth of the response time.

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

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.

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 inexpressivity. 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 th emonadic 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 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)*, volume 8 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 133–144. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 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 μ-calculus. *Bulletin of the Belgian Mathematical Society*, 8(2), May 2001.

APPENDICES

A Proof of Lemma 4

This section contains the details of the reduction to PEEK-G_6 from the proof of Lemma 4. Note that, while PEEK-G_6 allows the agents to pass, we dislow it for simplicity. This is, however, no restriction: to simulate passing, we could add a single boolean variable for each agent that does not occur in the formula. Passing can then be identified with toggling the value of this variable.

A.1 Full 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 with the following restrictions. Agent 1 (he, for convenience) is the safety agent while Agent 2 (she, for convenience) is the reachability agent.

- $\mathcal{Q} = \{r, t_1, \ldots, t_{h+k}, f_1, \ldots, f_{h+k}, 1, \ldots, k\}$. Specifically, there are two states t_i and f_i for each variable in $P_1 \cup P_2$.
- There are $k + 3$ atomic propositions in $\mathcal{P} = \{s, p, c_1, \ldots, c_k\}$.
- Initial state r is the only state where s is true ($\lambda(q) = \{s\}$ iff $q = r$). For each $i \in [1, h + k]$, $\lambda(t_i) = \{p\}$ and $\lambda(f_i) = \emptyset$. For the remaining states $i \in [1, k]$, we have $\lambda(i) = \{c_i\}$.
- The state r has two successors, t_1 and f_1 in $\mathcal{E}$. For $i < h + k$, both t_i and f_i have two successors, t_{i+1} and f_{i+1}. t_{h+k} and f_{h+k} have $k + 1$ successors, $1, \ldots, k$, and they all have one successor, r.
- t_{h+k} and f_{h+k} belong to a *reachability* agent (rectangular nodes), while all other states belong to a *safety* agent (circular nodes).

Note that $\neg\gamma$ can be rewritten in DNF by dualising the γ (which is in CNF), that is, by swapping conjunctions and disjunctions and negating the literals.

The game is played in rounds. Every time the game is at state r, it enters a new round. Formally, the safety agent makes his moves at states $t_1, \ldots, t_{h+k-1}, f_1, \ldots, t_{h+k-1}, 1, \ldots, k$, and r, while the reachability agent makes her moves at states t_{h+k} and f_{h+k}. The specification we provide, however, will require that the safety agent must change exactly the variable of the reachability agent identified by the state the reachability agent has previously moved to. It also forces the safety agent to make his choice for the *following* round each time at state r, and to make it in a way that the value of exactly one variable is toggled.

For ease of notation we use, for any $i \in \mathbb{N}$, the formula template $\langle + \rangle \bigcirc^{(i)} \psi_1$ to denote a sequence of i successive $\langle + \rangle \bigcirc$ followed by subformula ψ_1. Such formulas are used to assert that ψ_1 is true in i steps of the game.

For this game, we model-check the following formula

$$\phi \overset{\text{def}}{=} \langle 1 \rangle (\theta_1 \wedge \langle + \rangle \Box (\neg s \vee (\theta_2 \wedge \theta_3 \wedge \theta_4))),$$

where θ_1, θ_2, θ_3, and θ_4 reflect the following guarantees:

- θ_1 specifies the correctness of the initial condition. Specifically,

$$\theta_1 \overset{\text{def}}{=} \bigwedge_{p_i \in I} \langle + \rangle \bigcirc^{(i)} p \wedge \bigwedge_{p_i \in P_1 \cup \mathsf{P}_2 - I} \langle + \rangle \bigcirc^{(i)} \neg p$$

- At every occurrence of r, the game enters a round in which the safety agent may toggle at most one of $p_1, \ldots, p_h$. This is specified with θ_2.

  $\theta_2 \overset{\text{def}}{=} \bigvee_{i \in [1,h]} \delta_i \bigwedge_{j \in [1,h], j \neq i} \epsilon_j$, with

  $\delta_i \overset{\text{def}}{=} \big((\langle + \rangle \bigcirc^{(i)} (p \wedge \langle + \rangle \bigcirc^{(h+k+2)} \neg p)\big) \vee \big((\langle + \rangle \bigcirc^{(i)} (\neg p \wedge \langle + \rangle \bigcirc^{(h+k+2)} p)\big)$ and

  $\epsilon_i \overset{\text{def}}{=} \big((\langle + \rangle \bigcirc^{(i)} (p \wedge \langle + \rangle \bigcirc^{(h+k+2)} p)\big) \vee \big((\langle + \rangle \bigcirc^{(i)} (\neg p \wedge \langle + \rangle \bigcirc^{(h+k+2)} \neg p)\big)$.
- The reachability agent declares her choice for a change by selecting a state $i \in \{1, \ldots, k\}$. Choosing $i \in [1,k]$ means the toggling of p_{h+i}. This is specified by θ_3.

  $\theta_3 \overset{\text{def}}{=} \bigwedge_{i \in [1,k]} \eta_i^+ \vee \eta_i^-$ with

  $\eta_i^+ \overset{\text{def}}{=} (\langle + \rangle \bigcirc^{(h+i)} p) \wedge \langle + \rangle \bigcirc^{(h+k+1)} \big((c_i \wedge \langle + \rangle \bigcirc^{(h+i+1)} \neg p) \vee (\neg c_i \wedge \langle + \rangle \bigcirc^{(h+i+1)} p)\big)$ and

  $\eta_i^- \overset{\text{def}}{=} (\langle + \rangle \bigcirc^{(h+i)} \neg p) \wedge \langle + \rangle \bigcirc^{(h+k+1)} \big((c_i \wedge \langle + \rangle \bigcirc^{(h+i+1)} p) \vee (\neg c_i \wedge \langle + \rangle \bigcirc^{(h+i+1)} \neg p)\big)$.
- Globally, at r the formula γ is not satisfied (using the truth of p in i steps for p_i). This is reflected by replacing every literal p_i in $\neg\gamma$ (recall that $\neg\gamma$ is in DNF) by $\langle + \rangle \bigcirc^{(i)} p$ and every literal $\neg p_i$ by $\langle + \rangle \bigcirc^{(i)} \neg p$.

The turn taking and the order of the moves are reflected as well as the competitive nature of the game. It is apparent that the safety agent wins the PEEK game if the safety agent has a strategy scheme σ, and it is easy to translate one into the other.