Model-Checking Iterated Games

Chung-Hao Huang[1] Sven Schewe[2] Farn Wang[1,3,4]

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
4: Center for Information Technology Innovation (CITI), Academia Sinica, Taiwan, ROC

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 NL complete in the model. This advancement over nonelementary logics is bought by disallowing a too close entanglement between the 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* [27, 12] has been reduced to (repeated) reachability [15, 19], the satisfiability checking and synthesis problem has been tightly linked with game theory ever since the seminal works of Büchi and Landweber [8, 7]. Since the seminal work of *alternating time logic* (*ATL*) by Alur, Henzinger, and Kupferman [2] and in automaton-based μ-calculus model-checking (e.g., [36]), game theory has also emerged as a technology with great potential in verifying the correctness of reactive systems. With game theoretic challenges moving to the focus of researchers studying the specification and design of reactive systems, traditional problems of multi-player games are replacing the traditional 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 the execution . 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**, and the *alternating μ-calculus* (*AMC*), *game logic* (*GL*) [2], *strategy logics* [11, 21, 24, 22], *coordination logic* [14], *stochastic game logic* [5], and *basic strategy interaction* *logic* (*BSIL*) [35] 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. It would be interesting to see whether the expressiveness for close interaction among agent strategies and the efficiency for the verification or refutation of compliance with a specification can be better combined.

On the one hand, logics like ATL, ATL*, AMC, and GL [2] 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 [35] that ATL, ATL*, AMC, and GL [2] 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 representing (or reasoning about) real-world examples.

On the other hand, to solve the expressiveness problem in the above example, *strategy logics* (*SL*) were proposed in [5, 11, 24, 22]. 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 [35], 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 reasoning about Nash equilibria.

However, in our opinion, the previous work still lacks some important ingredients for practical specifications in real-world projects. In the following, we use the following example to explain the inadequacy of those languages in expressing real-world properties.

Example 1. Consider the example of a bank (Bank A) that would like to specify their information system embodied as a system security strategy that allows a client to use a strategy to transfer money from another bank (Bank B), and to use a strategy to transfer to Bank B. Moreover, the same system strategy should forbid any illegal operation on the banking system. This example looks similar to the one in the introduction of [35] for the argument that ATL*, AMC, and GL are inadequate in expressing certain important game properties in real-world projects. As explained in [35], such a property involves the interaction of one strategy of Bank A with two different strategies of the client and cannot be expressed with ATL*, AMC, and GL.

However, the logic BSIL proposed in [35] still lacks some important ingredients in specifying fine-grained control of a multi-agent systems. Consider the banking system example in the last paragraph. A typical transfer session usually consists of at least five segments roughly in timing order.

S1: The client and Bank A collaborate to initiate the transfer.
S2: Bank A and Bank B collaborate to carry out the transfer via the internet.
S3: Bank B updates the account balance.
S4: Bank A updates the account balance.
S5: Bank A and B commit the transaction.

As can be seen, the whole session involves the collaboration of the client, Bank A, and Bank B. However, in S1, the strategy of Bank B is irrelevant. In S2 and S5, the strategy of the client is irrelevant. In S3, only the strategy of Bank B matters. In

S4, only the strategy of Bank A matters. Since BSIL does not allow pure strategy interaction quantification inside temporal modalities, it is impossible to express a fine-grained control in specific timing intervals. For example, it is not possible to express in BSIL the following.

- After S1 finishes and before S3 and S4 start, Bank A can use the same strategy that initiates the transfer session and enforces the security to collaborate with Bank B to fulfill the transfer via internet.
- After S2 finishes, Bank B can use the same strategy that collaborates with Bank A to fulfill the internet transfer to again finish updating account balance.

Certainly, BSIL can be viewed as a compromise between expressiveness and its relatively low PSPACE model-checking complexity. Still, the above example motivates us for identifying a more expressive class of strategy logics while maintaining a relatively low model-checking complexity.

We thus propose to adapt the logic in [35] to a new temporal logic called *temporal cooperation logic* (*TCL*) for this purpose. Let us introduce TCL informally on a game with a variation of the classic game problem of Prisoners' dilemma.

Example 2. **Iterated prisoners' dilemma.** Inspired by the famous prisoners' dilemma, we consider a model where three suspects, who are initially in custody, are interrogated 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 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 [4]. The following TCL sentence refers to nice strategies of Prisoner 1.

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

$\langle 1\rangle$ is a *strategy quantifier* (SQ), which states that there exists a strategy of Prisoner 1 to achieve the temporal goal. $\langle +\rangle$ is a *strategy interaction quantifier* (*SIQ*) that inherits the strategy from its parent formula. Proposition $\texttt{betray}_i$ 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.'

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

Similar properties can be used to specify *forgiving*[1] or other related strategies [4]. A forgiving strategy of Prisoner 1 is reflected by the following TCL property.

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

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

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

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

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, and BSIL. While it can be expressed with SL, the verification complexity of SL is prohibitively high at 2EXPTIME-hard [21].

In [35], SIQs can neither override or revoke strategies assigned by SQ or SIQ 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 $\texttt{jail}_a$ be a proposition, which says that for prisoner a in jail. In TCL,

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

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 (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 [5, 11, 21, 24] 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 is thus a step toward a better balance between expressiveness and complexity/efficiency compared to ATL*, GL [2], and SL [11, 24, 22]. Given the expressive power as exemplified by the specifications from above, TCL can be viewed as an expressive yet inexpensive subclass of SL [24, 22].

Organization or the Paper. Section 2 reviews related work. Section 3 explains concurrent game graphs and turn-based game graphs for the description of multi-agent systems and section 4 presents the syntax and semantics of TCL. Section 5 discusses the expressiveness of TCL. In particular, we establish 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 incomparable with ATL*, AMC, and GL [2] in expressiveness, and discuss the effect of a mild extension of TCL. In the following sections, we develop an automaton-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 Related work

2.1 Before strategy logics

Alur, Henzinger, and Kupferman presented ATL (alternating-time temporal logic), ATL*, AMC (alternating-time μ-calculus), and GL (game logic) with strategy quantifier $\langle\!\langle M \rangle\!\rangle$ [2].

Walther, van der Hoek, and Michael Wooldridge presented ATLES that allows a specified strategy to interact with strategy quantification [34]. Specifically, we can write ATLES formulas of the form:

$$\langle 1 \rangle_{\{1 \mapsto \rho_1\}}((\langle 2 \rangle_{\{1 \mapsto \rho_1\}} \Box p) \wedge \langle 2 \rangle_{\{1 \mapsto \rho_1\}} \Diamond \neg p)$$

where ρ_1 is the symbolic name for a strategy of agent 1 and $\langle A \rangle_{\{b \mapsto \rho_b\}} \phi$ means that the collaborating strategy of agents in $A \setminus \{b\}$ can work with agent b with strategy ρ_b to enforce ϕ. Intuitively, ρ_b functions as a working variable that records the same strategy used across several strategy quantifiers. In the complexity aspect, [34] only considers memoryless strategies in the model-checking problem. If ρ_1 is prescribed, then their PTIME model-checking algorithm in [34] can check whether ρ_1 satisfies the formula. If ρ_1 is not prescribed, their model-checking algorithm runs in NP. As stated in [34], specifiying fine-grained interaction among strategies can result in combinatorial explosion in formula sizes.

Brihaye, Lopes, Laroussinie, and Markey introduced a very expressive extension to ATL* with other players' strategies and memory constraints [6]. They also showed that the model-checking problem of this extension is decidable, in fact, PSPACE-complete for memoryless strategies and EXPSPACE for strategies with bounded memory.

Pinchinat introduced a way to specify expressive constraints on strategies in concurrent games by extending μ-calculus with decision modalities [26]. Laroussinie and Markey reported decidability of satisfiability problems in this direction with new context constraints, e.g., the number of moves by agents are bound [20].

2.2 Strategy logics

Chatterjee, Henzinger, and Piterman introduced a strategy logic allowing for first-order quantification over strategies [11]. The decision procedure is however non-elementary.

Mogavero, Murano, Perelli, Sauro, and Vardi also identified fragments of strategy logics which enjoy a doubly exponential time complexity model-checking algorithm [24, 22, 23]. For example, in [23], conjunction and disjunction cannot happen in the same scope of strategy quantification. TCL is also a fragment of strategy logic with restriction on the hierarchy of SIQs and sometimes can be handy in expressing boolean relation among strategies. Moreover, the restriction on our TCL results in a much lower model-checking complexity of EXPTIME.

Another interesting extension to ATL is ATL$^*_{sc}$ by Laroussinie, Lopes, and Markey [21] that interprets nested strategy quantification as composition, instead of revocation, of strategy profiles. This is similar to the semantics of our SIQs. However, TCL keeps both the SQs from ATL (for strategy profile revocation) and new SIQs (for strategy profile composition). In this way, TCL does not subsume ATL* while still allows for flexible and practical specifications.

Moreover, with their special interpretation of nested strategy quantification, ATL$^*_{sc}$ [21] allows for strategy interaction across temporal modal operators and makes ATL$_{sc}$ as expressive as ATL$^*_{sc}$. As a result, the model-checking complexity of ATL$_{sc}$ is also 2EXPTIME. In contrast, TCL allows for strategy interaction in the same state while strictly forbids strategy interaction across temporal modal operators. As a result, TCL is carefully designed for a EXPTIME model-checking complexity while maintaining some practical expressiveness.

BSIL [35] is a strict subclass of our TCL and enjoys a lower model-checking complexity than TCL. But BSIL did not admit strategy quantification to interact across temporal modalities.

TCL and the various fragments of strategy logics all can express certain interaction and collaboration relation among strategies used by agents in a multi-agent game. They are designed with different consideration in balancing between expressiveness and verification efficiency. TCL is also special in that its design takes the natural way that strategy collaboration can happen in rounds and need be specified with this consideration.

2.3 Tools

There are several model-checkers for ATL, including MOCHA [3] and ATL Designer [31, 32]. RB±ATL is an extension to ATL with resource bound requirement [1]. However, unlike our model-checker for TCL and BSIL, those tools can only reason with memoryless strategies specified with ATL.

One model-checker that allows for the specification of strategy logics for close strategy interaction is MCMAS-SLK [9] which also incorporates syntax for epistemic requirement. However, MCMAS-SLK only synthesizes memoryless strategies.

As far as we know, our model-chcker for TCL and BSIL is the only one for memoryful strategies for a non-trivial fragment of strategy logics. It is thus not meaningful to compare the performance of our model-checker with the model-checkers mentioned in the above since the strategies that our implementation synthesized in the experiment cannot be synthesized by the other model-checkers.

3 System models

A vector of dimension m is a sequence of m elements. Notationally, we use $\overrightarrow{d}$ to denote a vector and $\overrightarrow{d}[1], \ldots, \overrightarrow{d}[m]$ for its m elements.

3.1 Concurrent games

A general framework of game structure is concurrent game that allows the agents to make concurrent decisions. A *concurrent* game is played by a finite number m of agents, indexed 1 through m. A game is a tuple $\mathcal{G} = \langle m, \mathcal{Q}, r, P, \lambda, \Sigma, \delta \rangle$, 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 $\mathcal{G}$.

- P is a finite set of atomic propositions.
- $\lambda : \mathcal{Q} \to 2^P$ is a proposition labeling function.
- $\Sigma = S_1 \times \ldots \times S_m$ is a set of joint decisions where S_a is a set of decisions available to an agent $a \in [1, m]$. Conceptually, elements in Σ are vectors of m dimensions with their a'th elements from S_a.
- $\delta : Q \times \Sigma \to Q$ is a transition function that maps a state q and a joint decision $\overrightarrow{d} \in \Sigma$ to the respective successor state $\delta(q, \overrightarrow{d})$ reached from q on the joint decision $\overrightarrow{d}$.

During a transition, each agent selects a token. For the convenience of presentation and proof of the lemmas, usually δ is defined for all token vectors. In practice, when some token vectors are not admitted, we may introduce an artificial failure state as the destination state or simply add self-loop transitions for those undefined token vectors.

For convenience, we let $E_G = \{(q, \delta(q, \overrightarrow{d})) \mid q \in Q, \overrightarrow{d} \in \Sigma\}$ be the set of unlabeled transitions.

In Figure 1, we have the graphical representation of a concurrent game graph. The

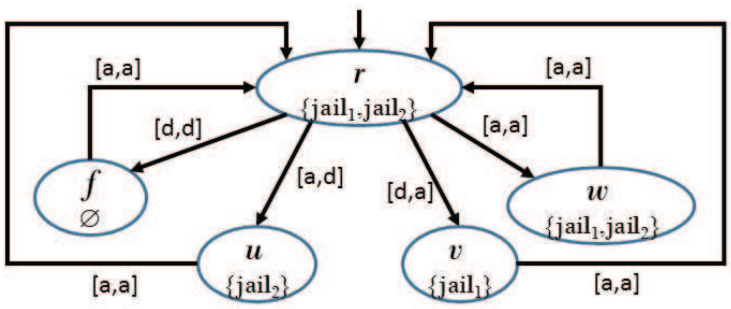


Fig. 1. A concurrent game graph for the iterated prison dilemma $m = 2$; $S_1 = S_2 = \{a, d\}$; '*' represents the set of all tokens.

ovals represent states while the arcs represent state transitions. We also put down the λ values inside the corresponding states. The transitions are labeled with the decisions, d for deny and a for admit, made by the agents, shown as [*decision of agent* 1,*decision of agent* 2]. $[*, *]$ represents the set of all token vectors.

For convenience, in the remaining part of the manuscript, we assume that we are always in the context of a given game graph $\mathcal{G} = \langle m, \mathcal{Q}, r, \mathcal{P}, \lambda, \Sigma, \delta \rangle$ with $\mathcal{E} = E_{\mathcal{G}}$.

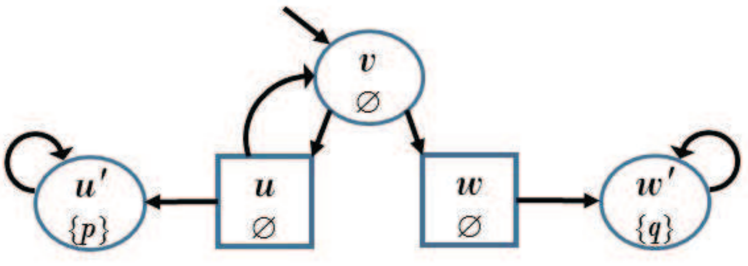
A *play* ρ is an infinite path $q_0 q_1 \ldots$ in $\mathcal{G}$ such that, for every $k \in \mathbb{N}$, $(q_k, q_{k+1}) \in \mathcal{E}$. ρ is *initial* if $q_0 = r$. For every $k \geq 0$, we let $\rho(k)$ denote q_k. Also, given $h \leq k$, we let $\rho[h, k]$ denote $\rho(h) \ldots \rho(k)$ and $\rho[h,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

Given a vector $\overrightarrow{d}$ with m elements, we use $\overrightarrow{d}[a]$ to denote the a'th element of $\overrightarrow{d}$. A *strategy* κ is a function from $\mathcal{Q}^*$ to Σ. A play ρ is *compatible* with a strategy κ_a of an agent $a \in [1, m]$ if, and only if, for every $k \in \mathbb{N}$, there exists a $\overrightarrow{d}$ such that $\overrightarrow{d}[a] = \kappa_a(\rho[0, k])$ and $\delta(\rho(k), \overrightarrow{d}) = \rho(k + 1)$.

3.2 Turn-based game graphs

Another popular game structure is *turn-based game* in which at each state, at most one agent gets to decide the next state. For example, in Figure 2, we have the graphical representation of a turn-based game graph. The ovals and squares represent states owned



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

Fig. 2. A turn-based game graph

by agent 1 and 2 respectively. The arcs represent state transitions. Many popular games, e.g., chess, GO, Bridge, Queen of Spades, etc., are turn-based. In the investigation of many research issues, turn-based game graphs are easier to handle than concurrent game graphs since there tends to be less decision vectors from each state. Moreover, many turn-based games are determined[2] while many concurrent games are not and may need randomization for meaningful analysis [10].

In fact, turn-based games can be represented as special cases of concurrent games. Specifically, a turn-based game $\mathcal{G} = \langle m, \mathcal{Q}, r, P, \lambda, \Sigma, \delta \rangle$ is a concurrent game such that for every $q \in \mathcal{Q}$, there is an $a \in [1, m]$, called the owner of q, such that for every $a' \neq a$, the decision of agent a' at q does not affect the destination state. Specifically, for every two token vectors $\overrightarrow{d}$ and $\overrightarrow{d'}$, if a is the owner of q and $\overrightarrow{d}[a] = \overrightarrow{d'}[a]$, then $\delta(q, \overrightarrow{d}) = \delta(q, \overrightarrow{d'})$. For convenience, for a turn-based game, we use *ownerSat*(a, q) to denote the following predicate that says agent a can be a owner of state q.

$$\forall \overrightarrow{d} \in \Sigma\ \forall \overrightarrow{d'} \in \Sigma\ \left(\overrightarrow{d}[a] = \overrightarrow{d'}[a] \to \delta(q, \overrightarrow{d}) = \delta(q, \overrightarrow{d'}) \right).$$

[2] A game is determined if a player knows whether she/he has a winning strategy even before the game is played.

Then the owner of q, in symbols $\omega(q)$, is defined as the minimum agent index a satisfying $\mathit{ownerSat}(a, q)$. 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 fact, there are temporal logic formulas that cannot be satisfied by turn-based game structures. For example, one such ATL formula is the following for a 2-player game.

$$(\langle 1, 2\rangle \bigcirc p) \wedge (\neg\langle 1\rangle \bigcirc p) \wedge (\neg\langle 2\rangle \bigcirc p) \qquad (\#)$$

The reason is the following, Assuming a turn-based game, the second conjunct $\neg\langle 1\rangle \bigcirc p$ and the third conjunct $\neg\langle 2\rangle \bigcirc p$ together say that no matter whether the initial state is owned by player 1 or player 2, p is false in all successor states. Thus, the first conjunct $\langle 1, 2\rangle \bigcirc p$ cannot be satisfied. But the formula can be satisfied by the concurrent game in figure 3.

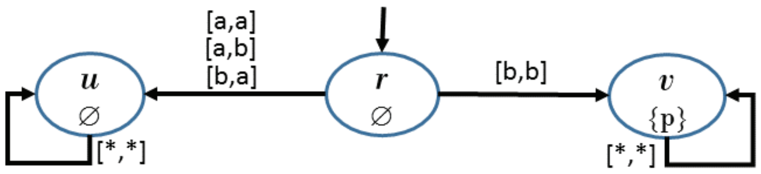


Fig. 3. A concurrent game graph satisfying formula (#)
$m = 2$; $S_1 = S_2 = \{a, b\}$; '*' represents the set of all tokens.

4 TCL

4.1 Syntax

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 TCL formula ϕ is constructed with the following three syntax rules.

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

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.

Property $\langle -A\rangle\psi_1$ is an strategy interaction quantification (SIQ) that revokes the binding of strategies to agents in A from the context. For example, $\langle 1,2,3\rangle p \mathrm{U} \langle -2\rangle \bigcirc r$ specifies that a state q satisfying $\langle -2\rangle \bigcirc r$ will be reached via collaboration of agents 1, 2, and 3. However, the fulfillment of $\bigcirc r$ at state q will only require the collaboration of agents 1 and 3.

'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.

4.2 Rationale for the design of TCL

As explained in the introduction, BSIL does not admit strategy interaction to cross temporal modality and TCL is aimed to gain expressiveness in this aspect with a balance between expressiveness and verification efficiency in mind. We observe that if we can always limit our attention to a bounded number of strategies and analyze their interaction in choosing the next states, the techniques for model-checking BSIL formulas can still apply.

If there is no alternation in the quantification of strategies, then the number of strategies that we need consider is bounded by the number of strategy quantifications in the formula. For example, in formula

$$\langle 1\rangle \bigcirc ((\langle +2\rangle p) \wedge \langle +2\rangle \neg p),$$

we only have to consider three strategies.

- κ_1 for agent 1 from the initial state.
- κ_2, κ_3 for agent 2 from the next state.

Then from the next state on in every play, we can reason the interaction among κ_1, κ_2, and κ_3 by investigating how they issue tokens in a state. Similarly, for formula

$$\langle 1\rangle((\Diamond\langle +2\rangle p) \wedge \Diamond\langle +2\rangle \neg p),$$

we only have to consider the interaction among three strategies from the destination state of the eventualities along every play. In fact, for any nesting depth of the eventuality modality, it is easy to see that we still only have to focus on bounded number of strategies.

In contrast, when there is an alternation of strategy declaration, the number of strategies that we need to consider may become unbounded. For example, in formula

$$\langle 1\rangle((\langle +2\rangle\Diamond p) \wedge \neg\langle +2\rangle\Diamond\neg p),$$

for a fixed strategy κ_1 of agent 1, we need to prove that no strategy of agent 2 can collaborate to fulfill $\Diamond\neg p$. In the worst case of such nestings, this will force us to consider the patterns of interaction between κ_1 and infinitely many strategies of agent 2. As can be seen from the strategy logics, this can easily blow up the model-checking complexity to 2EXPTIME. Thus, the first thing that we need to constrain is no strategy quantification alternation is allowed until all strategies are revoked by an SQ.

However, there is a subtle strategy quantification alternation embedded with temporal modalities. Consider formula

$$\langle 1\rangle(((\langle +2\rangle\Diamond p) \wedge \Box\langle +2\rangle\neg p).$$

The strategy that agent 2 used to enforce $\neg p$ can can be different from state to state due to the modality of $\Box$. In the worst case, such nestings can also force us to enumerate different patterns of interaction between one strategy of agent 1 and all strategies of agent 2. This can also blow up the model-checking complexity of TCL to those of strategy logics. The same argument also applies to modality $\phi \mathrm{U} \psi$ when an SIQ happens in the path subformula ϕ.

To summarize, we observe that to maintain relatively low model-checking complexity of TCL, we need to do the following.

- No alternation of non-trivial SIQs is allowed.
- No implicit alternation of SIQs induced by nontrivial SIQs inside $\Box$-modalities and the path subformulas of U-modalities.

The above two constraints lead us to the following definition of TCL.

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.

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

In general, it would also be nice to have the universal SQs and SIQs respectively as duals of existential SQs and SIQs. Could we not 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 negations.

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

4.3 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 either $\psi = \bigcirc\psi_1$ or $\psi = \psi_1 \mathsf{OP} \psi_2$, 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.

Function *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 unraveling $\mathcal{G}$ from r and expand the ownership and labeling functions from $\mathcal{G}$ to T_r in the natural way. Technically, we have the following definition.

A *computation tree* for a 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 maximal subset of T_q such that:

- $\pi \in T_q\langle \pi, \sigma, \psi_1\rangle$;
- for all $\pi' \in T_q\langle \pi, \sigma, \psi_1\rangle$ and $\pi' q' \in T_q\langle \pi, \sigma, \psi_1\rangle$, there exists a $\overrightarrow{d} \in \Sigma$ such that $\delta(\mathit{last}(\pi'), \overrightarrow{d}) = q'$ and for all $a \in \mathit{bnd}(\psi_1)$, $\sigma[\psi_1, a](\pi') = \overrightarrow{d}[a]$.

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 $\mathcal{G}, \mathit{last}(\pi) \models \phi$, 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 unraveling 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.

5 Expressiveness of TCL

Note that TCL is not a super-class of BSIL since BSIL allows for negation in front of SIQs while TCL does not. However, by examining the proofs in [35] for the inexpressibility of BSIL properties by ATL*, GL, and AMC, we find that the BSIL property used in the proof, i.e.,

$$\langle 1 \rangle ((\langle +2 \rangle \Box p) \wedge \langle +2 \rangle \Box q)$$

is in fact also a property in 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.*

Proof. By the same proof in [35] for the for the inexpressibility of BSIL properties by 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 straightforward way. ATL and CTL can be viewed as syntactic fragments of TCL.

But it is also simple to embed LTL and even CTL*.

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

Proof. Since LTL is a subclass of CTL*, we only have to prove that TCL is more expressive than CTL*. Following lemma 1, we know that CTL* is not as expressive as TCL since CTL* is a subclass of ATL*.

Now we want to prove that every CTL* formula can be converted to an equivalent TCL formula. The basic intuition is that in CTL* properties, the universal path quantification $\forall$ is equivalent to SQ $\langle\ \rangle$ saying that no collaboration is needed to enforce the property following the SQ. Symmetrically, the existential path quantification $\exists$ is equivalent to SQ $\langle 1, \ldots, m\rangle$ saying that a unique path is collaboratively selected by all agents to enforce the property following the SQ.

Specifically, we can convert any CTL* formula ϕ to an equivalent TCL formula $C^*(\phi, 0)$ according to the following inductive definition.

- $C^*(\mathit{true}, 0) = \mathit{true}$.
- $C^*(\mathit{false}, 0) = \mathit{false}$.
- $C^*(\mathit{true}, 1) = \mathit{false}$.
- $C^*(\mathit{false}, 1) = \mathit{true}$.
- $C^*(p, 0) = p$.
- $C^*(p, 1) = \neg p$.
- $C^*(\exists\psi, 0) = \langle 1, \ldots, m\rangle C^*(\psi, 0)$.
- $C^*(\exists\psi, 1) = \langle\ \rangle C^*(\psi, 1)$.
- $C^*(\forall\psi, 0) = \langle\ \rangle C^*(\psi, 0)$.
- $C^*(\forall\psi, 1) = \langle 1, \ldots, m\rangle C^*(\psi, 1)$.
- $C^*(\neg\psi_1, b) = C^*(\psi_1, 1-b)$.
- $C^*(\psi_1 \vee \psi_2, 0) = C^*(\psi_1, 0) \vee C^*(\psi_2, 0)$.
- $C^*(\psi_1 \vee \psi_2, 1) = (C^*(\psi_1, 1) \wedge C^*(\psi_2, 1)$.
- $C^*(\psi_1 \wedge \psi_2, 0) = C^*(\psi_1, 0) \wedge C^*(\psi_2, 0)$.
- $C^*(\psi_1 \wedge \psi_2, 1) = (C^*(\psi_1, 1) \vee C^*(\psi_2, 1)$.
- $C^*(\bigcirc\psi_1, b) = \langle +\rangle \bigcirc C^*(\psi_1, b)$.
- $C^*(\psi_1 \mathrm{U} \psi_2, 0) = \langle +\rangle C^*(\psi_1, 0) \mathrm{U} C^*(\psi_2, 0)$.
- $C^*(\psi_1 \mathrm{U} \psi_2, 1) = \langle +\rangle (C^*(\psi_1, 1) \wedge C^*(\psi_2, 1)) \mathrm{R} C^*(\psi_2, 1)$.
- $C^*(\psi_1 \mathrm{R} \psi_2, 0) = \langle +\rangle C^*(\psi_1, 0) \mathrm{R} C^*(\psi_2, 0)$.
- $C^*(\psi_1 \mathrm{R} \psi_2, 1) = \langle +\rangle (C^*(\psi_1, 1) \wedge C^*(\psi_2, 1)) \mathrm{U} C^*(\psi_2, 1)$.

Then according to the semantics of CTL*, it is clear that ϕ specifies the same set of traces that $C^*(\phi, 0)$ does.

Together with lemma 1, thus the lemma is proven. ■

Lemma 2 shows the advantage of TCL over BSIL [35] which is not more expressive than CTL* and LTL. However, this conversion does not extend to ATL*. In fact, we have the following lemma.

Lemma 3. *ATL* and TCL are incomparable in expressiveness.*

Proof. ATL* is not as expressive as TCL according to lemma 1. The following example shows an ATL* property that cannot be expressed with TCL.

$$\langle 1\rangle((\Box p) \vee \Box q) \tag{*}$$

The property says that there exists a strategy of agent 1 that enforces either $\Box p$ or $\Box q$. Note that this is different from the ATL property $(\langle 1\rangle\Box p) \vee \langle 1\rangle\Box q$, which says that two strategies of agent 1 respectively enforce $\Box p$ and $\Box q$. Property (*) is also different from the TCL property $\langle 1\rangle((\langle +\rangle\Box p) \vee \langle +\rangle\Box q)$, which says that there exists a strategy of agent 1 that either enforces $\Box p$ along all plays or enforces $\Box q$ along all plays.

Rigorously speaking, we can employ an inductive argument that shows TCL formulas of any number of operators (boolean, temporal, or strategy quantifications) cannot express ATL* formula (*). Note that since only one agent is used in the strategy quantification in formula (*), there is no point to consider TCL formulas with SQs and SIQs other than $\langle 1\rangle$ and $\langle +\rangle$. But since all such SQs and SIQs will be succeeded by exactly one temporal operator in TCL, all paths in the strategy tree cannot guarantee to satisfy two path modalities. Thus we can build our arguments from the base case of atomic propositions, and then from the inductive case of formulas with $k + 1$ operators. This inductive argument then suffices to prove that formula (*) is not expressible by any TCL formula. ■

In fact, the proofs and examples from [35] can also be applied in this work to show that there are properties of GL and AMC that cannot be expressed with TCL. This leads to the following lemma.

Lemma 4. *TCL is incomparable in expressiveness with either of GL and AMC.*

Proof. Due to lemma 1, we know that GL and AMC are not as expressive as TCL. Then with the same argument in [2], we can prove that TCL is as expressive as neither GL nor AMC. ■

Note, however, that allowing for boolean negations preceding SIQs 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 $\langle A\rangle\neg\langle +[1,m] \setminus A\rangle C^*(\psi, 1)$ 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.

Moreover, it is easy to reduce the realizability problem of prenex *QPTL* (*Quantified Propositional Temporal Logic*) [17], and hence a non-elementary problem, to the model-checking problem of this extended version of TCL. For convenience, TCL extended with negation operator in front of SIQs is called *extended TCL* (*ETCL*).

Lemma 5. *The model-checking problem of ETCL is non-elementary.*

Proof. 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 $\langle -1 \rangle \langle +1 \rangle \bigcirc (p_i \wedge \langle + \rangle \bigcirc \neg p_i)$,

and by preceeding every temporal operator in ϕ by an empty SIQ $\langle + \rangle$.

For the reduction of checking validity of the prenex QPTL formula from above (or: realizability, as the prenex QPTL formula has no free variables this is the same thing) to model-checking ϕ' on the game shown in Figure 4, we first recall that '$\forall p$' is an abbreviation of '$\neg \exists p \neg$', such that $\forall p_2 \exists p_3 \forall p_4 \ldots \exists p_n \phi$ is equivalent to

$$\neg \exists p_2 \neg \exists p_3 \neg \exists p_4 \ldots \exists p_n \phi.$$

We then take a look at the game shown in in Figure 4. The game has n nodes, agents, and atomic propositions. The nodes in Figure 4 are labeled with the agent that owns the nodes, and the atomic proposition p_i is true exactly in node i. From his state, Agent 1 can move to any node, while all other agents can either stay in their node or return to the node owned by Agent 1.

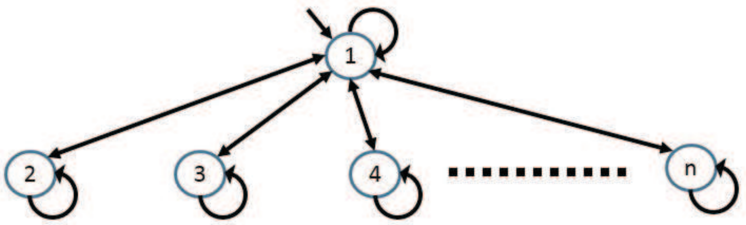


Fig. 4. The turn-based game graph from the non-elementary hardness proof of extended TCL.

Considering our translation, the first important observation is that the game starts in the node owned by Agent 1, and in order to comply with the specification, $\langle 1 \rangle \Box p_1$ must be satisfied. The outermost strategy chosen by Agent 1 must therefore choose to stay in the initial state forever. The (only) relevant main path is therefore 1^ω, the path that simply loops in node 1. (Decisions of Agent 1 on places different from 1^* have no influence on acceptance, see below.)

The next relevant observation is that ψ_ϕ changes ϕ at the level of its literals. (Recall that we have defined the temporal logic such that it is in negative normal form. Each of these literals therefore occurs positively.) The way the literals are changed, p_i to

$\langle -1\rangle\langle +1\rangle \bigcirc (p_i \wedge \langle +\rangle \bigcirc p_i)$ and $\neg p_i$ to $\langle -1\rangle\langle +1\rangle \bigcirc (p_i \wedge \langle +\rangle \bigcirc \neg p_i)$, consists of an opening part, $\langle -1\rangle\langle +1\rangle \bigcirc (p_i \wedge \ldots$ that says "we can *change* the strategy of Agent 1 such that, in the next move, p_i holds and ...". Reconsidering the only feasible strategy for Agent 1 (which is revoked by the leading $\langle -1\rangle$ operation), this operation is interpreted after an initial sequence of visits to node 1. From node 1, there is exactly one continuation, the continuation to node i, that will satisfy the $\langle +1\rangle \bigcirc p_i$ part. Whether or not the continuation '$\langle +\rangle \bigcirc p_i)$' or '$\langle +\rangle \bigcirc \neg p_i)$', respectively, holds entirely depends on the strategy of Agent i as fixed in the prenex part of the extended TCL formula.

Thus, on each position of the main path 1^ω, either $\langle -1\rangle\langle +1\rangle \bigcirc (p_i \wedge \langle +\rangle \bigcirc p_i)$ or $\langle -1\rangle\langle +1\rangle \bigcirc (p_i \wedge \langle +\rangle \bigcirc \neg p_i)$ holds when evaluating ψ_ϕ. This is determined by the strategy selected by $\langle +i\rangle$ in the prenex part of the extended TCL formula. Moreover, two strategies of Agent i that agree on their choices (to stay in node i or to return to node 1) on all histories of the form 1^*i are equivalent for all formulas ψ_ϕ. For the only viable outermost strategy of Agent 1 to stay in node 1, this provides an obvious bijection between the selection of assignments to p_i in the infinite word in the prenex QPTL formula and the selection of strategy classes for the extended TCL formula from our translation, interpreted over the turn-based game graph from Figure 4. Consequently, the prenex QPTL formula is valid / realizable if, and only if, the turn-based game graph from Figure 4 is a model of its translation to ETCL. ■

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

6 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 for fixed formulas, while the satisfiability problem for TCL 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* [33] as a de-facto sublanguage.

6.1 Automata and tree models

Trees. We use trees as a representation for a simple class of models. As usual, a (full) *tree* is given as the set Υ^* of all finite words over a given set of directions Υ.

For given finite sets Π and Υ, a *Π-labeled Υ-tree* is a pair $\langle \Upsilon^*, l\rangle$ with a labeling function $l : \Upsilon^* \to \Pi$ that maps every node of Υ^* to a letter of Π. For a set $\Xi \times \Upsilon$ of directions and a node $x \in (\Xi \times \Upsilon)^*$, $\mathsf{hide}_\Upsilon(x)$ denotes the node in Ξ^* obtained from x by replacing (ξ, υ) by ξ in each letter of x.

For a $\Sigma \times \Pi$-labeled Υ-tree $\langle \Upsilon^*, l\rangle$ we call the Σ-labeled Υ-tree $\langle \Upsilon^*, l_\Sigma\rangle$ and the Π-labeled Υ-tree $\langle \Upsilon^*, l_\Pi\rangle$ with $l(D) = \big(l_\Sigma(D), l_\Pi(D)\big)$ for all $D \in \Upsilon^*$ the Σ and Π projection, respectively, of $\langle \Upsilon^*, l\rangle$, denoted $\mathsf{proj}_\Sigma\big(\langle \Upsilon^*, l\rangle\big)$ and $\mathsf{proj}_\Pi\big(\langle \Upsilon^*, l\rangle\big)$, respectively.

For a $\mathit{\Pi}$-labeled $\mathit{\Xi}$-tree $\langle \mathit{\Xi}^*, l\rangle$ we define the $\mathit{\Upsilon}$-widening of $\langle \mathit{\Xi}^*, l\rangle$, denoted by $\mathsf{wide}_{\mathit{\Upsilon}}(\langle \mathit{\Xi}^*, l\rangle)$, as the $\mathit{\Pi}$-labeled $\mathit{\Xi} \times \mathit{\Upsilon}$-tree $\langle (\mathit{\Xi} \times \mathit{\Upsilon})^*, l'\rangle$ with $l'(x) = l(\mathsf{hide}_{\mathit{\Upsilon}}(x))$. In $\mathsf{wide}_{\mathit{\Upsilon}}(\langle \mathit{\Xi}^*, l\rangle)$, nodes that are indistinguishable for someone who cannot observe $\mathit{\Upsilon}$ (i.e., nodes x, y with $\mathsf{hide}_{\mathit{\Upsilon}}(x) = \mathsf{hide}_{\mathit{\Upsilon}}(y)$) have the same label.

Trees are a useful representation for the unraveling of a concurrent game structure. For the concurrent game structure $\mathcal{G} = \langle m, \mathcal{Q}, r, P, \lambda, \mathit{\Sigma}, \delta\rangle$, the $\mathcal{Q}$-labeled $\mathit{\Sigma}$-tree $\langle \mathit{\Sigma}^*, l_{\mathcal{G}}\rangle$ where $l_{\mathcal{G}}$ is inductively defined such that $l_{\mathcal{G}} : \varepsilon \mapsto r$ maps the root of the tree to the initial state of the concurrent game structure and $l : D \mapsto q$ implies $l : D \cdot d \mapsto \delta(q, d)$ for all $D \in \mathit{\Sigma}^*$, $d \in \mathit{\Sigma}$, and $q \in \mathcal{Q}$, is called the state unraveling of $\mathcal{G}$, and the 2^P-labeled $\mathit{\Sigma}$-tree $\langle \mathit{\Sigma}^*, l\rangle$ with $l : D \mapsto \lambda\big(l_{\mathcal{Q}}(D)\big)$ is called the *unraveling* of $\mathcal{G}$. Note that this is a more expansive notation of the computation tree T_r obtained from unravelling $\mathcal{G}$ from the root as discussed in Section 4. The notation in this section is the standard notation for automata constructions. It has been chosen to make the connection to the standard automata constructions clear.

Automata. ω-automata are finite automata that are interpreted over trees—or on concurrent game structures, where they are interpreted over their unraveling. They recognize ω-regular languages.

Alternating parity automata are tuples $\mathcal{A} = (Q, \mathit{\Pi}, \mathit{\Upsilon}, q_0, \delta, \mathsf{pri} : Q \to \mathbb{N})$, where

- Q is a finite set of states and $q_0 \in Q$ is a designated initial states,
- $\mathit{\Pi}$ is an input alphabet
- $\mathit{\Upsilon}$ is a set of directions,
- $\delta : Q \times \mathit{\Pi} \to \mathbb{B}^+(Q \times \mathit{\Upsilon})$ is a transition function that maps states and input letters to positive boolean combinations of states and directions, and
- $\mathsf{pri} : Q \to \mathbb{N}$ is a mapping from the states to the nonnegative integers.

Such an alternating automaton is interpreted over a $\mathit{\Pi}$-labeled $\mathit{\Upsilon}$-tree $\langle \mathit{\Upsilon}^*, l\rangle$. Acceptance for such an automaton can be defined in several equivalent ways. We define it as acceptance games played between a verifier and a falsifier on $Q \times \mathit{\Upsilon}^*$ that start in (q_0, ε). From a state $(q, u) \in Q \times \mathit{\Upsilon}^*$, the verifier (resp. falsifier) wins immediately if $\delta\big(q, l(u)\big)$ is true (resp. false). Otherwise, the verifier selects a (minimal) subset S of $Q \times \mathit{\Upsilon}$ that satisfies $\delta\big(q, l(u)\big)$ and the falsifier selects an element $(q', \upsilon) \in S$ of this subset. The game then proceeds to $(q', u \cdot \upsilon)$. This way, the verifier and the falsifier jointly construct a play $(q_0, u_0)\, S_0\, (q_1, u_1)\, S_1\, (q_2, u_2)\, S_2\, (q_3, u_3)\ \ldots$. The verifier wins if $\limsup_{i\to\infty} \mathsf{pri}(q_i)$ is even, and the falsifier wins if $\limsup_{i\to\infty} \mathsf{pri}(q_i)$ is odd. Such games are memoryless determined [13]: one of the player has a winning strategy, where his or her moves (the selection of a subset and the selection of an element of this subset, respectively) only depend on the current state $(q, u) \in Q \times \mathit{\Upsilon}^*$ (verifier) or the current set $S \subseteq Q \times \mathit{\Upsilon}$ (falsifier). (The set of satisfying assignments can be viewed as a state in the acceptance game.)

$\langle \mathit{\Upsilon}^*, l\rangle$ is *accepted* by $\mathcal{A}$ if, and only if, the verifier has a winning strategy. The set of trees accepted by the automaton is called its *language*.

Parity automata are called Büchi automata if the image of pri is in $\{1, 2\}$, Co-Büchi automata if the image of pri is in $\{0, 1\}$, and weak if all minimal satisfying assignments of $\delta(q, \pi)$ contain only sets with elements (q', υ), where $\mathsf{pri}(q') \geq \mathsf{pri}(q)$. (Weak automata can easily be rewritten as language equivalent Büchi and Co-Büchi automata.)

Special cases of alternating automata are

– universal automata, which are automata where all functions in the image of δ have a unique minimal satisfying assignment, and
– nondeterministic automata, which are automata where all functions in the image of δ contain only minimal satisfying assignments $S \subseteq Q \times \Upsilon$ that satisfy $(q,\upsilon),(q',\upsilon) \in S \Rightarrow q = q'$.

Automata that are both universal and nondeterministic are called deterministic.

Nondeterministic automata intuitively sent (at most) one state into each direction. A given strategy of the verifier therefore defines a run tree, a Q-labeled[3] Υ tree, which is accepting if, for every path on the tree, the highest priority of the labels that occur infinitely often on the path is even.

6.2 Model checking complexity

We show EXPTIME hardness by a reduction from the PEEK-G_6 [30] 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 6. *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 5, where Agent 1 (he, for convenience) represents the safety agent, while Agent 2 (she, for convenience) represents the reachability agent. States t_{h+k} and f_{h+k} are the only states owned by Agent 2.

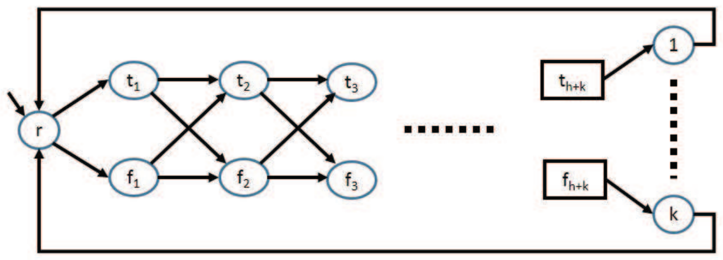


Fig. 5. The turn-based game graph from the EXPTIME hardness proof.

[3] If immediate acceptance is possible, a $Q \cup \{\text{accept,reject}\}$ labeled tree. In this case, the complete subtrees rooted in a state labeled with 'accept' (resp. 'reject') are also labeled with 'accept' (resp. 'reject'), where 'accept' is assigned an even and 'reject' an odd priority.

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 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 [16] naturally reduces to showing $\langle 1\rangle\Diamond p$.

Lemma 7. *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 automaton-based argument. For this, we first describe how to adjust the model to obtain a simple automaton-based model checking procedure. We then recall the basic automata transformations required to turn this automaton into an automaton that runs on the model itself.

Outline. Before discussing the details of the model transformation, we describe what we want to achieve and the problems that need to be addressed for model checking and satisfiability checking.

Especially in satisfiability checking, it is often convenient to encode all information required in the tree the automaton reads when using automata to produce a standard model. Similarly, it is often easier for model checking to start with model checking such rich models, and then use standard techniques like projection and narrowing to remove additional information.

When checking the satisfiability of a CTL formula, for example, a standard model has one direction for each path quantifier, and encodes the truth value of each state formula in the label. A proof obligation to show that a formula $E\phi$ holds is always send into the direction identified with the subformula $E\phi$.

For alternating time logics, it would be attractive to have a model that also encodes the strategies. The problem with this is that, while the simple strategy mentioned above for CTL is always encodable, states of a model of an alternating time logic normally refers to an unbounded number of strategies. In order to be able to represent the strategies for satisfiability checking, the key ingredient is to supply every agent with two choices for each strategy name it may use. (Recall that every agent a has a separate strategy name for each SQ and SIQ, where a strategy for a is introduced.) It has two such names (rather than one) to distinguish newly introduced and continued strategies. Using this, every node is reachable under at most one strategy for agent a, and the node in the tree where this strategy has been introduced can be identified. This allows for encoding the strategies of each agent in its decisions.

Model transformation. Let $\phi = \langle A \rangle \psi$ be a primitive TCL formula. The first observation is that a concurrent game structure $\mathcal{G} = \langle m, \mathcal{Q}, r, P, \lambda, \Sigma, \delta \rangle$ is a model of ϕ iff its unraveling $\langle \Sigma^*, l \rangle$ is, where the joint decision remains the same product $\Sigma = S_1 \times \ldots \times S_m$ of the individual decisions of the agents.

For this primitive TCL formula ϕ, we first widen the model by extending the set of decisions for each agent that occurs in an agency in the formula (in the SQ or in the agency of an SIQ $\langle +A \rangle$) by two names for each occurrence: one name for continuation, and one name for a fresh introduction. Intuitively, we use this name to refer to the selected strategy. The transitions themselves, however, are not affected by this additional information: they depend only on the old part of the decisions.

Let us refer to the set of strategy names extended by the 'new/continued' information as the set role_a of *roles* of an agent a. If an agent a does neither occur in the SQ nor in any SIQ in ψ, then role_a is a singleton. Then the new set of decisions for a is

$$S'_a = S_a \times \mathsf{role}_a.$$

For $\mathsf{role} = \mathsf{role}_1 \times \ldots \times \mathsf{role}_m$, we use

$$\Sigma' = \Sigma \times \mathsf{role} = S'_1 \times \ldots \times S'_m,$$

noting that the order is irrelevant.

Note that the additional information has no effect on the transitions. In particular, widening does not affect whether or not any TCL formula holds in this model.

Lemma 8. *$\langle \Sigma^*, l \rangle$ is a model of ψ if, and only if, $\mathsf{wide}_{\mathsf{role}}\big(\langle \Sigma^*, l \rangle\big)$ is a model of ψ.*

Proof. The model property is defined by the existence of strategies. If such strategies exist for $\langle \Sigma^*, l \rangle$, any extension of these strategies by the additional role information will do. Vice versa, if there is a strategy using the additional role information, the strategy obtained by projecting this additional information away maintains the property. ■

While the additional roles are not important for the existence of strategies, they affect how strategies can be represented. The irrelevance of the new roles for the existence is, indeed, a relevant feature: it allows for focusing on strategies with a certain property, namely strategies that identify what they go out to establish. This technique is inspired by [29], where it has been pivotal to establish the complexity of ATL*.

An SQ or SIQ, heading a subformula $\phi' = \langle ?A \rangle \psi'$ of ϕ, that introduces a strategy for agent a reasons about the existence of a strategy, which can be represented by a S_a-labeled Σ^*-tree $\langle \Sigma^*, s_a \rangle$, where s_a is the strategy $\sigma[\phi', a]$ from Section 4.3. We can now extend s_a with a role name, yielding the S'_a-labeled Σ^*-tree $\langle \Sigma^*, s_a^r \rangle$, as follows:

– $s_a^r : \varepsilon \mapsto \big(s_a(\varepsilon), (\phi', \text{new})\big)$ for the root ε and
– $s_a^r : D \mapsto \big(s_a(D), (\phi', \text{cont})\big)$ for all other nodes of the tree $\varepsilon \neq D \in \Sigma^*$.

We then expand it to the widened tree, using $\mathsf{wide}_{\mathsf{role}}\big(\langle \Sigma^*, s_a^r \rangle\big)$.

Note that we can similarly start in all positions of the (widened) tree.

This way, no agent makes the same local decision when selecting strategies that refer to different subformulas in a strategy scheme, or that refer to selections of strategies that refer to the same subformula, but to different nodes of the tree. Consequently, the strategies can be made *explicit* by encoding the *local* decisions for each role, where 'new' refers to the existential choice that needs to be made to establish that ϕ' is true. At the same time, the truth for the subformulas headed by an SQ or SIQ for the selected strategy can be made explicit. Both pieces of information can be depicted in an extended label[4]. We refer to a model with such an extended annotation as an *extended model*. Note that it is required for an extended model that the annotation is consistent: when a formula starting with an SQ or SIQ is marked as true, then the strategies encoded for it must witness this.

With the previous lemma, we get:

Lemma 9. *$\langle \Sigma^*, l \rangle$ is a model of ψ if, and only if, there exists an extended model $\langle {\Sigma'}^*, l' \rangle$ of ψ such that $\mathsf{wide}_{\mathsf{role}}\big(\langle \Sigma^*, l \rangle\big) = \mathsf{proj}_{2^P}\big(\langle {\Sigma'}^*, l' \rangle\big)$ holds.*

Note that the truth value both for the complete formula and for the subformulas headed by an SIQ is monotonous in the valuation of the subformulas that start with an SIQ. This allows us to only validate that these subformulas are true, while we do not have to validate that they are not.

Note also that the widening can be implemented on the concurrent game structure by simply increasing the set of decisions there.

Automaton transformations. The above lemma allows us to build an automaton that recognizes annotated models, and then project the annotation away. It is simple to construct a weak universal tree automaton $\mathcal{U}$ that recognizes the set of trees $\langle {\Sigma'}^*, l' \rangle$ that are extended models of ϕ, where $\mathcal{U}$ has a size polynomial in the length of ϕ.

For $\mathcal{U}$, we can then apply a number of automaton transformations.

Determinization. The weak universal tree automaton $\mathcal{U}_\phi$ can then be translated into a language equivalent deterministic Büchi tree automaton $\mathcal{D}_\phi$, using a breakpoint construction or the simulation theorem [25]. This step incurs an exponential blow-up.

Projection. In the next step, the additional annotation is projected away, resulting in a nondeterministic Büchi tree automaton $\mathcal{N}_\phi$ that recognizes the widened trees that can be annotated consistently.

Note that projection does not change the statespace of the automaton. This establishes the following lemma.

Lemma 10. *The model-checking problem of primitive TCL formulas against concurrent game structures is in EXPTIME, and in P for fixed formulas.*

Together with Lemma 6, this observation makes it easy to establish EXPTIME-completeness.

[4] This is in contrast to the original concurrent game structure and also to its unraveling, where the number of strategies a state / node can refer to is unbounded.

Theorem 1. *The model-checking problem of TCL formulas against concurrent game structures and against turn-based games is EXPTIME-complete. Both problems are P-complete for fixed formulas.*

Proof. Lemma 10 establishes that the problems are in EXPTIME and P for primitive TCL formulas. To expand this observation to full TCL, we can observe that this can be achieved by evaluating the truth of formulas of the form $\langle A\rangle\psi$ from the leaves of the formula tree to the root. The result of this model checking can be treated as a fresh atomic proposition, and it can be used instead of the respective subformula.

Hardness is inherited from Lemma 6. ■

Note that this argument shows more. For fixed formulas, TCL model checking reduces to solving a number of Büchi games. The size of the game and the number of games to be played is polynomial in $\mathcal{G}$.

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

6.3 Satisfiability checking complexity

The automaton construction from the previous subsection extends to a construction for satisfiability checking.

For satisfiability checking, it is more convenient to reduce to the original set of decisions.

Narrowing. The additional directions are then removed using the narrowing operation[5] from [18]. The resulting alternating Büchi automaton $\mathcal{A}_\phi$ runs on the original set of decisions. The size of the automaton is not effected.

Complementation. Complementation can be achieved by dualization, where the complement $\mathcal{A}_\phi^c$ of an alternating Büchi automaton $\mathcal{A}_\phi$ can be obtained from $\mathcal{A}_\phi$ by dualizing its transition function (swapping disjunctions and conjunctions as well as true and false) and decreasing the priority of all states by 1, resulting in an alternating Co-Büchi automaton that recognizes the complement of the language of $\mathcal{A}_\phi$.

Theorem 2. *For a fixed set of decisions, the TCL satisfiability problem for concurrent game structures can be solved in time doubly exponential in the length of a TCL formula ϕ.*

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 recognizes the (unraveled) enriched models of a specification for a given set of joint decisions Σ. This is quite simple: $\mathcal{A}$ merely has to check that the boolean combination of SQ formulas

[5] Narrowing is a standard operation, where all atoms $\big(q,(x,y)\big)$ are replaced by (q,x). The narrowing operation thus turns an automaton $\mathcal{B}$ that recognizes a labeled $\Sigma \times \Pi$ trees into an automaton $\mathcal{A}$ that recognize those labeled Σ trees, whose Π widenings are accepted by $\mathcal{B}$.

that forms the TCL sentence ϕ is satisfied in the root of the tree, and that the truth assignment of each SQ is consistent.

This is simple, as we can use the alternating Büchi tree automaton $\mathcal{A}_{\phi'}$ from above to validate the claim that a subformula ϕ' of ϕ that starts with an SQ is satisfied, and its dual $\mathcal{A}^c_{\phi'}$ to validate that it is not satisfied.

Hence, such an automaton has only two states more than the sum of the states of the individual $\mathcal{A}_{\phi'}$ and its dual $\mathcal{A}^c_{\phi'}$. In particular, it is exponential in ϕ.

For the resulting alternating automaton, we can again invoke the simulation theorem [25] 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 priorities are exponential in ϕ. Solving the emptiness game of this automaton reduces to solving a parity game, which can be done in time doubly exponential in ϕ, e.g., using [28]. ■

In the above proof, the size of the set of joint decisions enters through the individual alternating automata $\mathcal{A}_{\phi'}$ and $\mathcal{A}^c_{\phi'}$. Thus, in order to establish a doubly exponential complexity independent of Σ, we have to show that choosing a set Σ that is exponential in ϕ suffices. We establish this by providing a model transformation from an arbitrary model to a standard model, which is exponential in ϕ.

Preparation and goals of the model transformation. For this definition, let us assume that we have an arbitrary enriched model. For this model, we now revisit the transformations we have performed on the model for model checking in the previous subsection and refine the definition of the nondeterministic automaton.

The first observation is that we can widen the set of decisions S_a of an agent a in a similar fashion as described in the previous subsection to a set S'_a, where $S'_a = S_a \times \mathsf{role}_a$ is a product of the 'old' decisions S_a and the roles of a. Note that the set of roles role_a now refers to the full formula. (It is a union of the roles for the subformulas.) We can then unravel the model to a tree.

We now re-visit the translation from the deterministic Büchi automaton $\mathcal{D}_{\phi'}$ for a primitive TCL formula[6] and its translation to the nondeterministic Büchi automaton $\mathcal{N}_{\phi'}$. The verifier in $\mathcal{N}_{\phi'}$ has to guess two things: the strategy of the players for each role, and the set of valid subformulas of ϕ' that start with an SIQ. This selection defines which state is sent into which direction / joint decision. The falsifier then selects a joint decision.

Dualizing $\mathcal{N}_{\phi'}$ to its complement $\mathcal{C}_{\phi'}$ changes the roles of the verifier and the falsifier. That is, for $\mathcal{N}_{\phi'}$, the verifier selects the truth of SIQ subformulas and the strategy; once the selection is made, the falsifier can select the joint decision to be executed. For its dual $\mathcal{C}_{\phi'}$, the *falsifier* can select the truth of SIQ subformulas and the strategy; once the selection is made, the verifier can select the joint decision to be executed.

For the selection of the joint decision by the verifier, it is important to note that the verifier tries to show that an obligation represented by the state is not met. This raises the question, which obligation—represented by the automaton state—is sent along which path.

[6] Recall that we use an enriched model, where all TCL formulas that start with an SQ have a truth assignment and can therefore be treated as a primitive TCL formula.

For a primitive TCL formula ϕ', the obligation that the verifier tries to establish in $\mathcal{N}_{\phi'}$—or to refute in its dual $\mathcal{C}_{\phi'}$—is defined by the set of previous SIQs that have been claimed to hold and the representation of the strategies. Recall that the strategy for an agent a and a strategy $\sigma[\phi'', a]$ is only represented in terms of S_a, as the role information is canonically selected such that different strategies of a never concur.

Thus, the set of basic obligations bob of an agent a represented in a state of $\mathcal{N}_{\phi'}$ (or $\mathcal{C}_{\phi'}$) is a subset $B \in \mathsf{bob}$ of the subformulas of ϕ' that start with an SQ or SIQ. This set B refers to obligations defined by the path to this point through the decisions S_a of each agent $a \in A$ and therefore satisfy side constraints. In particular, for all such subformulas $\psi, \psi' \in B$, it holds that, for all agents $a \in \mathit{bnd}(\psi) \cap \mathit{bnd}(\psi')$ in the intersection of the bound agencies of ψ and ψ', ψ and ψ' refer to the same strategy of a ($\sigma[\psi, a] = \sigma[\psi', a]$). In addition to these obligations, states only keep the information, which untility is to be discharged next for a state to be accepting (has priority 2) for $\mathcal{N}_{\phi'}$ and rejecting (has priority 0) for $\mathcal{C}_{\phi'}$, respectively.

For a basic obligation $B \in \mathsf{bob}$, we call the agents $A = \{a \mid a \in \psi \in \mathsf{bob}\}$ bound in B and the agents $[1, m] \smallsetminus A$ free in B.

We define a set of extended obligations for each agent a, eob_a, as a triple of

– a basic obligation $B \in \mathsf{bob}$ such that a is free in B,
– the additional bookkeeping information (next utility) that defines a state together with the basic obligation, and
– a check integer in $[1, m]$,

and define a *standard decisions* $S_a = \mathsf{role}_a \cup \mathsf{eob}_a$ for each agent $a \in [1, m]$, which consists of its roles and extended set of obligations. We also define an arbitrary fallback role $r_a^0 \in \mathsf{role}_a$ for each agent.

We now define a translation from the joint standard decisions $\Sigma = S_1 \times \ldots \times S_m$ to Σ' in three steps, and then discuss how accepting strategies of the verifier translate.

In a first step, the translation takes a joint standard decision, and replaces a decision $E \in \mathsf{eob}_a$ with basic obligation $B \in \mathsf{bob}$ of an agent a by her fallback decision r_a^0 if

– there is an agent a' free in B who has not selected the same state (basic obligation and next utility) or
– the sum of the check integers from extended obligations selected by any agent does not equal to a number that identifies an agent a' free in B modulo m.

This translation ensures that there is at most one state selected, and that all agents free in the basic obligation B represented in it have selected it. Intuitively, they respond to the same choice of the remaining agents.

In a second step, we simply remove the check integer.

In a third step, we first translate the decisions $r_a \in \mathsf{role}_a$ for each agent a who has made such a decision. For this, we use a memoryless verifier strategy for $\mathcal{N}_{\phi'}$ to select the direction $(D, r_a) \in S'_a$ for the role r_a selected. We then use a memoryless falsifier for $\mathcal{N}_{\phi'}$ to select a joint decision for the obligation defined by the state. This selection is used to define the decisions for the remaining players.

This translation is then extended from transitions to define a labeled Σ-tree $\langle \Sigma^*, l \rangle$.

Translation of winning strategies. The translation describes how a play on the translation simulates a play on the original. It suffices to describe a single step of each translation. Let us start with the translation for $\mathcal{N}_{\phi'}$. The verifier selects the set of valid subformulas and the strategy, and the falsifier a direction. For convenience, we split the choice of the falsifier into first choosing the state to send, and then the direction.

We now describe how a step of the winning strategies of the verifier and falsifier, respectively, in the original are translated. It is important to note that a memoryless strategy for the verifier and a memoryless strategy for the falsifier have been used in the translation of the concurrent game graph, where one of these strategies is winning. This fact is the central ingredient in the argument.

We now look at the three individual sub-steps each step in the acceptance game consists of.

1. The verifier chooses his strategy.
2. The falsifier chooses the state.
3. The falsifier chooses a direction this state is sent to.

Note that (1) and (3) are hard coded into the agents' decisions in the translated game. The verifier strategy for making a decision of agent a is one of its roles: a name of one of the strategies of a plus a flag that shows if it is new or continued. The translation of this strategy is then simply the assignment of this strategy from the memoryless strategy used in the translation of the concurrent game graph.

If the falsifier wins the original, she can choose the same state in the translation in sub-step (2). The state then defines which of the agents are free in this state, and the falsifier chooses an extended obligation (with correct checksum) for all of these agents. In the translation, this state is then sent into the direction, where all agents that are bound in the basic obligation choose the according basic obligation, and all agents that are free choose the according extended basic obligation. This translates back to the same direction, the falsifier would have selected in the original.

The case where the verifier wins is even simpler. Whichever state is selected by the falsifier, it refers to a basic obligation. It was sent by the verifier in all directions, where the agents bound in this basic obligation have made the according choice (extended strategy name), which is translated back to the same choice in the original game.

When the falsifier then selects a direction in the translation, it can only fix the decisions of the agents free in this basic obligation: the decisions of the agents bound in the basic obligation are determined by the basic obligation selected. Thus, the falsifier has effectively a restricted choice, as she can only select directions in the image of the translation, which might not cover all decisions of agents that are free in the basic obligation. Naturally, this cannot alter the fact that she loses the game.

For $\mathcal{C}_{\phi'}$, the roles of verifier and falsifier are changed, but the plays defined by them are the same.

Theorem 3. *The TCL satisfiability problem for concurrent games can be decided in 2EXPTIME time.*

Note that the case of turn-based games is simpler: if we mark which player owns a state, then all claims translate to "for some successor" or 'for all successors'. In particular, inclusion in 2EXPTIME is maintained.

Noting that hardness is inherited (both for turn-based and concurrent games) from the embedding of CTL* and the 2EXPTIME-completeness of CTL* satisfiability checking [33], we get completeness results for TCL.

Theorem 4. *The TCL satisfiability problem is 2EXPTIME-complete for turn-based games and for concurrent game structures.*

7 Implementation and experimental results

As a proof of concept, we have implemented a model-checker, `tcl`, in C++. Our tool `tcl` accepts models composed of extended automata that communicate with synchronizers and shared variables. A 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. However, the main purpose of this experiment report is to demonstrate the feasibility of the ideas and techniques discussed in this article. The implementation is very basic. Research on further techniques to enhance the performance of the implementation and a comparison with related tools are left for future research.

7.1 Implementation

The implementation builds on a prototype for a PSPACE logic [35]. 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 11. *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 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 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 excempt 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 11. The tool (including source code) and experiment materials can be downloaded at the following site:

`https://github.com/yyergg/TCL`

7.2 Experiments

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 [35].

- Property (F) specifies that all prisoners except Prisoner 1 can collaborate to release Prisoner 1 and let Prisoner 1 decide their fate.

$$\langle 2,\ldots,m\rangle\big((\langle +\rangle\Diamond\neg\texttt{jail}_1) \wedge \textstyle\bigwedge_{i\in\{2,\ldots m\}}(\langle +1\rangle\Diamond\neg\texttt{jail}_i) \wedge (\langle +1\rangle\Box\texttt{jail}_i\big) \qquad \text{(F)}$$

- Property (G) specifies that Prisoner 1 has a strategy to put all other prisoners in jail while leaving her fate to them.

$$\langle 1\rangle\big((\textstyle\bigwedge_{i\in\{2,\ldots m\}}\langle +\rangle\Box\texttt{jail}_i) \wedge (\langle 2,\ldots,m\rangle\Diamond\neg\texttt{jail}_1) \wedge \langle 2,\ldots,m\rangle\Box\texttt{jail}_1\big) \qquad \text{(G)}$$

For these benchmarks, we have collected the performance data for various parameter values in Table 1. The experiment was carried out on an Intel i5 2.4G notebook with 2 cores and 4G memory running ubuntu Linux version 11.10.

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

properties \ m	2	3	4	5	6	7	8	9	10	11
(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	>1000s
	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	158s
	163M	163M	164M	165M	168M	176M	200M	241M	318M	480M
(D)	0.5s	0.51s	0.57s	0.74s	1.01s	1.79s	7.41s	33.8s	141s	>1000s
	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	177s
	163M	163M	163M	165M	168M	175M	202M	243M	295M	629M
(G)	0.52s	0.52s	0.65s	0.72s	1.03s	1.85s	4.86s	16.1s	93.5s	>1000s
	163M	163M	164M	165M	169M	177M	189M	208M	235M	

s: seconds; M: megabytes.
The models are with 1 policeman and m prisoners.

As discussed in Subsection 2.3, our tool is the first one that synthesizes memoryful strategies for multi-agent systems. Its purpose is to demonstrate that TCL model checking is feasible. Thus it is not really meaningful to compare with other tools [3, 1, 9, 31, 32], which would necessary compare those tools on the intersection of their languages, e.g., on ATL for MOCHA [3] and ATL Designer [31, 32].

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.

8 Conclusion

TCL results from an investigation of practical approaches for applying game theory to the specification and control of groups of agents who balance their strategies in order to cooperate with different partners to achieve different objectives. Thus, we have carefully considered various aspects of specification expressiveness and synthesis efficiency in the design of TCL. One the one hand, we can argue for the expressiveness of TCL with several examples, including the classical problem of iterated prisoners' dilemma [4]. Especially, several properties in [4] can readily be specified in TCL. We also rigorously proved that TCL subsumes CTL* and LTL while incomparable with ATL*, GL, and AMC in expressiveness. We feel that ATL at least has offered a new perspective in designing specification languages that enable the application of game theoretical techniques to real-world projects. Whether the expressiveness offered by TCL is indeed valuable to real-world projects is still an interesting and open problem that deserves further investigation.

On the other hand, TCL 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. As argued in the above, 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.

Finally, our experiment report seems encouraging 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. Our implementation certainly can be used as a basis for further investigation of performance techniques in the implementation by our colleagues in the community. It will be interesting to see by which extent symbolic representation like BDDs will enhance the performance and how an automaton-based tool would fare.

Acknowledgment

We would like to thank the reviewers for their constructive an helpful comments, which greatly helped to improve the article. The work is supported by the following:

- Engineering and Physical Science Research Council through the grants EP/H046623/1 ‘Synthesis and Verification in Markov Game Structures’ and EP/M027287/1 ‘Energy Efficient Control’;
- Ministry of Science and Technology, Taiwan, ROC through Grant MOST 103-2221-E-002-150-MY3, Taiwan, ROC; and
- Research Center for Information Technology Innovation, Academia Sinica, Taiwan, ROC.

References

1. N. Alechina, B. Logan, H. N. Nguyen, and F. Raimondi. Symbolic model checking for one-resource rbÂśatl. In *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence (IJCAI)*, 2015.
2. R. Alur, T. A. Henzinger, and O. Kupferman. Alternating-time temporal logic. *Journal of the ACM (JACM)*, 49(5):672–713, September 2002.
3. R. Alur, T. A. Henzinger, F. Mang, S. Qadeer, S. K. Rajamani, and S. Tasiran. Mocha: Modularity in model checking. In *Proceedings of the Tenth International Conference on Computer-aided Verification (CAV 1998)*, volume Lecture Notes in Computer Science (LNCS) 1427, pages 521–525. Springer-Verlag, 1998.
4. R. Axelrod. Effective choice in the prisoner’s dilemma. *Journal of conflict resolution*, 24(1):3–25, 1980.

5. C. Baier, T. Brázdil, M. Gröser, and A. Kucera. Stochastic game logic. In *QEST*, pages 227–236. IEEE Computer Society, 2007.
6. T. Brihaye, A. D. C. Lopes, F. Laroussinie, and N. Markey. ATL with strategy contexts and bounded memory. In *LFCS*, volume LNCS 5407, pages 92–106. Springer-Verlag, 2009.
7. J. Büchi and L. Landweber. Definability in the monadic second-order theory of successor. *Journal of Symbolic Logic*, 34(2):166–170, 1969.
8. J. Büchi and L. Landweber. Solving sequential conditions by finite-state strategies. *Trans. AMS*, 138(4):295–311, 1969.
9. P. Cermak, A. Lomuscio, F. Mogavero, and A. Murano. Mcmas-slk: A model checker for the veri
cation of strategy logic speci
cations. In *International Conference on Computer-Aided Verification (CAV)*, volume LNCS 8559. Springer-Verlag, 2014.
10. K. Chatterjee and T. A. Henzinger. A survey of stochastic ω-regular games. *Journal of Computer and System Sciences*, 78(2):394âĂŞ413, March 2012.
11. K. Chatterjee, T. A. Henzinger, and N. Piterman. Strategy logic. *Information and Computation*, 208:677–693, 2010.
12. 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.
13. E. Emerson and C. Jutla. Tree automata, mu-calculus and determinacy. In *32nd IEEE FOCS*, pages 368–377, 1991.
14. B. Finkbeiner and S. Schewe. Coordination logic. In *CSL*, pages 305–319, 2010.
15. G. J. Holzmann. The model checker SPIN. *IEEE Trans. Software Eng.*, 23(5), 1997.
16. N. Immerman. Number of quantifiers is better than number of tape cells. *Journal of Computer and System Sciences*, 22(3):65–72, 1981.
17. Y. Kesten and A. Pnueli. Complete proof system for QPTL. *Journal of Logic and Computation*, 12(5):701–745, 2002.
18. O. Kupferman and M. Y. Vardi. Synthesis with incomplete information. In *In Advances in Temporal Logic*, pages 109–127. Kluwer Academic Publishers, 2000.
19. 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.
20. F. Laroussinie and N. Markey. Satisfiability of ATL with strategy contexts. In *Workshop on Games, Automata, Logics and Formal Verification (GANDALF)*, volume EPTCS 119, pages 208–223, 2013.
21. A. D. C. Lopes, 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.
22. 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.
23. F. Mogavero, A. Murano, and L. Sauro. On the boundary of behavioral strategies. In *ACM/IEEE LICS*, 2013.
24. 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.

25. 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.
26. S. Pinchinat. A generic constructive solution for concurrent games with expressive constraints on strategies. In *Automated Technology for Verification and Analysis (ATVA)*, volume LNCS 4762, pages 253–267. Springer-Verlag, 2007.
27. A. Pnueli. The temporal logic of programs. In *18th annual IEEE-CS Symposium on Foundations of Computer Science*, pages 45–57, 1977.
28. 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.
29. S. Schewe. ATL* satisfiability is 2ExpTime-complete. In *Proceedings of the 35th International Colloquium on Automata, Languages and Programming, Part II (ICALP 2008), 6–13 July, Reykjavik, Iceland*, volume 5126 of *Lecture Notes in Computer Science*, pages 373–385. Springer-Verlag, 2008.
30. L. J. Stockmeyer and A. K. Chandra. Provably difficult combinatorial games. *SIAM Journal on Computing (SICOMP)*, 8(2):151–174, 1979.
31. F. Stoica and L. F. Stoica. Implementing an atl model checker tool using relational algebra concepts. In *22nd International Conference on Software, Telecommunications and Computer Networks (SoftCOM)*, pages 361 – 366. IEEE, 2014.
32. L. Stoica, F. Stoica, and F. Boian. Verification of jade agents using atl model checking. *INTERNATIONAL JOURNAL OF COMPUTERS COMMUNICATIONS & CONTROL*, 10(5):718–731, October 2015.
33. M. Y. Vardi and L. J. Stockmeyer. Improved upper and lower bounds for modal logics of programs: Preliminary report. In R. Sedgewick, editor, *Proceedings of the 17th Annual ACM Symposium on Theory of Computing (STOC)*, pages 240–251, May 6-8, 1985.
34. D. Walther, W. van der Hoek, and M. Wooldridge. Alternating-time temporal logic with explicit strategies. In *Proceedings of the 11th Conference on Theoretical Aspects of Rationality and Knowledge (TARK)*, pages 269–278, 2007.
35. F. Wang, S. Schewe, and C.-H. Huang. An extension of ATL with strategy interaction. *ACM Transactions on Programming Language and Systems (TOPLAS)*, 37(3):9, June 2015. A preliminary version is in the proceedings of CONCUR 2011, LNCS 6901, Springer-Verlag.
36. 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 6

This section contains the details of the reduction to PEEK-G_6 from the proof of Lemma 6. Note that, while PEEK-G_6 allows the agents to pass, we disllow 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 5 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 dualizing 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.