Incremental hyperproperty model checking via games

Dimiter Milushev and Dave Clarke

iMinds-DistriNet, KU Leuven, Heverlee, Belgium

Abstract. Hyperproperties were proposed as an abstract formalization of security policies, but unfortunately they lack a generic verification methodology. In an attempt to remedy this, we introduced the notion of *incremental hyperproperties* (IHPs), motivated by the observation that they have a clearer and more feasible verification methodology. To show that verification is indeed feasible, a decidable IHP verification methodology via games is presented and evaluated. The main advantage of the approach is that the games in combination with winning strategy evidence give valuable intuition about the security of a system and are very helpful when analyzing systems w.r.t. policy specifications.

1 Introduction

Clarkson and Schneider introduced the notion of *hyperproperties* [3] in an attempt to formalize security policies. A *hyperproperty* is a set of sets of execution traces over some alphabet. Hyperproperties are important and intuitively appealing as they generalize properties and can be seen as very generic system specifications. Some prominent instances of security-relevant hyperproperties are the large variety of notions of noninterference [12,21,10].

Unfortunately, hyperproperties lack a generic verification methodology: for instance, there is no such verification methodology for possibilistic information flow hyperproperties [3]. In order to make a step towards such a methodology, in recent work [15] we proposed an incremental approach to both system and hyperproperty specification and verification. As a result, systems can be seen as potentially infinite trees and hyperproperties as coinductive predicates on k-tuples of trees expressed in a logic called $\mathcal{IL}$. Specifications defined in such a manner are called *incremental hyperproperties* (IHPs) and we argued that they have a clear and feasible verification methodology [15]. Given a hyperproperty H, an IHP H' is the greatest fixed point of a monotone function over k-tuples of trees such that H' implies or is equivalent to H. We also introduced H'-*simulation relations* which correspond to a monotone operator whose greatest fixed point is the coinductive tree predicate H'. Showing the existence of such a relation is sufficient to show that H' (and thus H) holds [15].

In order to show that IHPs can express a large class of useful, security-relevant hyperproperties, we demonstrated that our coinductive unwinding relations (which happen to be H'-simulations and thus IHPs) can express and reason

about arbitrary security-relevant hyperproperties, including but not limited to noninference [21], generalized noninference [21], generalized noninterference [12] and the perfect security property [21] (shown in recent work [14]).

However, it turns out that the initially proposed logic $\mathcal{IL}$ [15] for IHPs is undecidable and not expressive enough for a large class of useful IHPs arising in practice, such as the incremental variants of possibilistic information flow policies explored in recent work [14]. To address these problems, we investigated several related logics [13] for IHPs and found out that the most appropriate one (for the known IHPs) is a fragment $\mathcal{IL}^k_\mu$ of the polyadic mu-calculus $\mathcal{L}^k_\mu$ [1].

In this paper, we start by proposing a characterization of the satisfaction relation between a system and an IHP in $\mathcal{L}^k_\mu$ in terms of playing the so called IHP *game*. Such games are intimately related to parity games and this is used to enable model checking IHPs based on game-based, off-the-shelf tools. In particular, we explore the problem of IHP model checking via games, by proposing two sound, game-based approaches (one of them based on off-the-shelf tools). In addition, we show that the games in combination with winning strategy evidence give valuable intuition about the security of a system and can be very helpful when analyzing why a system fails to respect some policy specification. We show that using such techniques and visualizations in terms of games can potentially result in tools with more intuitive debugging functionality.

The rest of the paper is structured as follows. Section 2 provides background material. Section 3 proposes the logic $\mathcal{IL}^k_\mu$ which is a fragment of $\mathcal{L}^k_\mu$. It also introduces IHP *games* for $\mathcal{L}^k_\mu$ and establishes their relation to parity games. Section 4 presents possible model checking approaches and an empirical evaluation of one of them. Section 5 discusses the advantages of model checking IHPs via games. In Sections 6 we discuss the main contributions and compare them with related work. Finally, we conclude and share some ideas for future work.

2 Background

Let A be a fixed alphabet of abstract observations. A *string* is a finite sequence of elements of A. The set of all strings over A is denoted A^*. A *stream* of A's is an infinite sequence of elements of A. The set of all streams over A is A^ω. A stream σ can be specified in terms of its first element $\sigma(0)$ and its stream derivative σ', given by $\sigma'(n) = \sigma(n+1)$. A *trace* is a finite or infinite sequence of elements of A. The set of all traces over A is denoted $A^\infty = A^* \cup A^\omega$. Let 2 be any two element set. A *system* is a set of traces. The set of all systems is $\mathsf{Sys} = 2^{A^\infty}$, the set of infinite systems is $\mathsf{Sys}_\omega = 2^{A^\omega}$.

2.1 Properties vs. hyperproperties

Clarkson and Schneider present a theory of policies based on properties and hyperproperties [3]. Our definitions are slight generalizations of the original ones, as we do not require all traces to be infinite. As a result, termination-sensitive definitions can be expressed more naturally. A *property* is a set of traces. The set of all properties is $\mathsf{Prop} = 2^{A^\infty}$. A *hyperproperty* is a set of sets of traces or a set of properties. The set of all hyperproperties is $\mathsf{HP} = 2^{2^{A^\infty}} = 2^{\mathsf{Prop}} = 2^{\mathsf{Sys}}$.

2.2 Models of systems

In this work, we model systems as *partial automata* [16], trees or sets of traces, as these can be seen as equivalent views [15]. A *partial automaton* with input alphabet A and a start state is a 4-tuple $\langle S, o, t, s_0 \rangle$, where set S is the possibly infinite state space of the automaton, the observation function $o : S \to 2$ indicates whether a state is accepting or not, the function $t : S \to (1 + S)^A$ gives the transition structure and s_0 is the initial state. If $t(s)$ is defined for some $a \in A$, then $t(s)(a) = s'$ gives the next state; s' is then called an *a-derivative* of s and denoted s_a. When the function $t(s)$ is undefined for some $a \in A$, it is mapped to $\bot$. The observation function indicates whether the empty trace is in the set of traces acceptable by the partial automaton from state s. Note that $t(s)(a) = s'$ is typically abbreviated as $s \xrightarrow{a} s'$ and $t(s)(a) = \bot$ as $s \not\xrightarrow{a}$.

Alternatively, it is often more intuitive and convenient to think of a system as being equivalent to its behavior. In such cases, we talk about the unique tree of system behavior. A *tree* can be obtained from a partial automaton by continuously taking derivatives with respect to elements of A. The start state of the system corresponds to the root of the tree and *subtrees* are obtained by taking derivatives. Yet another view of systems is as sets of traces accepted by a partial automaton. As partial automata, trees and sets of traces are equivalent views on systems, we often implicitly switch between these views: for instance, we may write $t(T)(a)$ and T_a, where the type of T is either tree or system. Finally, for a k-tuple of trees $\overline{T}$ we use notation $\overline{T} \xrightarrow{a}_i \overline{T'}$ to mean that $\overline{T'}_i = t(\overline{T}_i)(a)$, where $\overline{T}_i$ is the i-th tree in $\overline{T}$ and for all j s.t. $1 \leq j \leq k$ and $j \neq i$, $\overline{T'}_j = \overline{T}_j$.

2.3 Auxiliary definitions

For a partition of alphabet A as $A = A_v \cup A_n \cup A_c$, define a *view* to be a tuple (A_v, A_n, A_c) corresponding to *visible*, *neutral* and *confidential* events [10]. Let sets A_i and A_o be inputs and outputs such that $A_i \subseteq A$, $A_o \subseteq A$ and $A_i \cap A_o = \emptyset$.

We also introduce definitions from previous work [14]. Coinductively define $no_Z : A^\infty \to 2$, which states that there are no events from set Z in a trace as:

$$\frac{}{no_Z(\epsilon)}\ coind \qquad \frac{a \in A \setminus Z \qquad no_Z(x)}{no_Z(a \cdot x)}\ coind$$

Next, inductively define $w \rightsquigarrow_Z a \cdot w'$ (for $Z \subseteq A$, w Z-reveals a with tail w')

$$\frac{}{\epsilon \rightsquigarrow_Z \epsilon} \qquad \frac{a \in Z}{a \cdot w \rightsquigarrow_Z a \cdot w} \qquad \frac{b \in A \setminus Z \qquad w \rightsquigarrow_Z a \cdot w'}{b \cdot w \rightsquigarrow_Z a \cdot w'}$$

Finally, coinductively define *weak bisimulation* w.r.t. set Z as follows:

$$\frac{}{\epsilon \sim_Z \epsilon}\ coind \qquad \frac{w \rightsquigarrow_Z a \cdot w' \qquad u \rightsquigarrow_Z a \cdot u' \qquad w' \sim_Z u'}{w \sim_Z u}\ coind$$

The coinductive definitions are denoted as *coind* on the right side of the rule.

2.4 Incremental hyperproperties as coinductive predicates [14]

In recent work [15] we introduced and formalized the notion of *incremental hyperproperties* (IHPs). Such a hyperproperty is the greatest fixed point of a monotone functional over Sys^k, given in a fragment of Least Fixed Point Logic (the extension of first order logic with fixed point operators) [2], denoted $\mathcal{IL}$.

An *incremental hyperproperty* (IHP) is a coinductive predicate on a k-tuple of trees that can be specified by some formula $\phi \in \mathcal{IL}$. The set of all incremental hyperproperties is $\{\overline{S} \subseteq \mathsf{Sys}^k \mid \overline{S} \models \phi \text{ where } \phi \in \mathcal{IL}\}$. Coupled with an IHP H', we introduced the notion of an H'*-simulation* — an n-ary relation R such that $R \subseteq \Psi_{H'}(R)$, where $\Psi_{H'}$ is a monotone operator determined by H'. To illustrate these notions, consider a variant of noninterference (called noninference [14]):

$$NI(X) \;\widehat{=}\; \forall x_0 \in X\; \exists x_1 \in X.\; (no_{A_c}(x_1) \wedge x_1 \sim_{A_v} x_0).$$

The corresponding notion of NI' is given as follows:

$$\begin{aligned} NI' \;\widehat{=}\; \mathbf{gfp}\, R(s,t)\;.\; \forall a \in A \setminus A_c\; \forall s_a \in \mathsf{Sys}.\; \Big(s \xrightarrow{a} s_a \rightarrow \\ \exists \sigma \in (A \setminus A_c)^*\; \exists t_\sigma \in \mathsf{Sys}.(t \xrightarrow{\sigma} t_\sigma \wedge a \sim_{A_v} \sigma \wedge R(s_a, t_\sigma))\Big) \wedge \\ \forall a \in A_c\; \forall s_a \in \mathsf{Sys}\;.\; (s \xrightarrow{a} s_a \rightarrow R(s_a, t)). \end{aligned}$$

It is known [14] that for all $T \in \mathsf{Sys}$, $NI'(T,T)$ implies $NI(T)$. As a result, to show that $NI(T)$, we can alternatively reason about $NI'(T,T)$.

2.5 The polyadic modal mu-calculus interpreted over trees [13]

The polyadic modal mu-calculus $\mathcal{L}_\mu^k$ [1] is a logic whose formulae are interpreted over k-tuples of transition systems. It is an extension of the modal mu-calculus [2] with different diamond and box modalities associated with each system (from the k-tuple). In this work, formulae will be interpreted over k-tuples of trees, denoted $\overline{\mathcal{T}}$. The elements of these tuples will be referred to as $\mathcal{T}_i$, where $1 \leq i \leq k$.

Assume a set $Var_2 = \{X, Y, Z, \ldots\}$ of second-order variables and a set $P = \{Q_i, O_i, \ldots : 1 \leq i \leq k\}$ of propositional constants. Formulae in $\mathcal{L}_\mu^k$ have the following syntax:

$$\Phi ::= tt \mid ff \mid Z \mid \Phi_1 \wedge \Phi_2 \mid \Phi_1 \vee \Phi_2 \mid [a]_i\Phi \mid \langle a\rangle_i\Phi \mid \nu Z.\Phi \mid \mu Z.\Phi,$$

where tt and ff are the constant true and false formulae, $a \in A$, $[a]_i$ and $\langle a\rangle_i$ are the typical modal operators relativized to the i-th tree, where $1 \leq i \leq k$. As usual, μZ and νZ are the least and greatest fixed point operators, respectively. Sometimes, for $K \subseteq A$ we abbreviate $\bigwedge_{a\in K}[a]_i\Phi$ as $[K]_i\Phi$ and $\bigvee_{a\in K}\langle a\rangle_i\Phi$ as $\langle K\rangle_i\Phi$. Finally, propositional variables are ranged over by second-order variables from Var_2. The semantics of $\mathcal{L}_\mu^k$ on trees is given in recent work [13].

Any hyperproperty expressed in $\mathcal{L}_\mu^k$ can be checked in polynomial time [1]. There is an algorithm for deciding $\overline{\mathcal{T}} \models \Phi$, where Φ is closed, $\overline{\mathcal{T}}$ a k-tuple of finite transition systems with state spaces $S_1, \ldots, S_k$ and m is the alternating depth of Φ, in time $O(|\Phi|^m(|S_1|\ldots|S_k|)^{m-1}|\mathcal{T}_1|\ldots|\mathcal{T}_k|)$. Here $|\mathcal{T}_i|$ is the size of

the underlying state space plus the size of the transition relation plus 1 and $|S_i|$ is the size of the respective state space. The result is applicable to our setting for reasoning about potentially infinite trees, generated by finite-state partial automata.

3 Incremental hyperproperty checking games

This section starts by presenting a fragment of $\mathcal{L}_\mu^k$ which is expressive enough for the known IHPs. Then it shows how to interpret IHP checking as playing a game (called an IHP *game* and related to parity games) and thus lays the foundations for game-based verification of IHPs.

3.1 A new logic for incremental hyperproperties

The logic $\mathcal{IL}_\mu^k$, which is a fragment of $\mathcal{L}_\mu^k$, is expressive enough for all IHPs encountered in our former work [15,14]. Formulae in $\mathcal{IL}_\mu^k$ have syntax:

$$\Psi ::= \nu Z.\Phi \qquad \Phi ::= tt \mid ff \mid Z \mid \Phi_1 \wedge \Phi_2 \mid \Phi_1 \vee \Phi_2 \mid [a]_i\Phi \mid \langle a\rangle_i\Phi \mid \mu Z.\Phi.$$

The maximal alternating depth of any formula in $\mathcal{IL}_\mu^k$ is 2, which results in lower model checking complexity compared with $\mathcal{L}_\mu^k$. This is reflected in a result about the complexity of model checking $\mathcal{IL}_\mu^k$: there is an algorithm running in time $O(|\Phi|^2|S_1|\dots|S_k||\mathcal{T}_1|\dots|\mathcal{T}_k|)$ for deciding $\overline{\mathcal{T}} \models \Phi$, where Φ is closed and $\overline{\mathcal{T}}$ a k-tuple of finite transition systems with state spaces $S_1, \dots, S_k$ [13]. The logic allows (a restricted form of) coinductive/inductive definitions reminiscent of the idea that any hyperproperty is the intersection of hypersafety and hyperliveness [3]: the latter are generalizations of safety and liveness properties.

To illustrate the need of alternation of least and greatest fixed point operators (and to give intuition why $\mathcal{IL}$ is insufficient), consider the coinductive unwinding relation osc_V [14] in $\mathcal{IL}_\mu^k$. Intuitively, osc_V gives the indistinguishability of possible behaviors at level A_v, where $O_1 \leftrightarrow O_2$ means that the related states have the same observations (both accepting or both rejecting, see Section 2.2).

$$osc_V \;\widehat{=}\; \nu X.\, O_1 \leftrightarrow O_2 \wedge \bigwedge_{a \in A \setminus A_c} [a]_1 \mu Z.(\langle a\rangle_2 X \vee \langle A_n\rangle_2 Z).$$

Although $\mathcal{IL}_\mu^k$ is expressive enough for the known IHPs, the theory presented in this paper is more general as it works for the full polyadic modal mu-calculus.

3.2 Incremental hyperproperty checking games (IHP games)

In this section we propose a game-theoretic characterization of when an IHP expressed in $\mathcal{L}_\mu^k$ holds for a k-tuple of trees $\overline{\mathcal{T}}$, relative to a second-order valuation V. The IHP games presented next are played by two players: refuter (R) and verifier (V). R attempts to disprove that $\overline{\mathcal{T}}$ satisfies an IHP H', whereas V attempts to prove that H' holds for $\overline{\mathcal{T}}$. A system satisfies an IHP whenever player V has a winning strategy for the respective IHP game.

Assume Φ expresses an IHP. A *play* of the IHP game $HG_{\mathsf{V}}((T^1,\ldots,T^k),\Phi)$ is a finite or infinite sequence of pairs of k-tuples of trees and $\mathcal{L}^k_\mu$ formulae:

$$((T^1_0,\ldots,T^m_0,\ldots,T^k_0),\Phi_0)\ldots((T^1_i,\ldots,T^m_j,\ldots,T^k_l),\Phi_n)\ldots.$$

Note that each formula Φ_i is a subformula of Φ_0 and each tree T^j_i is a subtree of T^j_0. The next move in a play from any position $((T^1_i,\ldots,T^m_j,\ldots,T^k_l),\Phi_n)$ depends on the main connective in Φ_n. The possible moves are given next:

- If $\Phi_n=\Psi_1\wedge\Psi_2$, then R choses one of the conjuncts Ψ_i ($i\in\{1,2\}$), the k-tuple of trees $(T^0_i,\ldots,T^m_j,\ldots,T^k_l)$ remains unchanged and formula $\Phi_{n+1}=\Psi_i$.
- If $\Phi_n=\Psi_1\vee\Psi_2$, then V choses one of the disjuncts Ψ_i ($i\in\{1,2\}$), the k-tuple of trees $(T^0_i,\ldots,T^m_j,\ldots,T^k_l)$ remains unchanged and formula $\Phi_{n+1}=\Psi_i$.
- If $\Phi_n=[a]_m\Psi$, then R has to move along the transition $T^m_j\xrightarrow{a}T^m_{j+1}$, the k-tuple of trees $(T^0_i,\ldots,T^m_j,\ldots,T^k_l)$ becomes $(T^0_i,\ldots,T^m_{j+1},\ldots,T^k_l)$ and formula $\Phi_{n+1}=\Psi$.
- If $\Phi_n=\langle a\rangle_m\Psi$, then V has to move along the transition $T^m_j\xrightarrow{a}T^m_{j+1}$, the k-tuple of trees $(T^0_i,\ldots,T^m_j,\ldots,T^k_l)$ becomes $(T^0_i,\ldots,T^m_{j+1},\ldots,T^k_l)$ and formula $\Phi_{n+1}=\Psi$.
- If $\Phi_n=\sigma Z.\Psi$, then formula Φ_{n+1} becomes Z and the k-tuple of trees $(T^0_i,\ldots,T^m_j,\ldots,T^k_l)$ remains unchanged.
- If $\Phi_n=Z$ and the subformula of Φ_0 identified by Z is $\sigma Z.\Psi$, then formula $\Phi_{n+1}=\Psi$ and the k-tuple of trees $(T^0_i,\ldots,T^m_j,\ldots,T^k_l)$ remains unchanged.

The winning conditions are considered next. Player R wins a finite play if a false configuration is reached: the evaluated formula Φ_n is ff, or position $((T^1_i,\ldots,T^m_{j+1},\ldots,T^k_l),Z)$ is reached where Z is free in Φ_0 and the k-tuple $(T^1_i,\ldots,T^m_{j+1},\ldots,T^k_l)\notin\mathsf{V}(Z)$, or V has to move, but such a move is impossible. The rules for V are dual. The winner in an infinite play depends on the outermost fixed point subformula that is unfolded infinitely often: if it is a least fixed point one, R wins; dually, if this is a greatest fixed point one, V wins.

The proposed approach of reasoning about games to determine if a system satisfies an IHP specification is justified by the following theorem.

Theorem 1 (Correctness of IHP games [13]). *The following equivalences are valid:*

1. $(T^1,\ldots,T^m,\ldots,T^k)\models_{\mathsf{V}}\Phi$ *iff player* V *has a history-free winning strategy for* $HG_{\mathsf{V}}((T^1,\ldots,T^m,\ldots,T^k),\Phi)$.
2. *Dually,* $(T^1,\ldots,T^m,\ldots,T^k)\not\models_{\mathsf{V}}\Phi$ *iff player* R *has a history-free winning strategy for* $HG_{\mathsf{V}}((T^1,\ldots,T^m,\ldots,T^k),\Phi)$.

3.3 From IHP games to parity games

IHP games can be converted into equivalent parity games over k-tuples of trees. This means that player $P\in\{V,R\}$ has a history-free winning strategy for an IHP game iff P has a history-free winning strategy for the respective parity game. This is not surprising, as it is known that the solving of a parity game has equivalent complexity to the model checking problem for the modal mu-calculus [2]: the result can be lifted to our polyadic setting. The method for converting IHP games into parity games is similar to the one used by Stirling [18] for converting property checking games into (min-) parity games and can be found in the first author's PhD thesis [13]. The conversion allows the use of results and tools developed for solving parity games to model check IHPs.

4 Model checking the polyadic modal mu-calculus $\mathcal{L}_{\mu}^{k}$

This section starts by presenting the use of traditional model checking techniques for IHPs. Then we introduce two novel, game-based approaches for model checking IHPs. The major advantage of model checking via games is that it gives a very accurate and intuitive account of whether a system respects a specification.

4.1 Traditional Model Checking of $\mathcal{L}_{\mu}^{k}$

It is possible to use traditional model checking techniques for IHPs in $\mathcal{L}_{\mu}^{k}$. Andersen himself proposed a model checking approach for his $\mathcal{L}_{\mu}^{k}$ [1]. The approach is a reduction of the problem of model checking $\mathcal{L}_{\mu}^{k}$ to model checking the ordinary modal mu-calculus $\mathcal{L}_{\mu}$ on a product of the original system.

Given an n-ary tuple of transition systems $(T_1, \ldots, T_n)$, define the product $prod(\overline{T})$ of these to be the labelled transition system $(S, \rightarrow, i)$, where S is the state space given as $S \mathrel{\widehat{=}} S_1 \times \ldots \times S_n$, $\rightarrow$ is the transition relation and i the tuple of the start states. Relation $\rightarrow \subseteq S \times (A \times \mathbb{N}) \times S$ is defined as follows:

$$(s_1, \ldots, s_n) \xrightarrow{a,i} (s'_1, \ldots, s'_n) \text{ iff } s_i \xrightarrow{a} s'_i \text{ and } \forall j.(1 \leq j \leq n \wedge j \neq i) \rightarrow s_j = s'_j.$$

Next, define $prod(\Phi)$ as the homomorphic map on formulae in $\mathcal{L}_{\mu}^{k}$ such that $prod(\langle a \rangle_i \Phi) = \langle (a, i) \rangle prod(\Phi)$. Note that instead of $\langle (a, i) \rangle \Phi$ we typically write $\langle a_i \rangle \Phi$. It is clear that $prod(\overline{T})$ is a single system (vs. tuple of systems) and $prod(\Phi)$ is defined over such systems.

Theorem 2 (Reduction of $\mathcal{L}_{\mu}^{k}$ to $\mathcal{L}_{\mu}$ [1]). *Consider an n-tuple of transition systems $\overline{T}$ and an $\mathcal{L}_{\mu}^{k}$ formula Φ, as well as the respective $prod(\overline{T})$ and $prod(\Phi)$ as defined above. Then the following equivalence is valid:*

$$\overline{T} \models \Phi \text{ iff } prod(\overline{T}) \models prod(\Phi).$$

This result is important, as it suggests the use of standard model checking techniques for verification of IHPs expressed in the polyadic modal mu-calculus $\mathcal{L}_{\mu}^{k}$.

We next propose two model checking approaches for IHPs via games. The first is based on the combination of IHP games and the parity game solver PGSolver [5]. The second is based on the use of several tools (including MLSolver [6], a tool to reason about satisfiability and validity of modal fixed point logics) and has the advantage that it can be fully automated. Both approaches are based on creating and solving the appropriate parity game, eventually using PGSolver [5].

4.2 Model Checking IHP Games

Start with some IHP game $HG_{\mathsf{V}}((T^1, \ldots, T^m, \ldots, T^k), \Phi)$.

1. Convert $HG_{\mathsf{V}}((T^1, \ldots, T^m, \ldots, T^k), \Phi)$ into the equivalent min-parity game $PG_{\mathsf{V}}((T^1, \ldots, T^k), \Phi)$ (See Section 3.3 and our recent work [13]).
2. Use PGSolver to convert $PG_{\mathsf{V}}((T^1, \ldots, T^k), \Phi)$ to its equivalent max-parity game $PG_{\mathsf{V}}^{max}((T^1, \ldots, T^k), \Phi)$, as PGSolver solves max-parity games.

3. Use PGSolver to solve the parity game $PG^{max}_{\mathsf{V}}((T^1,\ldots,T^k),\Phi)$.

To know if $(T^1,\ldots,T^m,\ldots,T^k) \models_{\mathsf{V}} \Phi$ holds or does not hold, it is enough to solve the game *locally* for the start node. We have shown the correctness of IHP games and of the conversion to parity games [13]. Hence, if player V has a history-free winning strategy, then it has to be that $(T^1,\ldots,T^m,\ldots,T^k) \models_{\mathsf{V}} \Phi$; if player R has a winning strategy, it has to be that $(T^1,\ldots,T^m,\ldots,T^k) \not\models_{\mathsf{V}} \Phi$.

Example 1. Let V_0 be the view of $A_v = \{l_1, l_2\}$, $A_n = \emptyset$ and $A_c = \{h\}$. Consider the (termination-insensitive version of) IHP definition NI' (see Section 2.4):

$$NI'_{V_0} \;\widehat{=}\; \nu X.[l_1]_1\langle l_1\rangle_2 X \wedge [l_2]_1\langle l_2\rangle_2 X \wedge [h]_1 X.$$

Consider system T, given by the omega regular expression $(l_1 h l_2 \mid l_1 l_2)^\omega$. The respective IHP game is given in Fig. 1. We next illustrate the use of PGSolver for

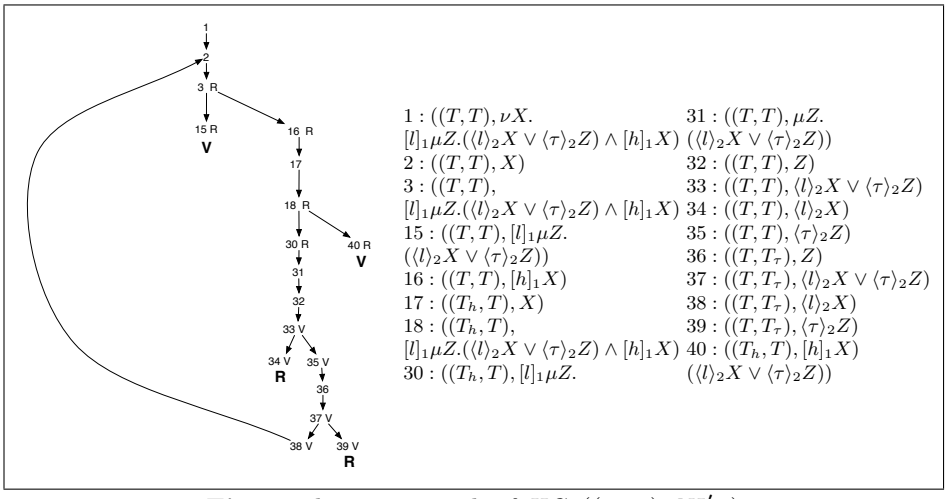


Fig. 1: The game graph of $HG_{\mathsf{V}}((T,T), NI'_{V_0})$

solving IHP games. We can convert game $HG_{\mathsf{V}}((T,T), NI'_{V_0})$ into a parity game $PG_{\mathsf{V}}((T,T), NI'_{V_0})$ (see Fig. 2, the conversion method is from [13]). The resulting parity game $PG_{\mathsf{V}}((T,T), NI'_{V_0})$ can be specified in PGSolver as follows:

```
parity 17;
0 2 0 1 "1";
1 2 0 2 "10";
2 1 1 3,4 "19";...
```

In such a specification, the first line is optional and gives the highest identifier. Each further line specifies a vertex by giving it an identity number, its parity, its owner, the vertices that are successors and an optional, symbolic name of the vertex [5]. PGSolver converts and solves the parity game globally:

```
Player 0 wins from nodes:
{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 13, 14, 15, 16}
with strategy [0->1,1->2,3->3,5->6,8->8,11->13,13->14,15->16,16->1]
Player 1 wins from nodes: {12, 17} with strategy [12->12,17->17]
```

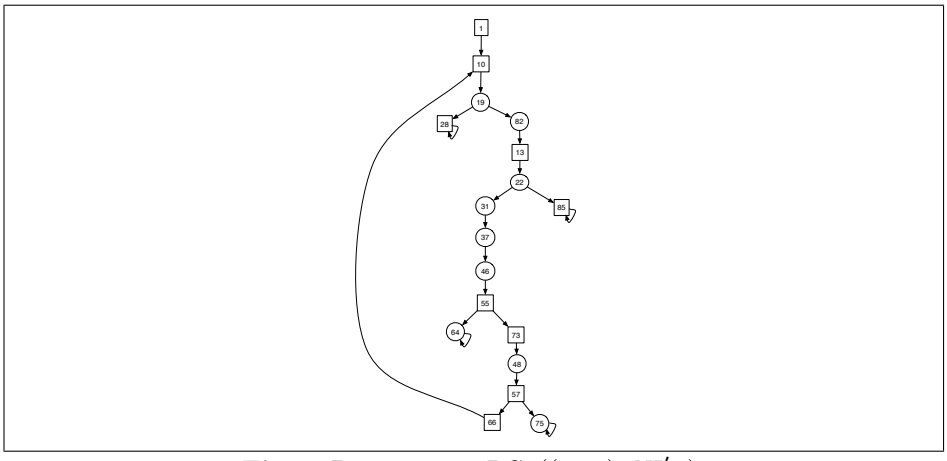


Fig. 2: Parity game $PG_{\mathsf{V}}((T,T), NI'_{V_0})$.

The winning positions for V and R are presented, as well as the positional strategy from each node. As V has a history-free winning strategy from the start node, it follows that $(T,T) \models NI'_{V_0}$. Hence $NI_{V_0}(T)$ holds [14]. The strategy itself (and not only its existence) is important, as it provides a witness why a hyperproperty holds or does not hold, as well as some intuition. Alternatively, if we were only interested in the validity of the checked formula, we could simply use PGSolver to perform local model checking and determine whether V has a winning strategy from the start node. One obvious way of automating this approach is to use a tool to create an IHP game from the formula and system. However, it is more convenient to convert the (product of) transition systems and formula into a parity game, and then solve that game. This would allow the use of existing tools (e.g. MLSolver and mCRL2 [7]) and can be fully automated.

4.3 Model Checking without Going through IHP Games

The alternative approach that constructs the parity game automatically and does not rely on an IHP game is presented next. The needed steps, given systems $(T^1, \ldots, T^m, \ldots, T^k)$ and formula Φ, are:

1. Make the product of the systems denoted $prod(T^1, \ldots, T^m, \ldots, T^k)$, as outlined in Section 4.1. This can be done in mCRL2 using the parallel composition operator ($||$) and disabling the simultaneous occurrence of multiple actions, as we are not interested in those for the product (see Section 4.2).
2. Convert system $prod(T^1, \ldots, T^m, \ldots, T^k)$ into MLSolver [6] format.
3. Convert formula Φ to work on the product $prod(T^1, \ldots, T^m, \ldots, T^k)$. In essence, each action in the formula is given a subscript linking it with a particular transition system. The result is $prod(\Phi)$ (see Section 4.1).
4. Use MLSolver to create a parity game for system $prod(T^1, \ldots, T^m, \ldots, T^k)$ and formula $prod(\Phi)$.
5. Use PGSolver to solve the parity game resulting from step 4.

The correctness of such an algorithm results from Theorem 2 and the suitable construction of the product. In principle, writing a tool for fully automating these steps is straightforward. For the model checking performed in this work, we only wrote a script automating (the most tedious) step 2.

Example 2. Let view V_1 be: $A_v = \{l\}$, $A_n = \{\tau\}$ and $A_c = \{h\}$. Consider the system S given as $(hl \mid \tau h\tau)^\omega$. We use mCRL2 to build the product $prod(S_1, S_2)$:

`act` $h_1, h_2, l_1, l_2, \tau_1, \tau_2$;
`proc` $T_1 = h_1.l_1.T_1 + \tau_1.h_1.\tau_1.T_1;\ T_2 = h_2.l_2.T_2 + \tau_2.h_2.\tau_2.T_2;\ S = T_1||T_2$;
`init allow`$(\{h_1, h_2, l_1, l_2, \tau_1, \tau_2\}, S)$;

The policy of interest is $prod(NI'_{V_1})$, here in the format for MLSolver:

$nu\ X\ .\ (([l_1]mu\ Z\ .\ (\langle l_2\rangle X \mid \langle \tau_2\rangle Z))\ \&\ ([h_1]X))\ .$

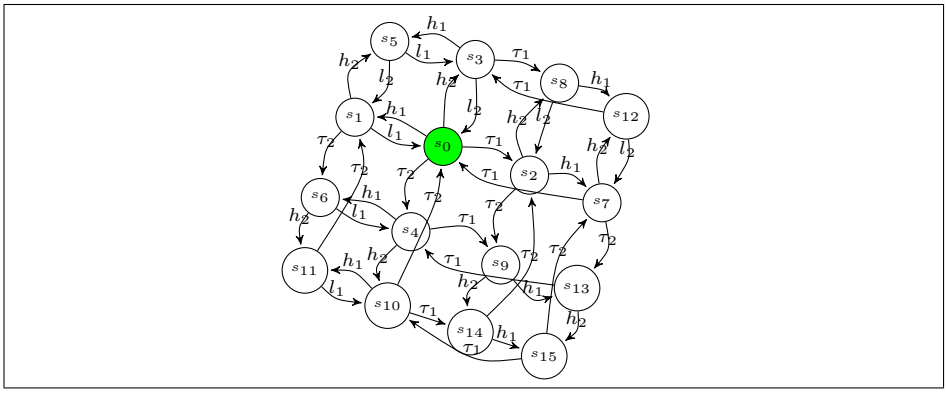


Fig. 3: The product $prod(S_1, S_2)$

The resulting transition system of $prod(S_1, S_2)$ can be seen in Fig. 3. The specification format (given next) is simple. The first line says that the transition system has 16 states, the second line gives the start state 1. Each further line specifies a state and lists its successors together with the respective labels. The specification of the transition system is encoded in MLSolver as follows:

lts 16;
start 1;
1 $h_1 : 2,\ \tau_1 : 3,\ h_2 : 4,\ \tau_2 : 5$;
2 $l_1 : 1,\ h_2 : 6,\ \tau_2 : 7$; ...

The output (checking whether $prod(NI'_{V_1})$ holds for $prod(S_1, S_2)$) in MLSolver:

```
Game has 15 states. Finished solving: 0.00 sec.
Transition system is no model of the formula!
```

Hence, we may conclude that $(S, S) \not\models NI'_{V_1}$. Instead of directly solving the game, it is possible to display the parity game using MLSolver and the strategy using PGSolver. For instance, the strategy is given as follows:

```
Player 0 wins from nodes: {2, 7} with strategy []
Player 1 wins from nodes: {0, 1, 3, 4, 5, 6, 8, 9, 10, 11, 12, 13,
14, 15} with strategy [1->3,3->4,5->6,6->8]
```

4.4 Experiments

A number of experiments on relatively large (albeit admittedly artificial) systems have been performed and reported in Table 1. The policy was always a variant of the policy $prod(NI'_V)$, the approach is the one from Section 4.3. Although the

Program	Domain size(product)	Time(in sec)-secure	Time(in sec)-insecure	Game Size
1	1816	0	0	124
2	10139	0	0	330
3	18415	1	1	672
4	22645	8	6	550
5	63271	13	9	1925
6	78392	34	19	2988
7	79500	36	20	3942
8	122492	60	55	4103

Table 1. Proof of concept evaluation of the approach from Section 4.3

system products become large quite fast, the games are substantially smaller. Proving a formula that is true requires more time than disproving the same formula for a slight variant of the system, having the same number of states (obtained by relabeling). We have not continued this experiment to obtain much larger games, as such experiments for games exist [5]. It would be interesting to further evaluate the approach on large reactive systems (left for future work).

5 Advantages of Model Checking via Games

Two of the presented approaches are game-based and this section presents some of the advantages of these in comparison with traditional approaches, such as the one presented in Section 4.1. Although game graphs similar to the one in Fig. 1 are useful for visualizing the respective games, there is more that can be done for understanding and analyzing IHP games. To show this, we introduce two new views of IHP games. These views allow focusing on interesting aspects of the game and are based on information calculated by the model checking algorithm.

We first present notation needed for the formalization of the game graphs (of IHP games) and the new views. Let $\sigma = \{V, R\}$ denote the set of players. A *game graph* can be formalized as the tuple $(\mathbb{V}, \rightarrow, L)$, where $\mathbb{V}$ is the set of positions (vertices), $\rightarrow$ is a binary relation on vertices, and the partial function $L : \mathbb{V} \rightarrow \sigma$, when defined, denotes whose turn it is at a position. The games we consider are positionally determined [13]. Hence, the strategy for player σ is a partial function $f_\sigma : \mathbb{V}_\sigma \rightarrow \mathbb{V}$, where $v \in \mathbb{V}_\sigma$ iff $L(v) = \sigma$. Let Win_σ be the set of winning positions for player σ. Technically, Win_V and Win_R are not part of the game graph and need to be calculated by the model checking algorithm.

We next present the *extended game graph view*, which enhances the game graph with the winning strategy and the complete winning positions. An *extended game graph* view is a 3-tuple (G, f^G_σ, Win^G), including the graph view G, the winning strategy f^G_σ, where $\sigma \in \{V, R\}$, for the winner from the start node,

and the set of winning positions for both players denoted $Win^G = Win^G_V \cup Win^G_R$. A simple extended game graph view is presented in Fig. 4. The graph additionally includes the progression of k-tuples of trees along the graph.

The second view is the *tree view* of the game — a finite list of the states visited in a play with stuttering states removed, starting at the root of the game tree and ending at the current position. A *tree view* for an IHP game $HG_{\mathsf{V}}((T^1, \ldots, T^k), \Phi)$ at position $HG_{\mathsf{V}}((T^1_l, \ldots, T^k_m), \Phi_n)$ is a list of states, starting with $(T^1, \ldots, T^k)$ and ending at $(T^1_l, \ldots, T^k_m)$, without repeating states. As an example, the tree view at position 12 in Fig. 4 is: $(T, T), (T_h, T), (T_{hl}, T)$. This is essentially a view showing the history of a game. The rules for this game come from the policy to be checked; the arena of the game (k-tuple of trees) together with the current position and the rules determine which next moves are possible. Such games capture the intuition behind H'-simulations well.

The ability of users to see and cross-reference both views introduced above and to play interactively in IHP games in the role of V can be useful for debugging: the fact that the system does not satisfy some policy (in $\mathcal{L}^k_\mu$) can be seen interactively as the inability of V to win the respective game [19]. An advantage of the views is that they are computed during the model checking process. A tool such as PGSolver comes up with a winning strategy and winning positions for the respective parity game automatically. Thus, the views can be constructed automatically with relatively minor modifications of existing tools.

In order to better visualize the strategy-based evidence and show that it is useful to enhance the user's understanding (why a policy does not hold when R has a winning strategy), we propose to combine the extended game graph view with the tree view. This visualization can be done by a specially constructed interactive tool, similar to the one proposed for property checking games [19]. The visualization starts by showing the part of the extended game graph view, for which no player is responsible (such as initial positions) in combination with the respective tree view. At each point in time there are several options. If the current vertex is labeled R, then V has no choice but to observe what the next position, determined by the winning strategy for R, is. The play goes into the new position, the extended view is changed appropriately and the tree view is changed when necessary. If the current vertex is labeled V, it is V's turn to choose a move and the tool presents the possible next moves. If the current vertex is not labeled, the game progresses automatically. At any time, both views are given to the user (Fig. 4 and 5) and the combined information helps V to make an informed choice. V is also able to backtrack and explore different plays.

Example 3. To illustrate these ideas, consider the system T given by the omega-regular expression $(hlh \mid \tau h \tau)^\omega$. We want to check whether $(T, T) \models NI'_{ti}$, where NI'_{ti} is the termination insensitive version of NI' (restricted to some view (A_v, A_n, A_c) with $A_v = \{l\}, A_n = \{\tau\}$ and $A_c = \{h\}$), given as follows:

$$NI'_{ti} \;\widehat{=}\; \nu X.\, ([l]_1 \mu Z.(\langle l \rangle_2 X \vee \langle \tau \rangle_2 Z)) \wedge [h]_1 X.$$

The extended game graph is given in Fig. 4. Player R has a winning strategy for the IHP game $HG_{\mathsf{V}}((T, T), NI'_{ti})$. The strategy is given as the red (grey) arrows. We next illustrate a visualization, building incrementally the views, that

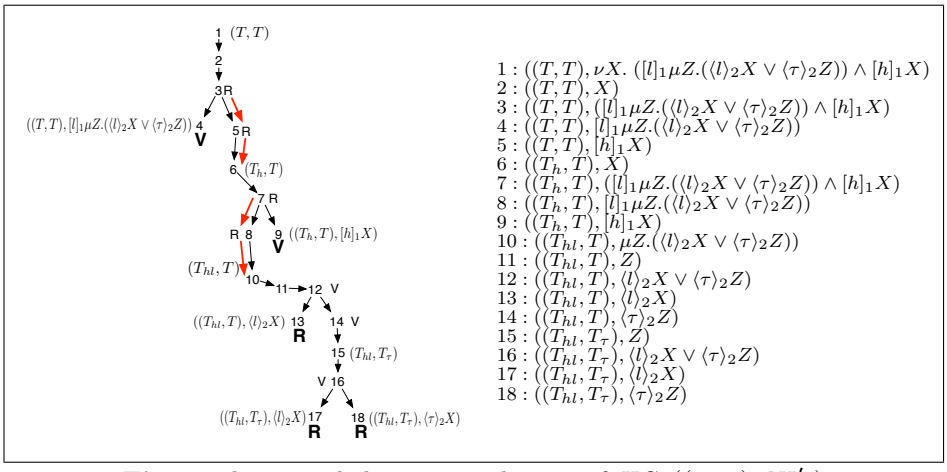
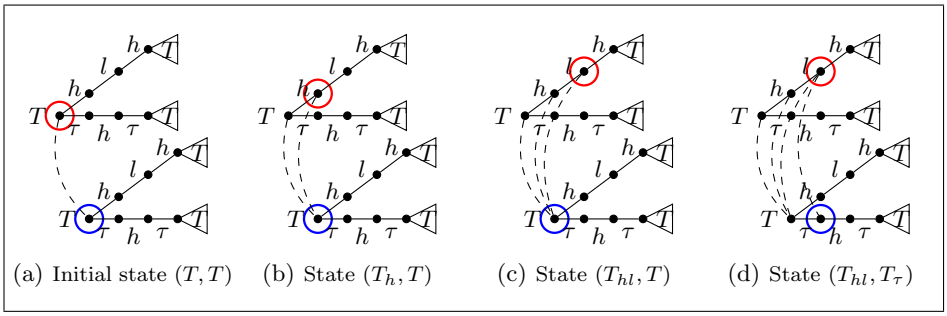


Fig. 4: The extended game graph view of $HG_{\mathsf{V}}((T,T), NI'_{ti})$

is helpful for the user to understand why the policy NF'_{ti} is violated by system T. Fig. 5 presents the positions (and their respective tree views) corresponding to the interesting plays of the game, namely the plays witnessing the winning strategy for R. The visualization of the tree view can be seen as a game in its own right: on the arena of two copies of T, player R moves in the first tree and has a red (light grey) token, player V moves in the second tree and has a blue (dark grey) token. The rules are implicit and depend on the policy, here NF'_{ti}.



(a) Initial state (T,T) (b) State (T_h,T) (c) State (T_{hl},T) (d) State (T_{hl},T_τ)

Fig. 5: The tree views of game $HG_{\mathsf{V}}((T,T), NF'_{ti})$ at different positions/states

Initially, player V (i.e. the user) sees the extended graph until position 3 and the tree view in Fig. 5(a). The next two moves are automatic and determined by the winning strategy for R. The play is now at position 6, the extended view consists of positions $1,\ldots,6$ and the tree view is the one in Fig. 5(b). At positions 7 and 8 R follows her winning strategy. The extended game graph progresses and is built to position 10, the respective tree view is the one in Fig. 5(c). Positions 10 and 11 are not labeled so the play goes to position 12, which is the

first possible choice for player V: she is now asked to choose between actions l and τ, i.e. whether to go to position $((T_{hl}, T), \langle l \rangle_2 X)$ or $((T_{hl}, T), \langle \tau \rangle_2 X)$. If V chooses the first option (position 13), she loses the game. This can be seen in the tree view, as V has to make an l-move in the second tree, but making such a move is impossible. The user can backtrack to position 12 and choose to go to position 14 instead. Player V continues playing and now the tree view changes to the one in Fig. 5(d). The play then goes on until position 16, where V has to again choose between l and τ, i.e. whether to go to positions $((T_{hl}, T_\tau), \langle l \rangle_2 X)$ or $((T_{hl}, T_\tau), \langle \tau \rangle_2 X)$. Choosing either position, the user loses. Consulting the tree view in Fig. 5(d), the user can see that neither action l nor τ can be performed at position (T_{hl}, T_τ) and this is the reason for losing. Interestingly, this is also the reason why the policy does not hold on the system. The two views have helped identify and visualize the problem. Thus the views can be used to provide useful intuition as to what goes wrong with the system-policy interaction.

The techniques presented here can be used to explore any IHP game in $\mathcal{L}_\mu^k$ on any finite-state system. The user can systematically explore different paths and strategies to play against player R. As shown above, this helps with understanding both the policy and system behavior, as well as their interaction.

6 Discussion and related work

This work enables practical reasoning about security-relevant hyperproperties via IHPs and games. To achieve this, we propose two game-based model checking approaches reusing some results on model checking parity games, in particular algorithms and tools [19,22,20,17]. In this sense, using the tool PGSolver is particularly beneficial, as it implements most known algorithms for model checking games, both local, on-the-fly and global ones, as well as heuristics to improve performance. Thus, depending on the particular problem, one may choose an algorithm with good theoretical properties or experiment with a multitude of different algorithms. More importantly, such a tool is an excellent candidate to build upon, as it calculates all the data needed to create the proposed views. As a result, one can easily build an interactive visualization tool, allowing users to play against player R to enhance their understanding of problematic system-policy interaction. Building such a tool is left for future work.

The idea of using strategies for analyzing why a system does not satisfy a policy is not new. Stevens and Stirling [19] present a similar idea of using strategies to construct and prove the correctness of local, on-the-fly model checking algorithms. They also present the idea of visualizing why a property does not hold as a byproduct of the local model checking algorithm finding the strategy. In comparison with our idea of visualization via views, their visualization seems to be less intuitive (it is given by a command line tool). More importantly, our views present a useful separation of the states, moves and rules of the game. In addition, the views present a visualization of H'-simulations.

We have not experimented with model checking very large systems with respect to IHPs, as, due to our reduction of the problem to solving parity games, doing so would be dependent on the concrete algorithms for solving parity games.

Instead, we present a short survey of the time complexity of such algorithms. Currently, the existence of a polynomial time algorithm for solving parity games is a major open problem [8]. The reason is that solving a parity game is equivalent to the problems of model checking the mu-calculus and the complementation of ω-tree automata [18]. Most of the algorithms for solving parity games run in exponential time, for instance this is the case for the recursive algorithm by Zielonka [22] and the strategy improvement one by Vöge and Jurdziński [20].

Surprisingly, the promising and well-behaved in practice (see [5]) policy iteration algorithms, proposed by Vöge and Jurdziński [20] and Schewe [17], can also take exponential time on some parity games [4]. The theoretically fastest algorithms for the problem are randomized algorithms by Kalai [9] as well as by Matoušek, Sharir and Welzl [11]. The fastest deterministic subexponential (in the size of the game) algorithm for the solution of parity games uses only polynomial space and runs in time $2^{O(\sqrt{n\ log\ n})}$, where n is the size of the game.

Although there is no proof that there are polynomial algorithms for solving parity games, Friedmann and Lange [5] show that parity games can be solved efficiently in practice. One of their results is that the recursive algorithm by Zielonka [22] has the best performance in practice, being able to handle games of size up to 1 million nodes. Although these results look promising, we acknowledge that the topic of practical model checking of IHPs needs further exploration.

7 Conclusion

We have developed a verification methodology for IHPs in $\mathcal{IL}^k_\mu$ and $\mathcal{L}^k_\mu$, based on a characterization of the satisfaction relation between a system and an IHP in terms of playing a game. This is the first generic verification methodology (via H'-simulations and games) that can reason about a class of liveness hyperproperties (i.e. the coinductive variants of possibilistic information flow policies from our recent work [14]), although it is not limited to liveness hyperproperties.

In addition, we have demonstrated the potential of practical game-based model checking of IHPs using two approaches based on off-the-shelf tools. The main advantage of these game-based approaches is the possibility of using a winning strategy as a witness why a particular system is or is not secure with respect to some policy. We also proposed two views that have the potential to facilitate the illustration of system-policy interactions. Possible directions for future work include extending the approaches to decidable classes of infinite state systems and developing a game-theoretic semantics for IHP-preserving refinement.

References

1. Henrik Reif Andersen. A Polyadic Modal μ-Calculus. Technical Report 1994-145, Technical University of Denmark (DTU), 1994.
2. Julian Bradfield and Colin Stirling. Modal mu-calculi. In *Handbook of Modal Logic*, pages 721–756. Elsevier, 2007.
3. Michael R. Clarkson and Fred B. Schneider. Hyperproperties. *Journal of Computer Security*, 18:1157–1210, September 2010.
4. Oliver Friedmann. An exponential lower bound for the latest deterministic strategy iteration algorithms. *Logical Methods in Computer Science*, 7(3), 2011.

5. Oliver Friedmann and Martin Lange. Solving parity games in practice. In *Proceedings of the 7th International Symposium on Automated Technology for Verification and Analysis*, ATVA '09, pages 182–196, Berlin, Heidelberg, 2009. Springer-Verlag.
6. Oliver Friedmann and Martin Lange. A Solver for Modal Fixpoint Logics. *Electron. Notes Theor. Comput. Sci.*, 262:99–111, May 2010.
7. Jan Friso Groote, Aad Mathijssen, Michel Reniers, Yaroslav Usenko, and Muck van Weerdenburg. The Formal Specification Language mCRL2. In Ed Brinksma, David Harel, Angelika Mader, Perdita Stevens, and Roel Wieringa, editors, *Methods for Modelling Software Systems (MMOSS)*, number 06351 in Dagstuhl Seminar Proceedings, Dagstuhl, Germany, 2007.
8. M. Jurdziński, M. Paterson, and U. Zwick. A deterministic subexponential algorithm for solving parity games. In *Proceedings of ACM-SIAM Symposium on Discrete Algorithms, SODA 2006*, pages 117–123. ACM/SIAM, 2006.
9. Gil Kalai. Linear programming, the simplex algorithm and simple polytopes. *Mathematical Programming*, 79:217–233, 1997.
10. Heiko Mantel. *A Uniform Framework for the Formal Specification and Verification of Information Flow Security*. PhD thesis, Universität des Saarlandes, Saarbrücken, Germany, July 2003.
11. Jiří Matoušek, Micha Sharir, and Emo Welzl. A subexponential bound for linear programming. In *Proceedings of the eighth annual symposium on Computational geometry*, SCG '92, pages 1–8, New York, NY, USA, 1992. ACM.
12. Daryl McCullough. Specifications for multi-level security and a hook-up. In *Proceedings of the 1987 IEEE Symposium on Security and Privacy*, pages 161–166, Los Alamitos, CA, USA, 1987. IEEE Computer Society.
13. Dimiter Milushev. *Reasoning about Hyperproperties*. PhD thesis, KU Leuven, Heverlee, Belgium, June 2013.
14. Dimiter Milushev and Dave Clarke. Coinductive unwinding of security-relevant hyperproperties. In *Proceedings of the 17th Nordic Conference on Secure IT Systems*, volume 7617 of *LNCS*, pages 121–136. Springer, October 2012.
15. Dimiter Milushev and Dave Clarke. Towards Incrementalization of Holistic Hyperproperties. In *Proceedings of the First International Conference on Principles of Security and Trust*, volume 7215 of *LNCS*, pages 329–348. Springer, March 2012.
16. Jan J. M. M. Rutten. Automata and Coinduction (An Exercise in Coalgebra). In Davide Sangiorgi and Robert de Simone, editors, *CONCUR*, volume 1466 of *LNCS*, pages 194–218. Springer, 1998.
17. Sven Schewe. An optimal strategy improvement algorithm for solving parity and payoff games. In *Proceedings of the 22nd international workshop on Computer Science Logic*, CSL '08, pages 369–384, Berlin, Heidelberg, 2008. Springer-Verlag.
18. Colin Stirling. *Modal and temporal properties of processes*. Springer-Verlag, New York, NY, USA, 2001.
19. Colin Stirling and Perdita Stevens. Practical model-checking using games. In *TACAS 1998*, number 1384 in LNCS, pages 85–101, 1998.
20. Jens Vöge and Marcin Jurdziński. A Discrete Strategy Improvement Algorithm for Solving Parity Games. In *Proceedings of the 12th International Conference on Computer Aided Verification, CAV 2000*, volume 1855 of *LNCS*, pages 202–215. Springer-Verlag, 2000.
21. A. Zakinthinos and E. S. Lee. A general theory of security properties. In *Proceedings of the IEEE Symposium on Security and Privacy*, SP '97, pages 94–102, Washington, DC, USA, 1997. IEEE Computer Society.
22. Wieslaw Zielonka. Infinite games on finitely coloured graphs with applications to automata on infinite trees. *Theoretical Computer Science*, 200(1–2):135–183, 1998.