


Hyperproperty Verification as CHC Satisfiability

Shachar Itzhaky[1(✉)], Sharon Shoham[2], and Yakir Vizel[1]

[1] Technion, Haifa, Israel
shachari@cs.technion.ac.il
[2] Tel-Aviv University, Tel Aviv-Yafo, Israel

Abstract. Hyperproperties specify the behavior of a system across multiple executions, and are an important extension of regular temporal properties. So far, such properties have resisted comprehensive treatment by software model-checking approaches such as IC3/PDR, due to the need to find not only an inductive invariant but also a *total* alignment of different executions that facilitates simpler inductive invariants.
We show how this treatment is achieved via a reduction from the verification problem of $\forall^*\exists^*$ hyperproperties to Constrained Horn Clauses (CHCs). Our starting point is a set of universally quantified formulas in first-order logic (modulo theories) that encode the verification of $\forall^*\exists^*$ hyperproperties over infinite-state transition systems. The first-order encoding uses uninterpreted predicates to capture the (1) witness function for existential quantification over traces, (2) alignment of executions, and (3) corresponding inductive invariant. Such an encoding was previously proposed for k-safety properties. Unfortunately, finding a satisfying model for the resulting first-order formulas is beyond reach for modern first-order satisfiability solvers. Previous works tackled this obstacle by developing specialized solvers for the aforementioned first-order formulas. In contrast, we show that the same problems can be encoded as CHCs and solved by existing CHC solvers. CHC solvers take advantage of the unique structure of CHC formulas and handle the combination of quantifiers with theories and uninterpreted predicates more efficiently.
Our key technical contribution is a logical transformation of the aforementioned sets of first-order formulas to equi-satisfiable sets of CHCs. The transformation to CHCs is sound and complete, and applying it to the first-order formulas that encode verification of hyperproperties leads to a CHC encoding of these problems. We implemented the CHC encoding in a prototype tool and show that, using existing CHC solvers for solving the CHCs, the approach already outperforms state-of-the-art tools for hyperproperty verification by orders of magnitude.

1 Introduction

Hyperproperties [15] are properties that relate multiple execution traces, either taken from a single program or from multiple programs. Checking such properties is known as *relational verification*, and is essential when reasoning about security policies, program equivalence, concurrency protocols, etc. Existing specification languages for hyperproperties [14,6,43] extend standard ones, e.g., temporal logic or Hoare logic, with (explicit or implicit) quantification over traces. This shifts

S. Weirich (Ed.): ESOP 2024, LNCS 14577, pp. 212–241, 2024.
https://doi.org/10.1007/978-3-031-57267-8_9

the focus from properties of individual traces to properties of *sets* of traces. For example, k-safety [15] is a class of hyperproperties, where k universal quantifiers are used to define a relational invariant over states originating from k traces.

This paper addresses verification of hyperproperties with $\forall^*\exists^*$ quantification over traces and a body of the form $\square\phi$ (where $\square$ stands for "globally"). This fragment captures many hypersafety (e.g., the aforementioned k-safety) and hyperliveness properties, and was shown by [8] to express a wide class of properties of interest, including generalized non-interference (GNI) [38].

Verification of hyperproperties is more challenging than verification of single-trace properties, and, as a result, has gained a lot of attention in recent years. Unlike single-trace properties, verification of properties of k traces requires the discovery of *relational* inductive invariants, which define the relation between states of k execution traces. Since the construction of invariants that hold between *any* k reachable states is hard (or even impossible, depending on the assertion logic), proving hyperproperties often hinges on finding an *alignment* of any k traces such that the invariant only needs to describe aligned states.

In the case of k-safety properties, an alignment of traces is often given by a *self composition* [5,44] of the program, composing different copies of the program (or several different programs) together, *e.g.*, by running the different copies in lockstep [48] or by more sophisticated composition schemes, *e.g.*, [24]. While self composition allows to reduce k-safety verification to standard safety verification, this reduction requires to choose the alignment of the different copies a-priori. The choice of alignment, however, has a significant effect on the complexity of the inductive invariants themselves, as demonstrated by [41]. This renders the standard reduction from k-safety verification to safety verification, based on a fixed alignment, impractical in many cases. As a result, finding a good alignment as part of relational verification has been a topic of interest in recent years [43,27,45,6,8].

In the case of hyperliveness properties that stem from the use of existential quantification over traces (*i.e.* $\forall^*\exists^*$ properties), complexity rises further. Verifying such hyperliveness properties calls for finding "witness" traces that match the universally quantified traces, in addition to the relational invariant and alignment. This reduces verification of $\forall^*\exists^*$ properties to the problem of inferring three ingredients: (i) a witness function for existential quantification over traces, (ii) an alignment of traces, and (iii) a corresponding relational inductive invariant. These ingredients are all interdependent: different witnesses call for different alignments and give rise to different invariants, with different levels of complexity. It is therefore desirable to search for the *combination* of the three of them *simultaneously*, which is the focus of this paper.

We propose a novel reduction from verification of hyperproperties with a $\forall^*\exists^*$ quantification prefix over infinite-state transition systems to satisfiability of Constrained Horn Clauses (CHCs) [11,10], also known as CHC-SAT. Importantly, the reduction does not fix any of the aforementioned verification ingredients, in particular, the alignment, a-priori. Instead, it is based on a CHC encoding of their joint requirements. The unique structure of CHCs makes it

possible to adopt software model checking techniques (e.g. interpolation [39], IC3/PDR [32,35]) for solving them. Our reduction, thus, allows to use state-of-the-art CHC solvers [28,33,31,49] to achieve a highly efficient hyperproperty verification procedure.

While it is known that safety verification can be reduced to CHC-SAT, we are the first to show how inferring the combination of a witness function, a trace alignment and an inductive invariant for hyperproperties of the $\forall^*\exists^*$-fragment can be reduced to CHC-SAT.

The first step of our reduction to CHC-SAT is an encoding of the joint requirements of the witness-alignment-invariant ingredients as a set of universally quantified formulas in first-order logic (FOL) modulo theories, where uninterpreted predicates capture the witness, alignment and invariant, and first-order theories (*e.g.*, arithmetic and arrays) are used for modeling the transition system and the requirements. Such an encoding has been proposed by [41] for the problem of finding an invariant together with an alignment in the context of verification of k-safety properties (the universally quantified subset of this fragment). We extend their FOL encoding to $\forall^*\exists^*$ properties, based on the game semantics introduced in [8].

Unfortunately, the resulting FOL formulas are beyond what modern first-order satisfiability solvers can handle due to a combination of quantifiers with theories and uninterpreted predicates. In particular, the FOL formulas are not in the form of CHCs. As a result, previous works [41,45] that used a similar encoding could not rely on a (single) CHC-SAT query to find the alignment and invariant simultaneously. Instead, [41] resorted to an enumeration of potential alignments, using a separate CHC-SAT query to search for an inductive invariant (in a restricted language) for each candidate alignment. [45] developed a specialized solver that is able to handle these non-CHC formulas directly.

In contrast to previous works, we introduce a second step where we transform the set of universally quantified FOL formulas to a set of universally quantified CHCs. This step—which is also the key technical contribution of the paper—allows us to use any CHC solver for hyperproperty verification, and benefit from current and future developments in this lively area of research. We emphasize that the transformation to CHCs is surprising since it allows us to overcome a seemingly unavoidable obstacle: a disjunction of atomic formulas involving unknown predicates, which arises from the encoding of a choice between different alignment and witness options.

We implemented the reduction of $\forall^*\exists^*$-hyperproperty verification to CHC-SAT in a tool called HyHorn, on top of Z3 [23], using Spacer [31] as a CHC solver. Our results show that HyHorn is very efficient in verifying $\forall^*\exists^*$-hyperproperties, outperforming the state-of-the-art [45,8,41] by orders of magnitude.

Our main contributions are:

- We develop a satisfiability-preserving transformation of first-order formulas of a certain form to CHCs. The transformation is accompanied by a bi-directional translation of solutions.

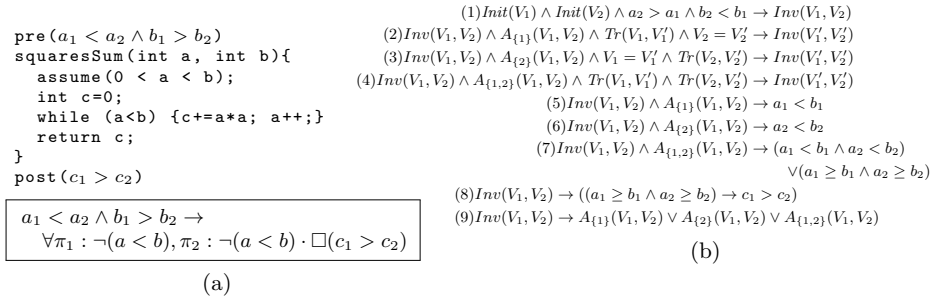

Fig. 1: (a) A program that computes the sum of squares of integer interval $[a, b)$ with a 2-safety specification for it, and (b) its first-order encoding.

- We apply the transformation to obtain, for the first time, a sound and complete reduction from verification of $\forall^*\exists^*$-OHyperLTL (w.r.t. a game semantics) to CHC-SAT. The reduction captures searching for an alignment, an $\exists^*$-witness function and an inductive invariant simultaneously. It is applicable to infinite-state transition systems, with the caveat that their branching degree needs to be finite (bounded by a constant) if the hyperproperty includes $\exists^*$ quantification.
- To handle $\exists^*$ in the presence of unbounded nondeterminism, we incorporate into the CHC encoding a sound abstraction based on a set of underapproximations ("restrictions").
- We implement a tool, HyHorn, that constructs CHCs for $\forall^*\exists^*$-OHyperLTL specifications, and solves them using Spacer. In most cases, HyHorn discovers the solution completely automatically, while in some, it uses predicate abstraction, based on user-provided predicates.

2 Overview

We illustrate our approach for verifying hyperproperties by reduction to CHC-SAT. We start with the simpler case of k-safety properties, followed by the more general case of $\forall^*\exists^*$ hyperproperties.

2.1 Motivating Example

As a means for highlighting the challenges in verifying hyperproperties, and, in particular, in reducing the problem to CHC solving, we present the example program `squaresSum` and its 2-safety specification from [41] in Fig. 1a. Given positive integers $a < b$, the program computes the sum of squares of all integers in the interval $[a, b)$. `squaresSum` is monotone in the sense that as the input interval increases, so does the output c. Formally, this is a 2-safety property that requires that whenever two traces satisfy the pre-condition $[a_2, b_2) \subset [a_1, b_1)$,

they also satisfy the post-condition $c_1 > c_2$, where variable indices correspond to the traces that they represent. This is a special case of k-safety, where the relational property is checked at the end of the executions. More generally, we consider k-safety properties where the relational property is specified at designated *observation points* (explained in Sec. 3).

To verify the 2-safety property, a prominent approach is to reduce the problem to a regular safety verification problem by composing the program with itself (known as "self composition"). There are (infinitely) many possibilities for aligning the traces in the composed system, and the alignment chosen has direct impact on the complexity of the inductive invariant needed to establish safety. For example, if the two traces of `squaresSum` are aligned in lockstep, then initially $c_1 = c_2$, after one step, $c_1 < c_2$, and only later on, $c_1 > c_2$. Showing that $c_1 > c_2$ at the end requires tracking the difference $c_1 - c_2$, which is a complex value because it involves the sum of squares itself. This cannot be captured by an inductive invariant in first-order logic using theories currently supported by automated solvers (*e.g.*, linear arithmetic) and is therefore beyond reach for state-of-the-art solvers. On the other hand, if the second trace, whose input is the smaller interval, "waits" for a_1 and a_2 to coincide before proceeding in lockstep, then the property that $c_1 > c_2$ becomes inductive (except for the first step), greatly simplifying the inductive invariant. It is therefore important to consider the alignment and the (relational) inductive invariant together.

The requirements that the alignment and inductive invariant need to satisfy can be formulated in first-order logic [41]. To do so, we denote the program variables by $V = \langle a, b, c \rangle$. We express the initial states and program steps as formulas over V (and primed variant V') : $\mathit{Init}(V) \triangleq a > 0 \wedge b > a \wedge c = 0$, $\mathit{Tr}(V, V') \triangleq a < b \wedge c' = c + a \cdot a \wedge a' = a + 1 \wedge b' = b$. To reason about two traces, we use two copies of V, denoted V_1 and V_2. We introduce "unknown" predicates $\mathit{Inv}, A_{\{1\}}, A_{\{2\}}, A_{\{1,2\}}$ over $\langle V_1, V_2 \rangle$ to capture the inductive invariant and desired alignment of the traces. $\{A_u\}_u$ define an *arbiter* that, when A_u is satisfied, schedules the steps of the traces according to u (for example, schedule $u = \{1\}$ stands for a step in trace 1 and a stutter in trace 2). The arbiter therefore determines the alignment of the traces. The inductive invariant Inv relates states of the two copies of the program, making it *relational*.

The problem of searching for the alignment and the inductive invariant simultaneously is then posed as a satisfiability problem (modulo the theory of arithmetic) of the formulas in Fig. 1b. To ensure that the arbiter, which determines the alignment, does not avoid violations of the post-condition by making one of the traces stutter forever s.t. it never reaches its final state, formulas 5-7 require that the arbiter only schedule a trace if it has not exited the loop, unless both traces exited the loop (in which case both are scheduled). This "validity" requirement means that, at the latest, the arbiter must schedule a trace when the other reaches the final state. Formulas 1-4 then ensure that all states that are reachable, subject to the steps permitted by the arbiter, must satisfy Inv. Specifically, the first formula ensures the initiation condition of the inductive invariant: the invariant satisfies the pre-condition and includes all the initial states

of the composed system. Formulas 2-4 ensure the consecution of the invariant under every choice the arbiter makes. The 8th formula ensures the safety of the invariant and the last formula mandates that there is always at least one choice that is enabled, and that the system never reaches a "stuck" state.

An interpretation for the unknown predicates $Inv, A_{\{1\}}, A_{\{2\}}, A_{\{1,2\}}$ defines an arbiter and a corresponding inductive invariant. A possible solution is

$$A_{\{1\}}(V_1, V_2) \triangleq a_1 < a_2 \vee (b_2 \leq a_1 < b_1) \qquad A_{\{2\}}(V_1, V_2) \triangleq \bot$$
$$A_{\{1,2\}}(V_1, V_2) \triangleq (a_1 = a_2 \wedge a_1 < b_2) \vee a_1 \geq b_1$$
$$Inv(V_1, V_2) \triangleq 0 < a_1 \leq b_1 \wedge 0 < a_2 \leq b_2 \wedge \big((a_1 < a_2 \wedge c_1 \geq c_2) \vee (a_1 \geq a_2 \wedge c_1 > c_2)\big)$$

This solution captures the arbiter that makes the second trace wait until $a_1 = a_2$, then makes both traces proceed together until the second one exits its loop, in which case the first trace continues to execute alone until it also exits its loop and both traces are again (vacuously) scheduled together. The solution to *Inv* captures the corresponding inductive invariant previously discussed.

2.2 Challenges in Encoding Hyperproperty Verification as CHC-SAT

The formulas of Fig. 1b, with the exception of the last one, are constrained *Horn* clauses. That is, when the implications in these formulas are converted to disjunctions, at most one predicate application appears positively in each clause.

Alas, the presence of the last formula precludes direct application of existing CHC solvers. The problem is the disjunction on the right hand side of the implication. Such a disjunction appears to be crucial for a correct encoding of the problem. The reason is that uninterpreted predicates designate semantic *relations*. With such predicates denoting the choice of schedule, it is easy to drop into a vacuous solution where some states have no corresponding choice and are essentially "stuck", unsoundly making a post-condition violation unreachable. Encoding the requirement that every state have a schedule results in a clause with multiple occurrences of positive literals, capturing inherent disjunctions over the possible choices, which are not Horn. In particular, these disjunctions cannot be eliminated by renaming [37].

Previous works tackled this obstacle either by employing explicit enumeration of alignments that satisfy the non-Horn clause to avoid the disjunction [41], or by developing specialized techniques that are able to handle such disjunctions [45].

2.3 Our Approach: Transformation to CHC

In this paper, we show that the problem of searching for an alignment together with a (relational) inductive invariant can be encoded using CHCs, allowing us to reduce the problem to CHC-SAT, *without* fixing the alignment *a priori*.

A key insight of our reduction to CHC-SAT is the use of "doomed" states as a way to avoid the problematic disjunction over all choices of schedules. We refer to a given state as "doomed" if it necessarily reaches a state that violates

$$D_{\{1\}}(V_1,V_2) \wedge D_{\{2\}}(V_1,V_2) \wedge D_{\{1,2\}}(V_1,V_2) \wedge \mathit{Init}(V_1) \wedge \mathit{Init}(V_2) \wedge a_2 > a_1 \wedge b_2 < b_1 \rightarrow \bot$$
$$\neg(a_1 \geq b_1 \wedge a_2 \geq b_2 \rightarrow c_1 > c_2) \rightarrow D_{\{1\}}(V_1,V_2)$$
$$\neg(a_1 \geq b_1 \wedge a_2 \geq b_2 \rightarrow c_1 > c_2) \rightarrow D_{\{2\}}(V_1,V_2)$$
$$\neg(a_1 \geq b_1 \wedge a_2 \geq b_2 \rightarrow c_1 > c_2) \rightarrow D_{\{1,2\}}(V_1,V_2)$$
$$\neg(a_1 < b_1) \rightarrow D_{\{1\}}(V_1,V_2)$$
$$\neg(a_2 < b_2) \rightarrow D_{\{2\}}(V_1,V_2)$$
$$\neg(a_1 < b_1 \wedge a_2 < b_2) \wedge \neg(a_1 \geq b_1 \wedge a_2 \geq b_2) \rightarrow D_{\{1,2\}}(V_1,V_2)$$
$$D_{\{1\}}(V_1',V_2') \wedge D_{\{2\}}(V_1',V_2') \wedge D_{\{1,2\}}(V_1',V_2') \wedge \mathit{Tr}(V_1,V_1') \wedge V_2 = V_2' \rightarrow D_{\{1\}}(V_1,V_2)$$
$$D_{\{1\}}(V_1',V_2') \wedge D_{\{2\}}(V_1',V_2') \wedge D_{\{1,2\}}(V_1',V_2') \wedge V_1 = V_1' \wedge \mathit{Tr}(V_2,V_2') \rightarrow D_{\{2\}}(V_1,V_2)$$
$$D_{\{1\}}(V_1',V_2') \wedge D_{\{2\}}(V_1',V_2') \wedge D_{\{1,2\}}(V_1',V_2') \wedge \mathit{Tr}(V_1,V_1') \wedge \mathit{Tr}(V_2,V_2') \rightarrow D_{\{1,2\}}(V_1,V_2)$$

Fig. 2: CHC encoding of Fig. 1a.

the hyperproperty along *every* valid alignment (as opposed to *some* in the direct encoding). Importantly, due to this conjunctive nature, doomed states lend themselves to a Horn encoding. If an initial state is identified as doomed (i.e., the CHCs are unsatisfiable), then the property is violated and a counterexample can be retrieved. Otherwise, if the set of initial states does not intersect the set of doomed states, then the hyperproperty is proved. Moreover, given an interpretation of the unknown predicates in which the initial states are not doomed, an alignment and a corresponding inductive invariant can be retrieved.

Based on this insight, in Sec. 4, we develop a general transformation of formulas of a certain form, to an equi-satisfiable set of CHCs. Furthermore, we provide a transformation of solutions between the two formulations (in both directions). The first-order formulas to which the transformation is applicable follow the overall structure of the formulas in Fig. 1b, but are somewhat more general. For example, some of the unknown predicates may have additional arguments, which turn out to be useful when considering a broader class of hyperproperties beyond k-safety ($\forall^*\exists^*$).

In Sec. 5 we apply the transformation of Sec. 4 to reduce k-safety verification to CHC-SAT. When applying the transformation on the formulas encoding our running example (Fig. 1b), we obtain the set of CHCs depicted in Fig. 2 over unknown predicates $D_{\{1\}}, D_{\{2\}}, D_{\{1,2\}}$.

In the CHCs of Fig. 2, an unknown predicate D_u represents states that are "doomed" if schedule u is chosen. The first CHC requires that no initial state that satisfies the pre-condition is completely doomed, i.e., for every such state there is a schedule for which it is not doomed. The remaining CHCs encode the properties of doomed states for each schedule. For example, the CHCs where $D_{\{1\}}$ is in the head (right hand side of the implication) imply that a state is doomed for schedule $\{1\}$ if: (a) it violates the post-condition, (b) it already exited the loop and hence trace 1 cannot be the only trace to be scheduled, or (c) it is the pre-state of a transition taken by 1 leading to a post-state that is doomed for every choice u.

A solution to the CHCs in Fig. 2 can be obtained from the solution to the formulas in Fig. 1b by $D_u \triangleq \neg(\mathit{Inv} \wedge A_u)$ for every $u \in \{\{1\},\{2\},\{1,2\}\}$.

More generally, in Sec. 4, we show a bi-directional transformation of solutions.

2.4 Beyond k-Safety

Our transformation to CHCs is not limited to an encoding of k-safety, but also generalizes to hyperproperties that use $\forall^*\exists^*$ quantification over traces, as presented in Sec. 6.

Hyperproperties with existential trace quantification become meaningful in the presence of nondeterminism in the program. For an example of such a property, consider a nondeterministic variant of `squaresSum` where the assignment `c += a * a` is replaced by `if (*) c += a * a`. That is, the increment of `c` may nondeterministically be skipped. We may now wish to verify that, if $[a_2, b_2) = [a_1, b_1)$, then for every trace from input $[a_1, b_1)$ there exists a trace from input $[a_2, b_2)$ such that when both terminate, $c_1 \neq c_2$. This is a $\forall\exists$-hyperproperty.

To verify such properties, a "witness" function is needed to map the universally quantified traces to the corresponding existentially quantified traces such that the body of the formula holds for the combination of the traces. Even if a witness function is known, to verify that the combination of the traces satisfies the body of the formula, we still need to find a proper alignment of the traces and an inductive invariant. As in the case of k-safety, these components are all interdependent, making it desirable to search for all of them together.

In general, the witness function for the existentially quantified traces may need to depend on the full universally quantified traces. However, [8] defines a sound but incomplete *game semantics*, in which the witness function essentially constructs the existentially quantified traces step-by-step, in response to moves of a "falsifier" who reveals the universally quantified traces step-by-step.

We show in Sec. 6.1 that the problem of searching for a step-by-step witness function, an alignment and a (relational) inductive invariant can be encoded in first-order logic, and the encoding is amenable to our transformation to CHCs. This results in a sound and complete CHC encoding of the game semantics of [8] for transition systems whose branching degree is bounded by a constant, which we henceforth refer to as "finite branching".

The idea in the $\forall^*\exists^*$-first-order encoding is to let the unknown predicates A_u specify not only the schedules chosen by the arbiter but also the choice of existentially quantified traces for the witness function. To do so, we assign a unique label to each of the possible transitions, and use these labels to identify the transitions along the traces. In this encoding, instead of u denoting a schedule only, it now denotes both a schedule and a choice of labels identifying the next transitions in the existentially quantified traces according to the witness function. Furthermore, the A_u predicates receive additional arguments that represent the next labels along the universally quantified traces.

For example, in the nondeterministic variant of `squaresSum`, there are at most two possible transitions in each control location. We therefore introduce two labels to distinguish between these possibilities: `i` for "increment" and `s` for "skip". The predicates that describe the schedules and the choices of existentially quantified traces for the $\forall\exists$-hyperproperty of interest are:

$$A_{\{1\},\mathtt{i}}, A_{\{2\},\mathtt{i}}, A_{\{1,2\},\mathtt{i}}, A_{\{1\},\mathtt{s}}, A_{\{2\},\mathtt{s}}, A_{\{1,2\},\mathtt{s}}.$$

They are defined over $\langle V_1, V_2, a\rangle$, where a ranges over the possible labels.

Note that in this encoding, the A_u predicates are no longer defined over $\langle V_1, V_2\rangle$ only, but have additional arguments for the labels of the universally quantified traces, while *Inv* does not. Thus, the reduction to CHCs applies our transformation in a more general setting than Fig. 1b. Furthermore, since u denotes both a schedule and a choice of labels for the existentially quantified traces, the number of A_u predicates depends on the number of labels. To ensure that there are finitely many predicates, we require the transition system to have a finite branching degree (otherwise, the space of possible labels becomes infinite).

Finally, in Sec. 6.2, we extend our approach to handle infinite (or unbounded) branching in the transition system, which can result, for example, from reading an input from an infinite domain. To do so, we introduce another first-order encoding that roughly replaces the infinitely-many concrete choices of transitions by finitely-many abstract choices. Unlike the cases of k-safety and $\forall^*\exists^*$-hyperproperties with finite branching, the resulting encoding is sound but incomplete w.r.t. the game semantics. By applying our transformation, we obtain a sound (albeit incomplete) reduction to CHC-SAT.

3 Background

We use first-order logic to model systems and their properties. Throughout the paper, we fix a background first-order theory $\mathcal{T}$ and denote its signature by Σ.

Transition Systems A (symbolic, labeled) *transition system* is a tuple $TS = (V, a, Init, Tr)$, where V is a vocabulary, *i.e.*, a vector of (logical) variables, each associated with a sort from Σ, denoting state variables; a is a label variable; *Init* is a formula over Σ with free variables V, and Tr is a formula over Σ with free variables $V \cup \{a\} \cup V'$, where V' consists of the primed variants of V.

A state of TS is a valuation to V, and we denote by $\mathbb{S}$ the set of all such valuations; $\mathbb{L}$ is the set of values that a can take, called labels; $\mathbb{S}_0 \subseteq \mathbb{S}$ is the set of initial states, which consists of all valuations that satisfy *Init*, and $\mathbb{R} \subseteq \mathbb{S} \times \mathbb{L} \times \mathbb{S}$ is the transition relation, which consists of the valuations for the composite vocabulary $V \cup \{a\} \cup V'$ that satisfy Tr. For simplicity, we assume that $\mathbb{R}$ is total, *i.e.*, $\forall s \in \mathbb{S}\ \exists \ell \in \mathbb{L}, s' \in \mathbb{S} \cdot (s, \ell, s') \in \mathbb{R}$[3]. We say that TS is *deterministic* when $\forall s \in \mathbb{S}, \ell \in \mathbb{L} \cdot \left|\{s' \,|\, \mathbb{R}(s, a, s')\}\right| = 1$ and that it has finite branching when $\mathbb{L}$ is finite. A trace of TS is an infinite sequence of states $t = s_0, s_1, \ldots$ such that for every $i \geq 0$ there exists $\ell \in \mathbb{L}$ such that $(s_i, \ell, s_{i+1}) \in \mathbb{R}$. We denote by $t[i]$ the i'th state in t. We further denote the set of traces that start from a state s by $\mathbb{T}(s)$, and the set of all traces of TS by $\mathbb{T}$.

Hyperproperties and their specification. We consider a fragment of the relational logic OHyperLTL [6] , which we call $\forall^*\exists^*$-OHyperLTL with formulas of the form:

$$\varphi \;=\; \psi \rightarrow \forall \pi_1 : \xi_1, \ldots, \pi_l : \xi_l \cdot \exists \pi_{l+1} : \xi_{l+1}, \ldots, \pi_k : \xi_k \cdot\ \Box\phi$$

[3] w.l.g.; Tr can always be replaced by $Tr \vee ((\forall a\, \forall V' \cdot \neg Tr) \wedge V' = V)$, which corresponds to adding self loops to states that have no outgoing transition.

where π_i are trace variables whose intended valuations are taken from $\mathbb{T}$; ξ_i are (non-temporal) formulas with free variables V that determine *observation points* along the k traces, where the traces must synchronize; ψ is a pre-condition that is assumed to hold initially; and ϕ needs to globally hold when all traces reach the observation points (which they must synchronize on before moving on). V_j denotes a copy of V where all variables are indexed by j. We refer to the variables in V_j as the state variables of the j'th trace (namely, π_j). When $l = k$, *i.e.*, all quantifiers are universal, φ is a *k-safety* property. A relational pre/post specification, as used in our motivating example, is a special case of a k-safety property where the observable points are the final states (which are augmented with self loops). For example, Fig. 1a presents the $\forall^*\exists^*$-OHyperLTL specification of the motivating example. When $l < k$, the formula also includes existential quantifiers, extending expressiveness to include some hyperliveness properties. An example of a security hyperliveness property that can be expressed in $\forall^*\exists^*$-OHyperLTL is *generalized non-interference (GNI)* [38]. GNI requires that for any two traces π_1 and π_2 there exists a trace π_3 whose high (secret) inputs agree with π_1 and whose low (public) inputs and outputs agree with π_2.

$\forall^*\exists^*$-OHyperLTL formulas are interpreted over transition systems. Intuitively, φ holds in a transition system if from every k initial states that jointly satisfy the pre-condition ψ, for every l traces from the first l states there exist corresponding $k-l$ traces from the remaining $k-l$ states s.t. the composed states of all traces globally satisfy ϕ, when the traces are projected to their observation points. Formally, given a transition system TS and φ as above, we refer to a tuple $(s_1, \ldots, s_k)$ of k states of TS as a *composed state*. A composed state defines a valuation to $V_1 \cup \ldots \cup V_k$, where s_j is the valuation of V_j. A composed state is initial if $s_i \in \mathbb{S}_0$ for every $1 \leq i \leq k$. We say that $TS \models \varphi$ if for every initial composed state $\overline{s} = (s_1, \ldots, s_k)$ such that $\overline{s} \models \psi$ the following holds: for every $t_1, \ldots, t_l \in \mathbb{T}(s_1) \times \cdots \times \mathbb{T}(s_l)$ there exist $t_{l+1}, \ldots, t_k \in \mathbb{T}(s_{l+1}) \times \cdots \times \mathbb{T}(s_k)$ such that $(\!|t_1|\!)_{\xi_1}, \ldots, (\!|t_k|\!)_{\xi_k} \models \Box\phi$, where $(\!|t_i|\!)_{\xi_i}$ is the projection (filtering) of trace t_i to states satisfying ξ_i. The semantics of $\Box\phi$ is that $t'_1, \ldots, t'_k \models \Box\phi$ iff $\forall i \leq \min(|t'_1|, \ldots, |t'_k|) \cdot (t'_1[i], \ldots, t'_k[i]) \models \phi$. Note that the semantics is oblivious to the transition labels since labels are only implicit in traces. Labels become useful in Sec. 6, where we use them to identify transitions along traces.

Remark 1. To simplify the presentation we consider hyperproperties defined w.r.t. a single transition system. The extension to multiple transition systems is straightforward. Similarly, $\Box\phi$ can be generalized to any temporal safety property via the standard automata-theoretic approach to model checking.

Constrained Horn Clauses (CHCs) are defined over a signature Σ' that extends Σ with a set $\mathcal{P}$ of (uninterpreted) predicates. Symbols in Σ are called *interpreted*, while the predicates in $\mathcal{P}$ are *uninterpreted* (sometimes called *unknown*). First-order formulas over Σ are called *constraints*. A CHC is a first-order formula of the form $\forall\mathcal{X} \cdot \bigwedge_i P_i(\mathcal{X}_i) \wedge \varphi(\mathcal{X}) \rightarrow H(\mathcal{X}_H)$ where $\mathcal{X}$ is a vector of (logical) variables; $P_i \in \mathcal{P}$ (not necessarily distinct, *i.e.*, it is possible that $P_{i_1} = P_{i_2}$ for

$$\alpha(\mathcal{V}) \to Inv(\mathcal{V})$$
$$Inv(\mathcal{V}) \wedge \beta(\mathcal{V}) \to \bot$$
$$\boxed{\forall} \quad Inv(\mathcal{V}) \wedge A_u(\mathcal{V},\mathcal{W}) \wedge \gamma_u(\mathcal{V},\mathcal{W}) \to \bot$$
$$\boxed{\forall} \; Inv(\mathcal{V}) \wedge A_u(\mathcal{V},\mathcal{W}) \wedge \delta_u(\mathcal{V},\mathcal{V}',\mathcal{W}) \to Inv(\mathcal{V}')$$
$$Inv(\mathcal{V}) \to \bigvee_{u\in U} A_u(\mathcal{V},\mathcal{W})$$

($\boxed{\forall} = \forall u \in U$) (a)

$$\bigwedge_{u\in U} D_u(\mathcal{V},\mathcal{W}) \wedge \alpha(\mathcal{V}) \to \bot$$
$$\boxed{\forall} \quad \beta(\mathcal{V}) \to D_u(\mathcal{V},\mathcal{W})$$
$$\boxed{\forall} \quad \gamma_u(\mathcal{V},\mathcal{W}) \to D_u(\mathcal{V},\mathcal{W})$$
$$\boxed{\forall} \; \bigwedge_{u'\in U} D_{u'}(\mathcal{V}',\mathcal{W}') \wedge \delta_u(\mathcal{V},\mathcal{V}',\mathcal{W}) \to D_u(\mathcal{V},\mathcal{W})$$

(b)

Fig. 3: Formula scheme before (a) and after (b) the transformation.

$i_1 \neq i_2$); H is either $\bot$ or a predicate from $\mathcal{P}$; $\mathcal{X}_i, \mathcal{X}_H \subseteq \mathcal{X}$; and φ is a constraint. The universal quantification over $\mathcal{X}$ is often omitted.

A set of CHCs (or, more generally, first-order formulas) is *satisfiable* (modulo $\mathcal{T}$) if it has satisfying model M such that the projection of M onto Σ is a model of $\mathcal{T}$. A *solution* to a set of CHCs maps every predicate in $\mathcal{P}$ to a formula over Σ that defines it such that substituting all occurrences of the predicates by their definitions results in formulas that are valid modulo $\mathcal{T}$. If a set of CHCs has a solution then it is satisfiable. However, the converse may not hold due to the limited expressive power of first-order formulas.

4 General Transformation to CHCs

In this section we describe a satsifiability-preserving transformation that lets us convert a set of formulas, which adheres to a specific FOL scheme, to an equi-satisfiable set of CHCs. An extended version, with step-by-step details of the transformation, appears in [34]. Later we show how verification of a $\forall^*\exists^*$-OHyperLTL property can be captured by a set of formulas of the aforementioned scheme, where this transformation allows us to then reason about the correctness of the $\forall^*\exists^*$-OHyperLTL property by deciding the satisfiability of the CHCs.

Consider the scheme in Fig. 3a for a set of formulas over a signature Σ' that extends the signature Σ of the background theory by unknown predicates Inv and $\{A_u\}_{u\in U}$, for some finite set U. $\mathcal{V}, \mathcal{V}', \mathcal{W}$ denote disjoint vocabularies, *i.e.*, vectors of (logical) variables that are implicitly universally quantified. A row prefixed by $\boxed{\forall}$ indicates $|U|$ formulas, where u is substituted by all corresponding values from U. $\alpha, \beta, \gamma_u, \delta_u$ designate *constraints* (no occurrence of Inv or A_u).

At a high level, formulas 1 and 4 in Fig. 3a use Inv to capture an inductive invariant of the "states" (valuations to $\mathcal{V}$) reachable from α by "transitions" of δ_u, restricted according to a choice $u \in U$ of an "arbiter" $\{A_u\}_u$. Formula 2 establishes the fact that the reachable states are disjoint from some "bad states" β. Formulas 3 allow to enforce that the arbiter meets certain requirements, and formula 5 ensures that the arbiter makes a choice for every "state" in Inv.

Example 1. For our running example, we have $\mathcal{V} = \langle V_1, V_2\rangle = \langle a_1, b_1, c_1, a_2, b_2, c_2\rangle$, $\mathcal{V}' = \langle V_1', V_2'\rangle = \langle a_1', b_1', c_1', a_2', b_2', c_2'\rangle$, and $\mathcal{W} = \langle\rangle$ (The extra vocabulary $\mathcal{W}$ will

come into use later in the paper). U is the set of arbitration choices $\{\{1\},\{2\},\{1,2\}\}$, and the corresponding completion of the constraint holes $\alpha, \beta, \gamma_u, \delta_u$ is easily discernible. (Note that a constraint on the right of $\rightarrow$ corresponds to its negation on the left.)

Note that the last formula in Fig. 3a is not a CHC since its head is a disjunction of unknown predicates. To remedy this shortcoming, we transform the formulas in Fig. 3a into the set of CHCs in Fig. 3b. The CHCs obtained for the running example are included in the extended version of the paper [34]. The transformation ensures:

Theorem 1. *The set of formulas in Fig. 3a is equi-satisfiable to the system of CHCs in Fig. 3b. Furthermore, there is an efficient translation of models of the former to models of the latter, and vice versa.*

Proof. The extended version of the paper [34] includes a stepwise transformation that shows how the CHCs in Fig. 3b are obtained from the formulas in Fig. 3a, where each step preserves equi-satisfiability and models. Here, due to space constraints, we only describe the final translation between models, which we have verified with Z3:

Given $Inv, A_u \models$ [Fig. 3a]	Given $D_u \models$ [Fig. 3b]
$D_u(\mathcal{V},\mathcal{W}) \triangleq \neg(Inv(\mathcal{V}) \wedge A_u(\mathcal{V},\mathcal{W}))$	$Inv(\mathcal{V}) \triangleq \forall\mathcal{W} \cdot \bigvee_{u\in U} \neg D_u(\mathcal{V},\mathcal{W})$ $A_u(\mathcal{V},\mathcal{W}) \triangleq \neg D_u(\mathcal{V},\mathcal{W})$

5 Encoding *k*-Safety Verification as CHCs

In this section we address the problem of verifying k-safety properties via a CHC encoding. To this end, we start with a natural, non-Horn, encoding of the problem, as described in the previous section and previous works [41,45,8], and apply our transformation to obtain an equi-satisfiable system of CHCs.

Consider the k-safty formula: $\varphi = \psi \rightarrow \forall \pi_1 : \xi_1, \ldots, \pi_k : \xi_k \cdot \Box\phi$

This formula holds in a transition system TS if, starting from initial composed states that satisfy the pre-condition ψ, the observable states along every tuple of k traces satisfy ϕ, when the observable states are reached synchronously. Note that a pre-post specification, as used in our motivating example, is a special case of such a formula where the observable states are the final states. Verifying φ corresponds to finding (1) an alignment of the traces that synchronizes the observation points defined by $\xi_1, \ldots, \xi_k$, and (2) an inductive invariant that establishes that ϕ holds whenever $\xi_1, \ldots, \xi_k$ hold. Note that the invariant needs to be inductive along the aligned traces, including intermediate states between observable points. As different alignments give rise to different inductive invariants, it is desirable to find both of them simultaneously [41].

As before, we model the alignment using an arbiter that schedules a subset $\varnothing \neq M \subseteq \{1, \ldots, k\}$ of the traces to make a step based on the current composed state $s_1 \cdots s_k$. The arbiter may be nondeterministic, but it must choose at least

$$\bigwedge_i Init(V_i) \wedge \psi(\mathcal{V}) \rightarrow Inv(\mathcal{V})$$
$$Inv(\mathcal{V}) \wedge Bad(\mathcal{V}) \rightarrow \bot$$
$$\boxed{\forall}\ Inv(\mathcal{V}) \wedge A_M(\mathcal{V}) \wedge \neg valid_M(\mathcal{V}) \rightarrow \bot$$
$$\boxed{\forall}\ Inv(\mathcal{V}) \wedge A_M(\mathcal{V}) \wedge \delta_M(\mathcal{V}, \mathcal{V}') \rightarrow Inv(\mathcal{V}')$$
$$Inv(\mathcal{V}) \rightarrow \bigvee_M A_M(\mathcal{V})$$
$$(\boxed{\forall} = \forall M)$$

(a)

$$\bigwedge_M D_M(\mathcal{V}) \wedge \bigwedge_i Init(V_i) \wedge \psi(\mathcal{V}) \rightarrow \bot$$
$$\boxed{\forall}\ Bad(\mathcal{V}) \rightarrow D_M(\mathcal{V})$$
$$\boxed{\forall}\ \neg valid_M(\mathcal{V}) \rightarrow D_M(\mathcal{V})$$
$$\boxed{\forall}\ \bigwedge_{M'} D_{M'}(\mathcal{V}') \wedge \delta_M(\mathcal{V}, \mathcal{V}') \rightarrow D_M(\mathcal{V})$$

(b)

Fig. 4: k-safety formula scheme before (a) and after (b) the transformation.

one set M. Furthermore, the arbiter must respect the synchronization of the observation points: it must not let a trace proceed beyond its observation point before the other traces reached theirs. This motivates the following definition.

Definition 1 (valid schedules). *M is a* valid schedule *for a composed state $s_1 \cdots s_k$ if either of the following two conditions holds:*

1. $\forall i \in M \cdot s_i \not\models \xi_i$
2. $\forall i \in M \cdot s_i \models \xi_i$ *and* $M = \{1, ..., k\}$.

Intuitively, the observation points act as a "barrier". All traces must reach the observation point before any of them can progress past it; and when they do, they do it simultaneously.[4]

To reason about composed states, we define a vocabulary $\mathcal{V} = V_1 \cup \ldots \cup V_k$ that consists of the set of state variables of all traces. We encode the arbiter using a family of unknown predicates $\{A_M(\mathcal{V})\}_M$ for every $\varnothing \neq M \subseteq \{1, \ldots, k\}$ and the inductive invariant using an unknown predicate $Inv(\mathcal{V})$. We express the situation where all traces reach an observable state but ϕ does not hold using the constraint: $Bad(\mathcal{V}) \triangleq \bigwedge_i \xi_i(V_i) \wedge \neg\phi(\mathcal{V})$. The joint steps of the traces as determined by the schedule M are given by the following constraint:

$$\Delta_M(\mathcal{V}, \mathcal{V}', a_1, \ldots, a_k) \triangleq \bigwedge_{i \in M} Tr(V_i, a_i, V_i') \wedge \bigwedge_{i \notin M} V_i = V_i'$$
$$\delta_M(\mathcal{V}, \mathcal{V}') \triangleq \exists a_1, \ldots, a_k \cdot \Delta_M(\mathcal{V}, \mathcal{V}', a_1, \ldots, a_k)$$

Note that the label variables are existentially quantified[5], indicating that any labeled transition can be used. The definition of a valid schedule is captured by:

$$valid_M(\mathcal{V}) \triangleq \begin{cases} \bigwedge_{i \in M} \neg\xi_i(V_i) & M \neq \{1, \ldots, k\} \\ \left(\bigwedge_{i \in M} \neg\xi_i(V_i)\right) \vee \left(\bigwedge_{i \in M} \xi_i(V_i)\right) & M = \{1, \ldots, k\} \end{cases} \quad (1)$$

Fig. 4a formalizes the joint requirements of the arbiter and the inductive invariant that ensures that φ holds. The following theorem summarizes the soundness of the encoding, which is a slight generalization of the encoding in [41] (where only pre/post specifications are considered):

[4] The requirement that all traces leave the observation point in tandem saves the need to record which of them already made a step since the last observation point.

[5] Since δ_M appears on the left-hand side of an implication, existential quantifiers can be pushed outside as universal quantifiers, resulting in quantifier-free bodies.

```
1  sum = 0;
2  b = *;
3  if (b > 0) {
4    i = 0;
5    while (i < n - 1) {
6      sum = sum + A[i];
7      i++;
8    }
9  }
10 else {
11   i = 1;
12   while (i < n) {
13     y = *;
14     sum = sum + A[i] + y;
15     i++;
16   }
17 }
```

$$(A_1 = A_2 \wedge n_1 = n_2) \rightarrow \forall \pi_1 : pc = 5\ \exists \pi_2 : pc = 5 \vee pc = 12 \cdot \square(b_2 \leq 0 \wedge sum_1 = sum_2)$$

Fig. 5: Example for a $\forall\exists$ hyperproperty.

Theorem 2. *The set of formulas in Fig. 4a is satisfiable iff* $TS \models \varphi$.

Example 2. Applying the scheme of Fig. 4a to the program and $\forall^*\exists^*$-OHyperLTL specification of the 2-safety property from Fig. 1a results in Fig. 1b, except for moving constraints to the right hand side of the implication when it assists readability. Note that in this example, the observation points ξ_i of both traces correspond to the condition for exiting the loop (which is the negated loop condition). As a result $valid_{\{i\}} \triangleq a_i < b_i$ for $i \in \{1, 2\}$ and $valid_{\{1,2\}} \triangleq (a_1 < b_1 \wedge a_2 < b_2) \vee (\neg(a_1 < b_1) \wedge \neg(a_2 < b_2))$.

The set of formulas in Fig. 4a fits the general scheme of Fig. 3a; Thus, it is amenable to our general satisfiability-preserving transformation, the CHCs in Fig. 4(b). Since the transformation is satisfiability preserving, we obtain the following as a corollary of Thm. 1 and 2:

Corollary 1. *The system of CHCs in* Fig. 4b *is satisfiable iff* $TS \models \varphi$.

Where $A_M(\mathcal{V})$ in Fig. 4a describes the states where choosing schedule M leads to successful verification with *Inv* as an inductive invariant, $D_M(\mathcal{V})$ in Fig. 4b can be understood as describing states where choosing M would *prevent* the verification from going through in the sense that no inductive invariant would exist. In other words, these states are "doomed" if M is chosen, hence the choice of notation. If the set of CHCs in Fig. 4b is satisfiable, it proves that initial states that satisfy the pre-condition are not doomed. This intuition can be interpreted in a dual manner: if the initial states are not doomed, then there exists an alignment for which a safe inductive invariant exist.

6 Encoding $\forall^*\exists^*$ Hyperproperties as CHCs

In this section we consider the more general case of $\forall^*\exists^*$-OHyperLTL specifications. Throughout the section, *TS* is a transition system, and we fix a formula:

$$\varphi \ = \ \psi \rightarrow \forall \pi_1 : \xi_1, \ldots, \pi_l : \xi_l \cdot \exists \pi_{l+1} : \xi_{l+1}, \ldots, \pi_k : \xi_k \cdot \ \square\phi$$

In order to encode the problem of deciding if $TS \models \varphi$ as a satisfiability problem, we follow [8], and consider a *game semantics*, which is natural due to the alternation of quantifiers. The $\forall$ and $\exists$ quantifiers are "demonic" and "angelic", thus controlled by the falsifier and the verifier, respecitvely.

In the following, we introduce the game semantics of [8] for $\forall^*\exists^*$-OHyperLTL. We then encode truth of φ in TS under the game semantics as a satisfiability problem, and use the transformation from Sec. 4 to obtain a system of CHCs that is satisfiable iff TS satisfies φ according to the game semantics.

Example 3. To illustrate the game semantics, we use the example in Fig. 5, which accompanies this section. The presented program computes the sum of an array slice, nondeterministically choosing between the slice $A[0..n-2]$ and $A[1..n-1]$. For the second variant, an arbitrary integer can be added to each summand. This allows the program to fulfill the specification at the bottom, which requires that for every execution there is a corresponding execution of the second variant (where $b_2 \leq 0$) such that the sums at lines 5 and 12 align at every iteration. The specification is valid because y at line 13 can always be chosen to compensate for the deviation due to the index i not being the same.

Considering the game semantics, the falsifier first has to choose a value for b, which can be either positive or nonpositive. If it is nonpositive, then the verifier wins the game vacuously because $\xi_1 \triangleq (pc = 5)$ is never reached. If the choice is positive, then the verifier must choose nonpositive to satisfy $b_2 \leq 0$ from the specification. In subsequent steps, the verifier must select a scheduling that will align $pc_1 = 5$ and $pc = 12$ at every iteration, and select a value for y such that after both assignments (lines 6 and 14) $sum_1 = sum_2$ is satisfied. When following these choices, the verifier manages to satisfy $sum_1 = sum_2$ at all observation points, which gives it a winning play.

Safety games are played between a *verifier*, whose goal is to avoid *bad* states, and a *falsifier* who tries to reach a bad state. Formally, the game is a tuple $\mathcal{G} = (VS, FS, S_0, \delta_V, \delta_F, B)$ where VS are *verifier states*, in which the verifier moves, and FS are *falsifier states*, in which the falsifier moves, and $VS \cap FS = \varnothing$. The *game states* are $S = VS \cup FS$. $S_0 \subseteq S$ is a set of initial states, and $B \subseteq S$ is a set of bad states. $\delta_V \subseteq VS \times S$ defines the possible moves of the verifier and $\delta_F \subseteq FS \times S$—of the falsifier. It is assumed that δ_V, δ_F are total *i.e.*, there is at least one move for each player from every state. A *play* is a sequence of game states $\sigma_0, \sigma_1, \ldots$ such that $\sigma_0 \in S_0$, and for every $i \geq 0$, $(\sigma_i, \sigma_{i+1}) \in \delta_V \cup \delta_F$. The play is winning for the verifier if it is infinite and $\sigma_i \notin B$ for every $i \geq 0$. A (memoryless) strategy for the verifier is a function $\chi : VS \to S$ such that $(\sigma, \chi(\sigma)) \in \delta_V$ for every $\sigma \in VS$. χ is a *winning strategy* for the verifier if all the plays in which the verifier moves according to χ are winning for the verifier.

Game semantics for $\forall^*\exists^*$-OHyperLTL Let φ be as above. The game that captures the semantics of φ is defined with respect to a deterministic labeled transition system $TS = (V, a, Init, Tr)$. (We can always determinize TS by extending the set of labels without affecting the semantics; this step may introduce infinitely many labels, which do not require any special treatment in the definition of the game, but whose CHC encoding will be addressed in Sec. 6.2.)

The game for φ and TS proceeds in rounds, where in each round the falsifier makes a move and the verifier responds. The falsifier states are composed states

(of k traces), and the verifier states augment them with a record of the falsifier's last move. The bad states are falsifier states where all traces are in their observation points but ϕ does not hold. The falsifier is responsible for choosing the transitions that define the $\forall$ traces $t_{1..l}$ assigned to $\pi_{1..l}$. The verifier responds by choosing the transitions of the $\exists$ traces $t_{l+1..k}$ assigned to $\pi_{l+1..k}$. Here the labels of the transitions come into play: the players specify the transitions of choice by picking a label $\ell \in \mathbb{L}$ for each trace. (Since *TS* is deterministic, transitions are uniquely identified by labels.) The traces then need to be aligned s.t. they synchronize on their observation points defined by ξ_i. The alignment does not affect the winner of the play, as long as it is a valid alignment. However, as in the case of k-safety, the alignment is instrumental for obtaining a winning strategy that has a simple description. As a result, the choice of the (valid) alignment is also left to the verifier. Altogether, a move of the falsifier consists of picking labels $\ell_l, \ldots, \ell_l \in \mathbb{L}$ for the $\forall$ trace variables; a move of the verifier consists of picking a valid subset $\varnothing \neq M \subseteq \{1, ..., k\}$ of the traces to progress (as in Sec. 5) and also labels $\ell_{l+1}, \ldots, \ell_k \in \mathbb{L}$ for the $\exists$ trace variables, and proceeding to the resulting composed state.[6] In this manner, the verifier iteratively "reads off" the states of $t_{1..l}$, properly aligned, and generates the traces $t_{l+1..k}$, while avoiding the bad states. If the verifier can do so indefinitely, then this proves that φ holds.

Formally, the components of the game are as follows (here, M represents a valid schedule according to Definition 1):

$$FS = \mathbb{S}^k \qquad VS = \mathbb{S}^k \times \mathbb{L}^l \qquad S_0 = \{\overline{s} \in \mathbb{S}_0^k \mid \overline{s} \models \psi\}$$
$$B = \{\overline{s} \in FS \mid \overline{s} \not\models \phi \text{ and } s_i \models \xi_i \text{ for every } 1 \leq i \leq k\}$$
$$\delta_F = \{(\overline{s}, (\overline{s}, \overline{\ell}^\forall)) \mid \overline{s} \in FS,\ \overline{\ell}^\forall \in \mathbb{L}^l\} \quad \delta_V = \{((\overline{s}, \overline{\ell}^\forall), \overline{s}') \mid \overline{s} \overset{M,\overline{\ell}}{\rightsquigarrow} \overline{s}' \text{ for } \overline{\ell}^\exists \in \mathbb{L}^{k-l}\}$$

The notation $\overline{s} \overset{M,\overline{\ell}}{\rightsquigarrow} \overline{s}'$ indicates that $\overline{s}'$ is obtained from $\overline{s}$ by taking the transition with label ℓ_i from s_i whenever $i \in M$, and stuttering otherwise, where $\overline{\ell} = \langle \ell_1, \ldots, \ell_k \rangle$. We refer to it as a transition of the composed system according to schedule M labeled $\overline{\ell}$. The labels are split into $\overline{\ell}^\forall = \langle \ell_1, .., \ell_l \rangle$ and $\overline{\ell}^\exists = \langle \ell_{l+1}, .., \ell_k \rangle$. Formally, $\overline{s} \overset{M,\overline{\ell}}{\rightsquigarrow} \overline{s}' \iff \bigwedge_{i \in M} \mathbb{R}(s_i, \ell_i, s_i') \wedge \bigwedge_{i \notin M} s_i = s_i'$.

Example 4. In the example of Fig. 5, the labels of transitions are integer values that reflect the choice of $*$ at lines 2 and 13 (and have no effect on other states). The verifier and falsifier specify their moves using these labels. For example, in order to ensure that $sum_1 = sum_2$ is satisfied at every iteration, the verifier selects a transition label $\ell = A[i-1] - A[i]$ in line 13, which sets the value of y accordingly; after both assignments at lines 6 and 14, $sum_1 = sum_2$ holds.

The game semantics of $\forall^*\exists^*$-OHyperLTL is based on the winner in the verification game:

Definition 2 (Game Semantics for $\forall^*\exists^*$-OHyperLTL [8]). *Let TS be a transition system and φ a $\forall^*\exists^*$-OHyperLTL formula. TS* satisfies φ according

[6] In [8], steps of the verifier are split to two. Our definition is more precise in the sense that a winning strategy in the game of [8] implies a winning strategy in our game.

to the game semantics, *denoted* $TS \models_{\mathcal{G}} \varphi$, *if the verifier has a winning strategy in the verification game* $\mathcal{G}_{TS,\varphi}$.

Theorem 3 (as shown in [8]). *If* $TS \models_{\mathcal{G}} \varphi$ *then* $TS \models \varphi$.

6.1 CHC Encoding of the Game with Finite Branching

To encode the verification game for φ and TS, we introduce unknown predicates $\{A_u\}_{u \in U}$ that describe the strategy of the verifier as well as an unknown predicate Inv that encodes an inductive invariant that ensures that the strategy is winning. We first consider the case where the set of labels $\mathbb{L}$ is finite, i.e., TS has a finite branching. This makes it possible to define U as the set of all possible concrete choices of the verifier and introduce a predicate A_u per every possible choice of the verifier. To do so, we define $U = \mathbb{M} \times \mathbb{L}^{k-l}$, where $\mathbb{M} = \mathcal{P}(\{1, \ldots, k\}) \setminus \{\varnothing\}$ is the set of possible schedules, and $\mathbb{L}^{k-l}$ are the choice labels for constructing the traces assigned to $\{\pi_i\}_{i=l+1..k}$. Note that U is finite in this case. For each $u = \langle M, \overline{\ell}^{\exists} \rangle \in U$, the predicate A_u describes the verifier states in which the verifier chooses u for its move. Recall that verifier states consist of both the previous state of the verifier, captured by the composed state vocabulary $\mathcal{V}$ defined as before, and the last move of the falsifier, captured by label variables $\langle a_1, \ldots, a_l \rangle$. We denote $\mathcal{L}^{\forall} = \langle a_1, \ldots, a_l \rangle$, $\mathcal{L}^{\exists} = \langle a_{l+1}, \ldots, a_k \rangle$ and $\mathcal{L} = \mathcal{L}^{\forall} \cup \mathcal{L}^{\exists} = \langle a_1, \ldots, a_k \rangle$. Then, the A_u predicates are defined over $\mathcal{V} \cup \mathcal{L}^{\forall}$. The Inv predicate is defined over $\mathcal{V}$ only, as it describes a set of falsifier states.

The formulas in Fig. 6a formalize the requirements that ensure that $\{A_u\}_u$ defines a winning strategy for the verifier, while accounting for the alternating choices of the falsifier ($\overline{\ell}^{\forall}$) and verifier ($\langle M, \overline{\ell}^{\exists} \rangle$) in every round, where

$$\Delta_M(\mathcal{V}, \mathcal{V}', \mathcal{L}) \quad = \bigwedge_{i \in M} Tr(V_i, a_i, V_i') \wedge \bigwedge_{i \notin M} V_i = V_i'$$

$$\delta_{M,\overline{\ell}^{\exists}}(\mathcal{V}, \mathcal{V}', \mathcal{L}^{\forall}) = \Delta_M(\mathcal{V}, \mathcal{V}', \mathcal{L})\left[\mathcal{L}^{\exists} \mapsto \overline{\ell}'\right] \qquad Bad(\mathcal{V}) \;=\; \bigwedge_i \xi_i(V_i) \wedge \neg\phi(\mathcal{V})$$

Δ_M is the formula expression of $\overset{M,\overline{\ell}}{\rightsquigarrow}$ from above. That is, $\overline{s}, \overline{s}', \overline{\ell}$ (valuations to $\mathcal{V}, \mathcal{V}', \mathcal{L}$) satisfy Δ_M if the composed system according to M has a transition from $\overline{s}$ to $\overline{s}'$ labeled $\overline{\ell}$. $\delta_{M,\overline{\ell}^{\exists}}$ is then the projection of Δ_M to a concrete choice of labels $\overline{\ell}^{\exists}$ for the existentially quantified traces; the labels for the universals, captured by $\mathcal{L}^{\forall}$, remain free.

Theorem 4. *The set of formulas in Fig. 6a is satisfiable iff* $TS \models_{\mathcal{G}} \varphi$.

Proof. A solution for Fig. 6a induces a winning strategy χ for the verifier in the game for φ and TS: $\chi(\overline{s}, \overline{\ell}^{\forall}) = \overline{s}'$ for $\overline{s} \models Inv$, where $\overline{s}'$ is reached by choosing $\langle M, \overline{\ell}^{\exists} \rangle$ (*i.e.*, $\overline{s}, \overline{s}', \overline{\ell}^{\forall} \models \delta_{M,\overline{\ell}^{\exists}}$) such that $\overline{s}, \overline{\ell}^{\forall} \models A_{M,\overline{\ell}^{\exists}}$; such $\overline{s}'$ must exist because the last formula states that there must always be a choice for the verifier in falsifier states that satisfy Inv. For $\overline{s} \not\models Inv$, $\chi(\overline{s}, \overline{\ell}^{\forall})$ is defined arbitrarily. In the other direction, given a winning strategy for the verifier, we define the interpretation of Inv to be its winning region and the interpretation of $A_{M,\overline{\ell}^{\exists}}$ to consist of the falsifier states $(\overline{s}, \overline{\ell}^{\forall})$ where the strategy chooses $\overline{s}'$ such that $\overline{s}, \overline{\ell}^{\forall} \models A_{M,\overline{\ell}^{\exists}}$.

$$\begin{aligned}
\textstyle\bigwedge_i \mathit{Init}(V_i) \wedge \psi(\mathcal{V}) &\to \mathit{Inv}(\mathcal{V}) \\
\mathit{Inv}(\mathcal{V}) \wedge \mathit{Bad}(\mathcal{V}) &\to \bot \\
\boxed{\forall} \quad \mathit{Inv}(\mathcal{V}) \wedge A_{M,\overline{\ell}^{\exists}}(\mathcal{V}, \mathcal{L}^{\forall}) \wedge \neg \mathit{valid}_M(\mathcal{V}) &\to \bot \\
\boxed{\forall} \; \mathit{Inv}(\mathcal{V}) \wedge A_{M,\overline{\ell}^{\exists}}(\mathcal{V}, \mathcal{L}^{\forall}) \wedge \delta_{M,\overline{\ell}^{\exists}}(\mathcal{V}, \mathcal{V}', \mathcal{L}^{\forall}) &\to \mathit{Inv}(\mathcal{V}') \\
\mathit{Inv}(\mathcal{V}) &\to \bigvee_{\langle M,\overline{\ell}^{\exists}\rangle \in U} A_{M,\overline{\ell}^{\exists}}(\mathcal{V}, \mathcal{L}^{\forall})
\end{aligned}$$

(a)

$$\begin{aligned}
\bigwedge_{\langle M,\overline{\ell}^{\exists}\rangle \in U} D_{M,\overline{\ell}^{\exists}}(\mathcal{V}) \wedge \bigwedge_i \mathit{Init}(V_i) \wedge \psi(\mathcal{V}) &\to \bot \\
\boxed{\forall} \quad \mathit{Bad}(\mathcal{V}) &\to D_{M,\overline{\ell}^{\exists}}(\mathcal{V}) \\
\boxed{\forall} \quad \neg \mathit{valid}_M(\mathcal{V}) &\to D_{M,\overline{\ell}^{\exists}}(\mathcal{V}) \\
\boxed{\forall} \quad \bigwedge_{\langle M',\overline{\ell}'^{\exists}\rangle \in U} D_{M',\overline{\ell}'^{\exists}}(\mathcal{V}') \wedge \delta_{M,\overline{\ell}^{\exists}}(\mathcal{V}, \mathcal{V}') &\to D_{M,\overline{\ell}^{\exists}}(\mathcal{V})
\end{aligned}$$

(b)

Fig. 6: A game formula scheme before (a) and after (b) the transformation, where $\boxed{\forall} = \forall \langle M, \overline{\ell}^{\exists}\rangle \in U$.

Remark 2. For k-safety properties, the encoding in Fig. 6a, based on the game semantics, is equivalent to the encoding in Fig. 4a (Sec. 5). In particular, in this case, the set $\mathcal{L}^{\exists}$ is empty, which means that $\overline{\ell}^{\exists} = \langle\rangle$, resulting in a game with finite branching, namely only the choices of the schedule M. Note that for such properties, the benefits of the game semantics are less obvious since if $TS \models \varphi$, then *every* strategy is winning for the verifier.

Encoding safety games in general The game encoding in Fig. 6a and Thm. 4 are stated here for the specific safety games corresponding to $\forall^*\exists^*$-OHyperLTL verification in order to avoid additional notational burden. However, the result is applicable to a more general class of safety games where the moves of the players are organized in rounds, each of which comprises of a move of the falsifier, followed by a move of the verifier. Furthermore, the states of the verifier are "intermediate states" defined as $VS = FS \times \Omega$, where Ω is a set of *auxiliary states* used to record the last falsifier move. The initial and bad states are falsifier states. The verifier moves to a new state according to the previous state together with the auxiliary state, while the falsifier is only allowed to choose the auxiliary part of the state. Therefore, $\delta_F \subseteq \{\langle \hat{s}, \langle \hat{s}, \omega\rangle\rangle \mid \hat{s} \in FS, \omega \in \Omega\}$. The encoding extends to such games, where $\mathit{Init}(V_i) \wedge \psi(\mathcal{V})$ is replaced by an encoding of S_0; Bad is replaced by an encoding of B; $\delta_{M,\overline{\ell}^{\exists}}(\mathcal{V}, \mathcal{V}', \mathcal{L}^{\forall})$ is replaced by an encoding of $\delta_V \circ \delta_F$ as a formula where the falsifier state variables and the choices of the falsifier are free, and $\mathit{valid}_M(\mathcal{V})$ is replaced by a guard encoded over the same free variables that ensures that the verifier step is applicable. Accordingly, our subsequent results (including the CHC encoding) extend to any such game.

Applying our transformation to the formulas in Fig. 6a results in the CHCs in Fig. 6b. Intuitively, $A_{M,\overline{\ell}^\exists}$ describe the winning strategy for the verifier: for "safe" states, represented by *Inv*, and given a move made by the falsifier, if the verifier chooses to move according to $\langle M, \overline{\ell}^\exists \rangle$, then it stays in the "safe" region. In contrast, $D_{M,\overline{\ell}^\exists}$ represents "doomed" states. Namely, if the verifier chooses to move according to $\langle M, \overline{\ell}^\exists \rangle$ from a state in $D_{M,\overline{\ell}^\exists}$, then the falsifier can force reaching a bad state for every choice of the verifier in the next steps of the game.

Corollary 2. *The set of CHCs in Fig. 6b is satisfiable iff* $TS \models_{\mathcal{G}} \varphi$.

Example 5. The example in Fig. 5 fits the case of finite branching if we assume that the integer values in the array A and those of *sum* and t are bounded modulo 2^m, and so are the labels $\mathbb{L}$. This means that the falsifier has 2^m possible steps at each game state, and the verifier has $3 \cdot 2^m$ (3 is the number of possible schedules out of $\{1, 2\}$). In the next subsection we explain how to encode the problem when the integers are considered to be unbounded.

6.2 CHC Encoding of the Game with Infinite Branching

The set of formulas in Fig. 6a, and the corresponding system of CHCs in Fig. 6b is well defined when the set U is finite. However, if $\mathbb{L}$ is infinite, so is U. In this case, instead of using $\mathbb{L}^{k-l}$ to specify the traces chosen by the verifier, we define a finite, abstract set of composed labels, denoted $\mathbb{L}^\sharp$, to be used by the verifier (the falsifier will continue to use the concrete labels to specify his transitions of choice). Each abstract label in $\mathbb{L}^\sharp$ is a relational predicate p with free variables $\mathcal{V}$ (the composed vocabulary) that relates the states of different traces. Thus, the vector of individual existential choices $\overline{\ell}^\exists$ of the verifier is now replaced with a *single* choice of a (relational) predicate $p \in \mathbb{L}^\sharp$ over all the copies. In contrast to the use of concrete labels to specify the (unique) next transition for each trace individually, an abstract label $p \in \mathbb{L}^\sharp$ determines the next transitions for the $\exists$ traces by relating their post-states to the rest of the composed post-state.

Specifically, given a set of labels $\overline{\ell}^\forall$ for the $\forall$ traces and a schedule M, a predicate $p \in \mathbb{L}^\sharp$ is used as a *restriction* (inspired by the homonymous concept from [8]) of the transitions of the composed system according to schedule M with $\forall$-choices $\overline{\ell}$, restricting the set of aforementioned transitions to those where the composed post-state satisfies p.

Example 6. In Fig. 5, at line 13, a nondeterministic integer value is assigned to variable t. Since the set of integers is infinite, assigning a unique label ℓ to each integer results in an infinite set $\mathbb{L}$. To specify the choices of the verifier, we therefore define a finite set of abstract labels. An example of such a set is $\mathbb{L}^\sharp = \{sum_1 = sum_2, sum_1 = y_2, sum_1 < y_2, sum_1 = sum_2 + A_2[i_2] + y_2\}$.The restriction $sum_1 = sum_2$ can result in an empty set of transitions (we will return to this point later in the section); but the restrictions $sum_1 = y_2$, $sum_1 < y_2$ and $sum_1 = sum_2 + A_2[i_2] + y_2$ always define a nonempty set of transitions when $pc_2 = 13$ and when a schedule $\{2\} \subseteq M$ is chosen: those transitions that choose a

value for y_2 such that the predicate holds after the transition; there is always at least one such value. In fact, for $sum_1 = y_2$ and $sum_1 = sum_2 + A_2[i_2] + y_2$ there is exactly one such value, while for $sum_1 < y_2$, the set of values (transitions) is infinite. Note that there are transitions that are not selected by any restriction (those that assign to y_2 a value such that none of the predicates hold).

Thus, the abstract labels define a space of *underapproximations* of the transitions of the composed system. This is an underapproximation since some (combinations of) individual transitions of TS may not be allowed by any $p \in \mathbb{L}^\sharp$.

The verifier uses $p \in \mathbb{L}^\sharp$ to specify the transitions of the traces assigned to the existentially quantified variables $\pi_{l+1..k}$. We then require that *all* of the composed post-states reached by the verifier's choice $\langle M, p\rangle$ are winning for the verifier. This amounts to proving that *all* restricted traces satisfy $\Box\phi$, which would mean that there *exist* traces that do, *as long as the restrictions do not lead to an empty set of traces*. Therefore, to ensure soundness of the encoding, we require that the restrictions be nonempty. Nonemptiness of the restrictions also ensures that the choices of the falsifier are never restricted, since the choices of the falsifier are always singletons (based on the concrete labels).

Rather than limiting the set of predicates used as abstract labels, we ensure nonemptiness by applying the restrictions only when the resulting set of transitions is nonempty; otherwise, the full set of transitions is considered. Technically, this is accounted for by special considerations in the construction of the CHC encoding, as detailed below.

CHC encoding We adapt the formulas in Fig. 6a to the case of abstract labels. We define $U = \mathbb{M} \times \mathbb{L}^\sharp$. The formulas from Fig. 6a carry over, except that the definition of $\delta_{M,\overline{\ell}}$ from the finite branching case is now replaced with $\delta_{M,p}$, which captures the transitions according to the abstract labels, as defined below.

For a schedule $\varnothing \neq M \subseteq \{1..k\}$ and $p \in \mathbb{L}^\sharp$, we define $allowed_{M,p}$, a formula that is satisfied by $\overline{s}, \overline{\ell}^\forall$ when *some transition* is possible from $\overline{s}$ with scheduling M and $\forall$-choice $\overline{\ell}^\forall$ such that the target composed state satisfies p. This means that the restriction to p is nonempty. $\delta_{M,p}$ then applies the restriction of the composed post-state to p only when allowed (otherwise all transitions remain):

$$allowed_{M,p}(\mathcal{V}, \mathcal{L}^\forall) \triangleq \exists \mathcal{V}', \mathcal{L}^\exists \cdot \Delta_M(\mathcal{V}, \mathcal{V}', \mathcal{L}) \wedge p(\mathcal{V}')$$
$$\delta_{M,p}(\mathcal{V}, \mathcal{V}', \mathcal{L}^\forall) \triangleq \big(\exists \mathcal{L}^\exists \cdot \Delta_M(\mathcal{V}, \mathcal{V}', \mathcal{L})\big) \wedge \big(allowed_{M,p}(\mathcal{V}, \mathcal{L}^\forall) \rightarrow p(\mathcal{V}')\big)$$

The resulting encoding is sound, but, unlike the case of finite branching, not complete.

Theorem 5. *If the set of formulas in Fig. 6a adapted to $\mathbb{L}^\sharp$ is satisfiable, then $TS \models_{\mathcal{G}} \varphi$.*

Example 7. Going back to the example in Fig. 5, choosing schedule $M = \{2\}$ and restriction $\ell^\sharp = (sum_1 = sum_2 + A_2[i_2] + y_2)$ when $pc_2 = 13$ ensures that the unique value of y_2 that satisfies the restriction is selected. With this value chosen, the assignment of the next line will produce a value of sum_2 that is equal to that of sum_1. This gives rise to the following winning strategy (at every

iteration): (i) schedule $\{1\}$ with any restriction until $pc_1 = 7$; (ii) schedule $\{2\}$ until $pc_2 = 13$, then schedule $\{2\}$ again with $\ell^\sharp = (sum_1 = sum_2 + A_2[i_2] + y_2)$, then $\{2\}$ again with any restriction; (iii) conclude the iteration by scheduling $\{1, 2\}$. As explained, the inductive invariant $sum_1 = sum_2$ is preserved in this behavior, *and* there are no "stuck" states (since, by construction of $\delta_{M,p}$, empty restrictions are lifted to the full set of transitions).

As a corollary of Thm. 5, satisfiability of the aforementioned formulas ensures that $TS \models \varphi$. To obtain an equi-satisfiable CHC encoding, we apply the transformation of Sec. 4. The resulting CHC encoding consists of the formulas in Fig. 6b adapted to use $\mathbb{L}^\sharp$ in the same way the formulas in Fig. 6a are adapted.

Corollary 3. *If the set of CHCs in Fig. 6b adapted to $\mathbb{L}^\sharp$ is satisfiable, then $TS \models_{\mathcal{G}} \varphi$.*

7 Evaluation

We implemented our CHC-encoding approach in a tool called HyHorn, on top of Z3 [23] (4.12.0) through its Python API, using Spacer [35,31] as a CHC solver. HyHorn takes as input a CFG, or several CFGs, whose transitions are annotated with two-vocabulary first-order formulas, and constructs a formula expressing the transition relation Tr. The specification is provided as: (i) a quantifier prefix $\forall\forall$, $\forall\exists$, or $\forall\forall\exists$, (ii) observation points ξ_i and (iii) safety condition ϕ that must hold globally in all observations. From that, the CHC encoding (Sec. 5, Sec. 6) is constructed and passed to Spacer for solving. HyHorn supports all first-order theories supported by Spacer (in our experiments, we used the theories of integer arithmetic and arrays). HyHorn further provides the option to apply predicate abstraction with a user-provided set of predicates, same as [8]. The abstraction is incorporated into the CHC encoding using the implicit abstraction encoding [13]. Notably, many of the benchmarks shown here are solved by HyHorn *even without* an abstraction, that is, directly over the concrete state.

In the area of hyperproperty verification, there are already several tools present, and the objective of our evaluation is to compare with such. Still, the field is not mature enough to have a standardized specification format (as is the case with SMTLIB and SV-COMP, to name a few). As a result, each tool has its own, opinionated, format, which varies from logical formulas to control-flow graphs. This makes it technically difficult to compare results of multiple solutions. In particular, benchmarks taken from previous work come in a range of formats, dictated by the tools that introduced them. A few of the benchmarks were translated by previous authors and, thanks to their efforts, are available in more than one format. For the majority of them, manual work is required for translating the benchmarks, and, more importantly, there is no one accepted translation, and the translation can introduce artifacts in the evaluation.

This forced us to prioritize the comparisons in our experiments. We chose to focus on comparison with the most closely related tools to our work. These

k-safety	HyHorn		HyPA	PCSat	Pdsc
	PA	concrete			
double square NI	0.56	—	67.0	—	6.8
double square NI ff	0.12	—	5.3	1.5	/
half square NI	0.30	0.30	63.0	13.4	3.4
squares sum	0.17	3.41	70.4	360.7	2.8
squares sum (simplified)	0.10	0.30	17.2	/	/
array insert	0.86	13.4	/	/	18.5
array insert (simplified)	1.33	2.58	16.2	378.6	/
exp1x3	0.08	0.09	2.9	/	/
fig 3 [FV19]	0.03	—	7.9	/	/
fig 2 [BF22]	0.11	—	13.6	/	/
col item symm	0.49	0.49	14.9	/	/
counter det	0.46	—	6.2	/	/
mult equiv	0.29	—	14.2	/	/
mult equiv (simplified)	0.19	—	10.3	/	/
array int mod	0.13	—	/	/	58.2
mult dist [FV19]	2.25	—	/	/	/

$\forall^*\exists^*$	HyHorn		HyPA
	PA	concrete	
non-det add	1.45	2.80	3.3
counter sum	0.09	—	4.0
async GNI	0.36	0.37	3.8
compiler opt 1	0.14	0.19	1.8
compiler opt 2	0.17	0.78	2.0
refine	0.18	0.29	4.0
refine 2	0.28	0.65	3.9
smaller	0.16	0.96	2.0
counter diff	0.17	—	6.8
fig 3 [BF22]	0.81	—	9.9
P1 (simple)	0.19	0.59	1.4
P1 (GNI)	0.26	0.75	138.7
P2 (GNI)	8.50	6.65	12.8
P3 (GNI)	0.32	0.20	4.6
P4 (GNI)	0.77	0.63	27.7

Table 1: Experimental results for k-safety properties. Time is measured in seconds. "—" represents timeouts after 20 minutes. "/" denotes benchmarks not present in the respective tool's suite.
In benchmark names, [FV19] refers to [27]; [BF22] refers to [8].

are HyPA [8], Pdsc [41], and PCsat [45]. HyPA is the most recent tool, and has already collected benchmarks from various previous papers (including Weaver [27]); Pdsc and PCsat both use the same first-order encoding as our starting point and thus are also relevant. HyPA's benchmarks include, in particular, $\forall^*\exists^*$ examples such as GNI, and Pdsc targets non-trivial alignments, and, as such all of its benchmarks have non-lockstep alignments.

Benchmarks For the evaluation of our approach we use *the full sets* of benchmarks from HyPA [8] and Pdsc [41]. The benchmarks of HyPA are divided into k-safety benchmarks, which are adopted from [43,27,41,45], and $\forall^*\exists^*$ benchmarks, which include refinement properties for compiler optimizations, general refinement of nondeterministic programs and generalized non-interference (GNI). For two benchmarks, we include both a simplified version as given in [8], as well as the original example. The benchmarks of Pdsc include more non-lockstep examples, as well as all of the comparator benchmarks from [43]. The comparator examples consist of both safe and unsafe instances. Weaver [27] considers 12 additional (sequential) k-safety benchmarks. As an additional test case, we manually translated the running example from Weaver, which is a 3-safety property with a nontrivial alignment, and tested it with HyHorn – HyHorn solved it in 2.25 seconds when provided with a few simple predicates (inequalities between program variables). We believe that being the running example makes it a good representative of the remaining 12. This brings our benchmark suite to a total of 112 k-safety examples (16 in Table 1 plus 96 comparator benchmarks).

Experiments To demonstrate the effectiveness of HyHorn we compare to HyPA [8], the most recent approach of formal verification of $\forall^*\exists^*$-hyperproperties, which

employs a construction using automata. To exhibit the benefits of the direct CHC encoding we also compare the k-safety examples to PCSat [45] and Pdsc [41]. Both encode the k-safety problem using FOL formulas as in Fig. 4a. PCSat uses a specialized solver for pfwCSP (a fragment of FOL that includes these formulas), while Pdsc solves the FOL formulas by enumerating alignments and using a CHC solver for each alignment. We do not compare to game solvers since, as reported by [8], state-of-the-art infinite-state game solvers, such as [26,2], which work without user-provided predicates, are unable to solve the benchmarks we consider.

We run HyHorn on the full set of benchmarks, and each of the other tools on the ones included in their benchmark suite. This is because each tool has its own input format: HyPA and Pdsc each has its own representation for the transition system and the property; PCSat accepts pfwCSP instances that are constructed manually. Some of the benchmarks are common to several tools.

All experiments are run on an AMD EPYC 74F3 with 32GB of memory. HyPA and PCSat are executed in Docker using their published artifacts[7].

Results The performance measurements of the tools for the k-safety benchmarks and for the $\forall^*\exists^*$ benchmarks are shown in Table 1. The results for the comparator examples are deferred to the extended version of the paper [34]. HyHorn is tested in two modes: with predicate abstraction ("PA") and without ("concrete"). HyPA and Pdsc require predefined predicates (the same predicates are used in all tools), while PCSat does not, but uses hints to solve 'array insert' and 'squares sum'. HyHorn solves almost all of the benchmarks with PA in under a second, outperforming previous approaches by up to two orders of magnitude; and also solves most of the benchmarks quickly without PA, esp. the $\forall^*\exists^*$ properties. In particular, HyHorn solves the two array benchmarks, while HyPA and PCSat do not support arrays and only solve simplified versions with integers. The runtime of HyHorn (both with and without predicates) on the comparator examples is similar to the runtime of Pdsc (see [34]), where HyHorn solves some benchmarks that Pdsc does not. (The other tools do not include these benchmarks.) On the unsafe examples, HyHorn provides a concrete counterexample, while Pdsc is only able to determine that there is no inductive invariant and alignment expressible with the given set of predicates.

8 Related Work

There is a large body of work studying verification of hyperproperties. While earlier verification techniques mostly focus on k-safety properties, or specific examples such as program equivalence, monotonocity, determinism [5,44,3,30,43,47] [24,27,41,1], lately verification of non-safety hyperproperties has been studied [4,16,45,7,8]. Below we discuss the works closest to ours.

[7] We evaluated HyHorn in Docker as well. There were no meaningful differences in runtime.

k-Safety Automatic verification of k-safety properties can be achieved by reducing the problem to a standard safety verification problem by means of self-composition [5], product-programs [3], and their derivatives [47,24]. Recently, however, it was identified that the alignment of the different copies has a substantial effect over the complexity of the verification problem [41,27,12]. Our approach is most related to the technique of Shemer *et al.* [41], which uses a semantic alignment that chooses which copy of the system performs a move based on the composed state of the different copies. They suggest an algorithm that iterates through the set of possible semantic alignments, such that in each iteration a CHC solver tries to prove the property, with the chosen alignmnet, using predicate abstraction. Unlike [41], HyHorn delegates the search for the alignment to the CHC solver, together with the search for the invariant, making the algorithm less dependent on predicate abstraction. Moreover, while [41] is restricted to k-safety only, our technique can handle k-safety as well as the more general $\forall^*\exists^*$-OHyperLTL.

$\forall^*\exists^*$ *Hyperproprties* Recently, verification of $\forall^*\exists^*$ hyperproperties has been studied, targeting both finite and infinite systems [45,16,8]. Unno *et al.* [45] present an approach based on an encoding of hyperproperties verification as satisfiability of formulas in FOL that extend Horn form with disjunctions, existential quantification and well founded relations. Deciding satisfiability of the generated set of formulas is based on a variant of the CEGIS framework. HyHorn is different as it encodes $\forall^*\exists^*$-OHyperLTL verification as a set of CHCs, which does not require a specialized solver and can use any off-the-shelf CHC solver. Coenen et.al. [16] suggested a game-based approach for verification of $\forall^*\exists^*$ properties over finite-state systems, which was then extended by Beutner *et al.* [8] to handle infinite-state systems. Similarly to [8], we use game semantics to solve $\forall^*\exists^*$ problems, but do not require building the game-graph in order to solve the game, instead reducing the game solution to satisfiability of CHCs. It is important to note that in the case of infinite branching degree, while the approach in [8] explicitly checks for emptiness of restrictions in hindsight, i.e., after they are used in a strategy, and removes them iteratively if needed, HyHorn embeds the emptiness requirements into the set of CHCs. Recently, [7] extended the game-based approach to use prophecy variables as a way to achieve completeness of the reduction to games. Extending our approach to this case is a promising avenue for future research.

Relational CHCs [40] present a method for discovering relational solutions to CHCs. Their setting is different: the inputs are CHCs that serve as the definition of the transitions, and synchronization is between sets of unknown predicates; at the current state, only lock-step semantics is considered. Furthermore, their algorithm extends and modifies Spacer [35], while our approach can use any CHC solver without modification.

Infinite-State Game Solving Our approach for verifying $\forall^*\exists^*$ hyperproperties is based on the game semantics of $\forall^*\exists^*$-OHyperLTL proposed in [16,8]. How-

ever, we do not propose a general game solving algorithm. Instead, we use the game semantics to come up with a first-order encoding of hyperproperty verification problems, which is then reduced to CHC solving. This allows us to use any CHC solver when solving the hyperproperty game. There is a large body of work on solving infinite-state games [21,9,46,26]. The game solving approach in [46] uses three-valued predicate abstraction to reduce the problem to finite-state game solving and requires to iteratively refine the controllable predecessor operator when computing candidate winning states. The approach in [26] targets games defined over the theory of linear real arithmetic and is based on an unrolling of the game and the use of Craig interpolants [18] to synthesize a winning strategy. The game solver in [2] is not restricted to a given FOL theory, but requires an interpolation procedure in order to compute sub-goals that are used to inductively split a game into sub-games. As reported by [8], game solving approaches [26,2], which work without a provided set of predicates, are unable to handle the infinite-state games for the benchmarks we consider. Moreover, the approaches in [26,16,2,8] cannot handle games that are defined using formulas over the theory of arrays, which are part of our benchmark. The approach of [9] to solving games over infinite graphs is based on reduction of games (including safety games) to CHCs. However, unlike the reduction presented in this paper, in [9] the games are encoded in a different fragment of Horn, namely $\forall\exists$-Horn where the head predicates can contain existential quantifiers. More recently (and concurrently with our work), [25] proposed a new reduction of game solving to CHC solving. Their approach handles safety games in which the branching degree of the "safe" player (the verifier in our setting) is bounded. In contrast, our encoding supports also infinite branching with the restrictions mechanism. Moreover, they do not support predicate abstraction, which is crucial for solving some of our benchmarks.

Restrictions as Underapproximations The use of restrictions as underapproxmations of the transition relation, inspired by [8], corresponds to the use of must hyper-transitions [36] in abstract transition systems [42,19] and games [20,22]. Similarly to [29,17], we use such underapproximations to replace an existential quantifier by universal quantification *within* the restriction.

9 Conclusion

We introduced a translation of a family of non-Horn first-order formulas to CHCs. This translation led to the first CHC encoding of a simultaneous inference of an invariant and an alignment for verifying k-safety properties. While the transformation itself is rather simple, identifying it was not straightforward and alluded previous works on the topic. We have further extended the CHC encoding to infer a witness function for existentially quantified traces arising in the verification of $\forall^*\exists^*$-OHyperLTL properties. Our experiments exhibit significant improvement over state-of-the-art hyperproperty verifiers thanks to the existence of advanced off-the-shelf CHC solvers, whose efficacy is expected to improve even

further. The approach shows promising capabilities in solving (many) hyeprproperty verification problems completely automatically. In some cases, predicates still have to be provided by the user, a limitation that we hope to overcome in the future by automatic inference of predicates. Applying (or extending) the transformation to obtain CHC encoding for other verification fragments is an interesting direction for future work.

Acknowledgment The research leading to these results has received funding from the European Research Council under the European Union's Horizon 2020 research and innovation programme (grant agreement No [759102-SVIS]). This research was partially supported by the Israeli Science Foundation (ISF) grant No. 2875/21 and No. 2117/23, and by the NSF-BSF grant No. 2018675.

References

1. Antonopoulos, T., Koskinen, E., Le, T. C., Nagasamudram, R., Naumann, D. A., and Ngo, M. An algebra of alignment for relational verification. *Proc. ACM Program. Lang. 7*, POPL (jan 2023).
2. Baier, C., Coenen, N., Finkbeiner, B., Funke, F., Jantsch, S., and Siber, J. Causality-based game solving. In *Computer Aided Verification - 33rd International Conference, CAV 2021, Virtual Event, July 20-23, 2021, Proceedings, Part I* (2021), A. Silva and K. R. M. Leino, Eds., vol. 12759 of *Lecture Notes in Computer Science*, Springer, pp. 894–917.
3. Barthe, G., Crespo, J. M., and Kunz, C. Relational verification using product programs. In *FM 2011: Formal Methods - 17th International Symposium on Formal Methods, Limerick, Ireland, June 20-24, 2011. Proceedings* (2011), pp. 200–214.
4. Barthe, G., Crespo, J. M., and Kunz, C. Beyond 2-safety: Asymmetric product programs for relational program verification. In *Logical Foundations of Computer Science, International Symposium, LFCS 2013, San Diego, CA, USA, January 6-8, 2013. Proceedings* (2013), S. N. Artëmov and A. Nerode, Eds., vol. 7734 of *Lecture Notes in Computer Science*, Springer, pp. 29–43.
5. Barthe, G., D'Argenio, P. R., and Rezk, T. Secure information flow by self-composition. In *17th IEEE Computer Security Foundations Workshop, (CSFW-17 2004), 28-30 June 2004, Pacific Grove, CA, USA* (2004), pp. 100–114.
6. Baumeister, J., Coenen, N., Bonakdarpour, B., Finkbeiner, B., and Sánchez, C. A temporal logic for asynchronous hyperproperties. In *Computer Aided Verification - 33rd International Conference, CAV 2021, Virtual Event, July 20-23, 2021, Proceedings, Part I* (2021), A. Silva and K. R. M. Leino, Eds., vol. 12759 of *Lecture Notes in Computer Science*, Springer, pp. 694–717.
7. Beutner, R., and Finkbeiner, B. Prophecy variables for hyperproperty verification. In *35th IEEE Computer Security Foundations Symposium, CSF 2022, Haifa, Israel, August 7-10, 2022* (2022), IEEE, pp. 471–485.
8. Beutner, R., and Finkbeiner, B. Software verification of hyperproperties beyond k-safety. In *Computer Aided Verification - 34th International Conference, CAV 2022, Haifa, Israel, August 7-10, 2022, Proceedings, Part I* (2022), S. Shoham and Y. Vizel, Eds., vol. 13371 of *Lecture Notes in Computer Science*, Springer, pp. 341–362.

9. Beyene, T. A., Chaudhuri, S., Popeea, C., and Rybalchenko, A. A constraint-based approach to solving games on infinite graphs. In *The 41st Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL '14, San Diego, CA, USA, January 20-21, 2014* (2014), S. Jagannathan and P. Sewell, Eds., ACM, pp. 221–234.
10. Bjørner, N., Gurfinkel, A., McMillan, K. L., and Rybalchenko, A. Horn clause solvers for program verification. In *Fields of Logic and Computation II - Essays Dedicated to Yuri Gurevich on the Occasion of His 75th Birthday* (2015), pp. 24–51.
11. Bjørner, N. S., McMillan, K. L., and Rybalchenko, A. On solving universally quantified horn clauses. In *Static Analysis - 20th International Symposium, SAS 2013, Seattle, WA, USA, June 20-22, 2013. Proceedings* (2013), F. Logozzo and M. Fähndrich, Eds., vol. 7935 of *Lecture Notes in Computer Science*, Springer, pp. 105–125.
12. Churchill, B. R., Padon, O., Sharma, R., and Aiken, A. Semantic program alignment for equivalence checking. In *Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI 2019, Phoenix, AZ, USA, June 22-26, 2019* (2019), K. S. McKinley and K. Fisher, Eds., ACM, pp. 1027–1040.
13. Cimatti, A., Griggio, A., Mover, S., and Tonetta, S. IC3 modulo theories via implicit predicate abstraction. In *Tools and Algorithms for the Construction and Analysis of Systems - 20th International Conference, TACAS 2014, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2014, Grenoble, France, April 5-13, 2014. Proceedings* (2014), E. Ábrahám and K. Havelund, Eds., vol. 8413 of *Lecture Notes in Computer Science*, Springer, pp. 46–61.
14. Clarkson, M. R., Finkbeiner, B., Koleini, M., Micinski, K. K., Rabe, M. N., and Sánchez, C. Temporal logics for hyperproperties. In *Principles of Security and Trust - Third International Conference, POST 2014, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2014, Grenoble, France, April 5-13, 2014, Proceedings* (2014), M. Abadi and S. Kremer, Eds., vol. 8414 of *Lecture Notes in Computer Science*, Springer, pp. 265–284.
15. Clarkson, M. R., and Schneider, F. B. Hyperproperties. *J. Comput. Secur. 18*, 6 (2010), 1157–1210.
16. Coenen, N., Finkbeiner, B., Sánchez, C., and Tentrup, L. Verifying hyperliveness. In *Computer Aided Verification - 31st International Conference, CAV 2019, New York City, NY, USA, July 15-18, 2019, Proceedings, Part I* (2019), I. Dillig and S. Tasiran, Eds., vol. 11561 of *Lecture Notes in Computer Science*, Springer, pp. 121–139.
17. Cook, B., and Koskinen, E. Reasoning about nondeterminism in programs. *SIGPLAN Not. 48*, 6 (jun 2013), 219–230.
18. Craig, W. Three Uses of the Herbrand-Gentzen Theorem in Relating Model Theory and Proof Theory. *J. of Symbolic Logic 22*, 3 (1957), 269–285.
19. Dams, D., and Namjoshi, K. S. The existence of finite abstractions for branching time model checking. In *19th IEEE Symposium on Logic in Computer Science (LICS 2004), 14-17 July 2004, Turku, Finland, Proceedings* (2004), IEEE Computer Society, pp. 335–344.
20. de Alfaro, L., Godefroid, P., and Jagadeesan, R. Three-valued abstractions of games: Uncertainty, but with precision. In *19th IEEE Symposium on Logic in Computer Science (LICS 2004), 14-17 July 2004, Turku, Finland, Proceedings* (2004), IEEE Computer Society, pp. 170–179.

21. de Alfaro, L., Henzinger, T. A., and Majumdar, R. Symbolic algorithms for infinite-state games. In *CONCUR 2001 - Concurrency Theory, 12th International Conference, Aalborg, Denmark, August 20-25, 2001, Proceedings* (2001), K. G. Larsen and M. Nielsen, Eds., vol. 2154 of *Lecture Notes in Computer Science*, Springer, pp. 536–550.
22. de Alfaro, L., and Roy, P. Solving games via three-valued abstraction refinement. In *CONCUR 2007 - Concurrency Theory, 18th International Conference, CONCUR 2007, Lisbon, Portugal, September 3-8, 2007, Proceedings* (2007), L. Caires and V. T. Vasconcelos, Eds., vol. 4703 of *Lecture Notes in Computer Science*, Springer, pp. 74–89.
23. de Moura, L. M., and Bjørner, N. Z3: an efficient SMT solver. In *Tools and Algorithms for the Construction and Analysis of Systems, 14th International Conference, TACAS 2008, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2008, Budapest, Hungary, March 29-April 6, 2008. Proceedings* (2008), pp. 337–340.
24. Eilers, M., Müller, P., and Hitz, S. Modular product programs. In *Programming Languages and Systems - 27th European Symposium on Programming, ESOP 2018, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2018, Thessaloniki, Greece, April 14-20, 2018, Proceedings* (2018), pp. 502–529.
25. Faella, M., and Parlato, G. Reachability games modulo theories with a bounded safety player. *Proceedings of the AAAI Conference on Artificial Intelligence 37*, 5 (June 2023), 6330–6337.
26. Farzan, A., and Kincaid, Z. Strategy synthesis for linear arithmetic games. *Proc. ACM Program. Lang. 2*, POPL (2018), 61:1–61:30.
27. Farzan, A., and Vandikas, A. Automated hypersafety verification. In *Computer Aided Verification - 31st International Conference, CAV 2019, New York City, NY, USA, July 15-18, 2019, Proceedings, Part I* (2019), I. Dillig and S. Tasiran, Eds., vol. 11561 of *Lecture Notes in Computer Science*, Springer, pp. 200–218.
28. Fedyukovich, G., Kaufman, S. J., and Bodík, R. Sampling invariants from frequency distributions. In *2017 Formal Methods in Computer Aided Design, FMCAD 2017, Vienna, Austria, October 2-6, 2017* (2017), D. Stewart and G. Weissenbacher, Eds., IEEE, pp. 100–107.
29. Godefroid, P., Nori, A. V., Rajamani, S. K., and Tetali, S. Compositional may-must program analysis: unleashing the power of alternation. In *Proceedings of the 37th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2010, Madrid, Spain, January 17-23, 2010* (2010), M. V. Hermenegildo and J. Palsberg, Eds., ACM, pp. 43–56.
30. Godlin, B., and Strichman, O. Regression verification: proving the equivalence of similar programs. *Softw. Test. Verification Reliab. 23*, 3 (2013), 241–258.
31. Gurfinkel, A. Program verification with constrained horn clauses (invited paper). In *Computer Aided Verification - 34th International Conference, CAV 2022, Haifa, Israel, August 7-10, 2022, Proceedings, Part I* (2022), S. Shoham and Y. Vizel, Eds., vol. 13371 of *Lecture Notes in Computer Science*, Springer, pp. 19–29.
32. Hoder, K., and Bjørner, N. S. Generalized property directed reachability. In *Theory and Applications of Satisfiability Testing - SAT 2012 - 15th International Conference, Trento, Italy, June 17-20, 2012. Proceedings* (2012), A. Cimatti and R. Sebastiani, Eds., vol. 7317 of *Lecture Notes in Computer Science*, Springer, pp. 157–171.

33. Hojjat, H., and Rümmer, P. The ELDARICA horn solver. In *2018 Formal Methods in Computer Aided Design, FMCAD 2018, Austin, TX, USA, October 30 - November 2, 2018* (2018), N. S. Bjørner and A. Gurfinkel, Eds., IEEE, pp. 1–7.
34. Itzhaky, S., Shoham, S., and Vizel, Y. Hyperproperty verification as chc satisfiability. Available at `https://doi.org/10.48550/arXiv.2304.12588`.
35. Komuravelli, A., Gurfinkel, A., and Chaki, S. SMT-based model checking for recursive programs. In *Computer Aided Verification - 26th International Conference, CAV 2014, Held as Part of the Vienna Summer of Logic, VSL 2014, Vienna, Austria, July 18-22, 2014. Proceedings* (2014), pp. 17–34.
36. Larsen, K. G., and Liu, X. Equation solving using modal transition systems. In *Proceedings of the Fifth Annual Symposium on Logic in Computer Science (LICS '90), Philadelphia, Pennsylvania, USA, June 4-7, 1990* (1990), IEEE Computer Society, pp. 108–117.
37. Lewis, H. R. Renaming a set of clauses as a horn set. *J. ACM 25*, 1 (1978), 134–135.
38. McCullough, D. Noninterference and the composability of security properties. In *Proceedings of the 1988 IEEE Symposium on Security and Privacy, Oakland, California, USA, April 18-21, 1988* (1988), IEEE Computer Society, pp. 177–186.
39. McMillan, K. L. Lazy annotation revisited. In *Computer Aided Verification - 26th International Conference, CAV 2014, Held as Part of the Vienna Summer of Logic, VSL 2014, Vienna, Austria, July 18-22, 2014. Proceedings* (2014), A. Biere and R. Bloem, Eds., vol. 8559 of *Lecture Notes in Computer Science*, Springer, pp. 243–259.
40. Mordvinov, D., and Fedyukovich, G. Property directed inference of relational invariants. In *2019 Formal Methods in Computer Aided Design, FMCAD 2019, San Jose, CA, USA, October 22-25, 2019* (2019), C. W. Barrett and J. Yang, Eds., IEEE, pp. 152–160.
41. Shemer, R., Gurfinkel, A., Shoham, S., and Vizel, Y. Property directed self composition. In *Computer Aided Verification - 31st International Conference, CAV 2019, New York City, NY, USA, July 15-18, 2019, Proceedings, Part I* (2019), I. Dillig and S. Tasiran, Eds., vol. 11561 of *Lecture Notes in Computer Science*, Springer, pp. 161–179.
42. Shoham, S., and Grumberg, O. Monotonic abstraction-refinement for CTL. In *Tools and Algorithms for the Construction and Analysis of Systems, 10th International Conference, TACAS 2004, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2004, Barcelona, Spain, March 29 - April 2, 2004, Proceedings* (2004), K. Jensen and A. Podelski, Eds., vol. 2988 of *Lecture Notes in Computer Science*, Springer, pp. 546–560.
43. Sousa, M., and Dillig, I. Cartesian hoare logic for verifying k-safety properties. In *Proceedings of the 37th ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI 2016, Santa Barbara, CA, USA, June 13-17, 2016* (2016), pp. 57–69.
44. Terauchi, T., and Aiken, A. Secure information flow as a safety problem. In *Static Analysis, 12th International Symposium, SAS 2005, London, UK, September 7-9, 2005, Proceedings* (2005), pp. 352–367.
45. Unno, H., Terauchi, T., and Koskinen, E. Constraint-based relational verification. In *Computer Aided Verification - 33rd International Conference, CAV 2021, Virtual Event, July 20-23, 2021, Proceedings, Part I* (2021), A. Silva and K. R. M. Leino, Eds., vol. 12759 of *Lecture Notes in Computer Science*, Springer, pp. 742–766.

46. Walker, A., and Ryzhyk, L. Predicate abstraction for reactive synthesis. In *Formal Methods in Computer-Aided Design, FMCAD 2014, Lausanne, Switzerland, October 21-24, 2014* (2014), IEEE, pp. 219–226.
47. Yang, W., Vizel, Y., Subramanyan, P., Gupta, A., and Malik, S. Lazy self-composition for security verification. In *Computer Aided Verification - 30th International Conference, CAV 2018, Held as Part of the Federated Logic Conference, FloC 2018, Oxford, UK, July 14-17, 2018, Proceedings, Part II* (2018), H. Chockler and G. Weissenbacher, Eds., vol. 10982 of *Lecture Notes in Computer Science*, Springer, pp. 136–156.
48. Zaks, A., and Pnueli, A. Covac: Compiler validation by program analysis of the cross-product. In *FM 2008: Formal Methods, 15th International Symposium on Formal Methods, Turku, Finland, May 26-30, 2008, Proceedings* (2008), pp. 35–51.
49. Zhu, H., Magill, S., and Jagannathan, S. A data-driven CHC solver. In *Proceedings of the 39th ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI 2018, Philadelphia, PA, USA, June 18-22, 2018* (2018), J. S. Foster and D. Grossman, Eds., ACM, pp. 707–721.