Property Directed Inference of Relational Invariants

Dmitry Mordvinov
JetBrains Research and Saint-Petersburg State University
Saint-Petersburg, Russia
Dmitry.Mordvinov@jetbrains.com

Grigory Fedyukovich
Princeton University
Princeton, USA
grigoryf@cs.princeton.edu

Abstract—Property Directed Reachability (PDR) is an efficient and scalable approach for solving systems of symbolic constraints, also known as Constrained Horn Clauses (CHC). In the case of non-linear CHCs, which may arise, e.g., from relational verification tasks, PDR aims to infer an inductive invariant for each uninterpreted predicate. However, in many practical cases, this reasoning is not successful, as invariants need to be discovered for groups of predicates, as opposed to individual predicates. We contribute a novel algorithm that identifies such groups automatically and complements the existing PDR technique. The key feature of the algorithm is that it does not require a possibly expensive synchronization transformation over the system of CHCs. We have implemented the algorithm on top of a state-of-the-art CHC solver SPACER. Our experimental evaluation shows that for some CHC systems, on which existing solvers diverge, our tool is able to discover relational invariants.

I. INTRODUCTION

With the progress in automated approaches to formal verification of programs against functional specifications [1]–[8], there is a growing need for applying this technology to verify multiple programs against relational specifications [9]–[12]. This discipline, called relational verification, is widely applicable in an iterative process of software development, when a current and the previous versions are compared and verified for the absence of newly introduced bugs. Another application is the verification of secure information flow properties, such as non-interference and time-balancing, in which executions of the same software are compared for various inputs.

Many automatic relational verification approaches are based on constructing a product program [9], [13]–[17] from the programs under comparison. This way, a given relational specification over multiple programs (or multiple executions of the same program) becomes a functional specification over the product program. Conceptually, such a relational-verification task can be addressed by state-of-the-art techniques, but in practice, most of them cannot handle a potentially complicated structure of the product program. The problem can be mitigated by merging certain loops in the product program (i.e., by applying so-called synchronization strategies), but often their discovery is manual or based on imprecise syntactic heuristics. Another downside is that the number of possible transformations is exponential with the number of merged programs. This paper contributes a fully automated approach to identify synchronization strategies that lead to an effective discovery of relational invariants for product programs.

We build on top of one of the most successful implementations of Property Directed Reachability (PDR) by Gurfinkel et. al. [5], called SPACER. PDR incrementally strengthens a given functional specification (i.e., a safety property) until it either becomes inductive, or a counterexample is found. It models programs with a set of logical implications, called Constrained Horn Clauses (CHCs), over a set of uninterpreted predicates. Intuitively, CHCs define the semantics of uninterpreted predicates, and by determining the satisfiability of CHCs w.r.t. some safety property, one can discover inductive invariants for programs under verification. SPACER maintains over-approximations and under-approximations of the semantics of uninterpreted predicates. It uses over-approximations to block spurious counterexamples, and under-approximations to analyze program traces without unrolling.

The CHCs constructed for product programs are essentially non-linear, and each uninterpreted predicate corresponds to a program under comparison. We propose a novel PDR-based approach that maintains over- and under-approximations of semantics of groups of predicates. It has the same effect as after doing a product-program transformation, but without actually transforming the system. More importantly, our algorithm identifies suitable groups of predicates on demand, by analyzing counterexamples-to-induction, obtained at different stages of our verification process. This allows us to effectively prune the search space of possible synchronization strategies, leading to performance gains. Note that without our approach, PDR attempts to discover an isolated invariant for uninterpreted predicate and often does not succeed (because e.g., the desired invariants are inexpressible by the modeling language).

We have implemented our approach on top of SPACER and have evaluated it on benchmarks arising from relational verification tasks. The experiments confirmed that for many CHC systems, on which SPACER diverges, our approach is able to discover relational invariants quickly.

The rest of the paper is structured as follows. We give background on CHCs in Sect. II and then we introduce our novel concept of relational invariants of CHCs in Sect. III. Our PDR-based algorithm for the discovery synchronization strategies and relational invariants is then presented in Sect. IV. In Sect. V, we show our experimental data. And finally, Sect. VI and Sect. VII concludes the paper.

II. PRELIMINARIES

A. Assertion language

Let $\Sigma$ be the first-order signature with equality and let $\mathcal{M}$ be some $\Sigma$-structure with the domain $|\mathcal{M}|$. For a $\Sigma$-sentence $\varphi$
(i.e. Σ-formula without free variables), $M$ is a model of $\varphi$ if $M$ satisfies $\varphi$, written $M \models \varphi$. Throughout the paper, we refer to the first-order language defined by $\Sigma$ as an assertion language. For $\Sigma$-formula with $n$ free variables $\varphi(x_1, \ldots, x_n)$, by $\cal M(\varphi)$ we denote a set of free variable valuations satisfying $\varphi$, i.e., $\cal M(\varphi) \triangleq \{(a_1, \ldots, a_n) \mid \cal M \models \varphi(a_1, \ldots, a_n)\} \subseteq |\cal M|^n$.

B. Constrained Horn Clauses

Let $\mathcal{R} = \{P_0, P_1, \ldots, P_n\}$ be a finite set of predicate symbols called relational (or uninterpreted) symbols. A constrained Horn clause (CHC) $C$ is a $\Sigma \cup \mathcal{R}$-formula of the form

$$\varphi \land R_1(\overline{x}_1) \land \ldots \land R_m(\overline{x}_m) \Rightarrow R(\overline{v})$$

where $\varphi$ is a quantifier-free $\Sigma$-formula, $R, R_i \in \mathcal{R}$, $\overline{v}$ and $\overline{x}_i$ are vectors of variables. The premise of the implication is called a body of $C$ and denoted by $body(C)$, the conclusion $R(\overline{v})$ is called a head of the clause. A CHC system is a finite set of CHCs. We treat the relational symbol $P_0$ as a special “query” symbol, the root of every derivation tree of a CHC system. If there is only one application of an uninterpreted symbol in the premise, the CHC is called linear (otherwise, non-linear). A CHC system is linear if every CHC in it is linear.

C. Safety Problem

A safety problem is a pair $\langle \mathcal{P}, \varphi_{safe}\rangle$, where $\mathcal{P} = \{C_1, \ldots, C_n\}$ is a CHC system, and $\varphi_{safe}$ is a $\Sigma$-formula over $\mathcal{P}$, called a safety property. We assume that heads of clauses $C_1, \ldots, C_n$ are applications of relational symbols with identical variables per each relational symbol, i.e., if clauses $C_i$ and $C_j$ have heads $R(\overline{v}_i)$ and $R(\overline{v}_j)$, then $\overline{v}_i = \overline{v}_j$. By rules $R$ we denote a set of clauses in $\mathcal{P}$ with the heads $R(\overline{v})$. By $body(R)$, we denote a disjunction of rules for $R$:

$$body(R) \triangleq \bigvee_{C \in rules(R)} body(C)$$

The merged body of relational symbols $R_1, \ldots, R_m \in \mathcal{R}$ is the conjunction of bodies

$$body(R_1, \ldots, R_m) \triangleq body(R_1) \land \ldots \land body(R_m),$$

where $\varphi \land \psi$ is a conjunction of $\varphi$ and $\psi$ that guarantees the disjointness of free variables of $\varphi \land \psi$:

$$\varphi(\overline{v}) \land \psi(\overline{y}) \triangleq (\varphi \land \psi)(\overline{v} \cup \overline{y})$$

We denote by $T_R$ the vector of existential (or local) variables of $R$, i.e. free variables of $body(R)$ without variables of the heads $T_R$.

Example 1. Let $\Sigma$ be a signature, and $M$ be the model of algebraic data types (ADT) where sort tree is defined with uninterpreted functions $leaf : tree$ and $node : N \times tree \times tree \rightarrow tree$. Consider the following safety problem that involves 1) counting nodes of a tree (relational symbol $size$), 2) summing the values of nodes (relational symbol $sum$), and 3) obtaining a new tree by increasing each node value of another tree by two (relational symbol $inc$):

$$T = leaf \land n = 0 \Rightarrow size(T, n)$$
$$T = node(v, L, R) \land n = 1 + n_L + n_R \land size(L, n_L) \land size(R, n_R) \Rightarrow size(T, n)$$
$$T = leaf \land s = 0 \Rightarrow sum(T, s)$$
$$T = node(v, L, R) \land s = v + s_L + s_R \land sum(L, s_L) \land sum(R, s_R) \Rightarrow sum(T, s)$$
$$T = leaf \land U = leaf \Rightarrow inc(T, U)$$
$$T = node(v, L, R) \land U = node(v + 2, L', R') \land inc(L', L') \land inc(R', R') \Rightarrow inc(T, U)$$
$$size(T, n) \land sum(T, s) \land inc(T', T') \land sum(T', s') \Rightarrow P_0(T, n, s, s')$$

We wish to prove that the sum of an inc-ed tree equals the sum plus twice the count of nodes of the original tree. Here, $\mathcal{R} = \{P_0, size, sum, inc\}$, $\overline{v}_0 = \{T, n, s, s'\}$, $\overline{v}_1 = \{T'\}$, $\overline{v}_{size} = \{T, n\}$, $\overline{v}_{size} = \{v, L, R, n, n_L, n_R\}$, $body(size) = (T = leaf \land n = 0) \lor (T = node(v, L, R) \land n = 1 + n_L + n_R \land size(L, n_L) \land size(R, n_R))$.

D. Fixedpoint Semantics

Let arities of $P_0, P_1, \ldots, P_n$ be $k_0, k_1, \ldots, k_n$ correspondingly. Let $X = \{X_0, X_1, \ldots, X_n\}$ be a tuple of relations with $X_i \subseteq |M|^{k_i}$. We denote the expansion $M \{P_0 \mapsto X_0, P_1 \mapsto X_1, \ldots, P_n \mapsto X_n\}$ by $(M, X)$. A semantics of a CHC system $\mathcal{P}$ in structure $M$ is the pointwise least $(n + 1)$-tuple of relations $X$ such that for all $P \in \mathcal{R}$, $(M, X) \models \forall \overline{v}_P \cup \overline{v}_P \cdot (body(P) \Rightarrow P(\overline{v}_P))$. The semantics of $\mathcal{P}$ is a least fixed point of immediate consequence operator of $\mathcal{P}$; it always exists by Knaster-Tarski theorem [18, 19]. We call the elements of the semantics tuple the semantics of corresponding procedures and write it as $(\llbracket P_0 \rrbracket_M, \llbracket P_1 \rrbracket_M, \ldots, \llbracket P_n \rrbracket_M)$.

A CHC system is safe with respect to $\varphi_{safe}$ if $\llbracket P_0 \rrbracket_M \subseteq M (\varphi_{safe})$. The CHC system in Example 1 is safe with respect to $s' = s + 2n$.

E. Safety Proofs

An Environment $\Pi$ maps $P \in \mathcal{R}$ to $\Sigma$-formulas over $\Sigma \cup \overline{v}_P$. For a $\Sigma \cup \mathcal{R}$-formula $\psi$, $\llbracket \psi \rrbracket_\Pi$ is a formula obtained by instantiating all applications of relational symbols in $\psi$ by their $\Pi$-images.

Given a safety problem $\langle \mathcal{P}, \varphi_{safe}\rangle$, an environment $\Pi$ is a safety proof, if it is safe and inductive:

$$M \models \forall \overline{v}_P . \Pi(P) \Rightarrow \varphi_{safe} \quad \text{(safety)}$$

for all $P \in \mathcal{R}$, $M \models \forall \overline{v}_P \cup \overline{v}_P \cdot (\llbracket body(P) \rrbracket_\Pi \Rightarrow \Pi(P)) \quad \text{(inductiveness)}$

Proposition 1. If there is a safety proof for safety problem $\langle \mathcal{P}, \varphi_{safe}\rangle$, then $\mathcal{P}$ is safe with respect to $\varphi_{safe}$.

III. Relational invariants

Systems of CHCs are widely used in automated verification for proving correctness of programs with respect to safety specifications. However, when it comes to verifying relational properties of several programs, modeled as non-linear CHCs,
the reasoning becomes significantly more complex. Often safety proofs are not expressible in their assertion language. For instance, although the system over ADTs in Example 1 is safe, there is no safety proof definable in $\mathcal{M}$.

Example 2. Consider a simpler example:

$$x = 0 \land z = 0 \Rightarrow \text{mul}(x, y, z)$$

$$x > 0 \land x' = x - 1 \land z = z' + y \land \text{mul}(x', y, z') \Rightarrow \text{mul}(x, y, z)$$

$$x = x' \land y = y' \land \text{mul}(x, y, z) \land \text{mul}(x', y', z') \Rightarrow \text{P}_0(x, y, z, x', y', z')$$

An invariant $\text{mul}(x, y, z) = (z = x \cdot y)$ is undefinable in linear integer arithmetic (LIA).

In this section, we generalize the notion of safety proof such that in both cases a definable proof exists. The key idea is to map groups of relational symbols (in contrast to singles) into formulas. This allows discovering the relations among variables from different calculations as opposed to summarizing each calculation in isolation.

A. Definition of the relational environment

By $\mathbb{N}^X$ we denote a set of multisets on $X$, i.e., a set of all maps from $X$ to natural numbers. If $\pi = x_1, \ldots, x_n$ is a vector of elements of $X$ (possibly repeating), we identify it with a multiset $\{x_i \mapsto \#x_i\}$, where $\#x_i$ is a number of occurrences of $x_i$ in $\pi$. Multisets are naturally ordered by inclusion: for $m_1, m_2 \in \mathbb{N}^X$, $m_1 \subseteq m_2$ iff $\forall x \in X, m_1(x) \leq m_2(x)$.

Definition 1. Let $\mathcal{P}$ be a system of CHCs over a set of relational symbols $\mathcal{R}$. A relational environment is a partial map from $\mathbb{N}^X$ to formulas that maps multiset $\mathcal{R} = \{R_1 \mapsto n_1, \ldots, R_k \mapsto n_k\}$ to a formula over $\forall \mathcal{R} \uparrow \mathcal{R}_R \equiv \forall R_1 \uparrow \ldots \uparrow \forall R_k \uparrow \mathcal{R}_R$.

Let $E$ be a relational environment. By $\text{dom}(E)$ we denote its domain. We assume that $\mathcal{R} \subseteq \text{dom}(E)$: if $R \in \mathcal{R} \setminus \text{dom}(E)$, then we map $R$ to $\top$. Let $R_1, \ldots, R_m$ be relational symbols from $\mathcal{R}$, and $\varphi$ be a formula. We (inductively) define

$$\llbracket \varphi \land R_1(\pi_1) \land \ldots \land R_m(\pi_m) \rrbracket_E \equiv \varphi \land \bigwedge_{R_1 \ldots R_m \in \{R_1, \ldots, R_m\}} \bigwedge_{\pi_1, \ldots, \pi_m \in \text{dom}(E)} E(R_1, \ldots, R_m)(\pi_1, \ldots, \pi_m)$$

and

$$\llbracket \bigwedge_{i=1}^k \bigvee_{j=1}^{m_i} F_{i,j} \rrbracket_E \equiv \bigvee_{1 \leq j_1 \leq m_1, 1 \leq j_k \leq m_k} \llbracket F_{1,j_1} \land \ldots \land F_{k,j_k} \rrbracket_E$$

Intuitively, (1) gathers all possible variants of “grouped” substitutions into (possibly merged) clause body $R_1(\pi_1) \land \ldots \land R_m(\pi_m)$. In (2), we use the relational environments to evaluate merged bodies of relations, which by definition are conjunctions of disjunctions of clause bodies. In (2), clause bodies are merged in each of $m_1 \ldots m_k$ possible ways, performing grouped substitution into merged clause bodies.

Example 3. Consider the following CHCs:

$$\varphi_1 \equiv f(x_1, x_2)$$

$$\varphi_2 \equiv f(x'_1, x'_2) \land f(x'_1, x'_2') \Rightarrow f(x_1, x_2)$$

$$\psi_1 \equiv g(y)$$

$$\psi_2 \equiv g(y')$$

and the following relational environment:

$$E = \{f \mapsto \top, g \mapsto \eta_1(y), (f, g) \mapsto \eta_2(x_1, x_2, y)\}$$

the evaluation of $\text{body}(f, g)$ in $E$ is as follows:

$$\llbracket \text{body}(f, g) \rrbracket_E = \left( \left( \varphi_1 \lor \left( \varphi_2 \land f(x'_1, x'_2) \land f(x'_1, x'_2') \right) \right) \land \left( \psi_1 \lor \psi_2 \land g(y') \right) \right)$$

The relational environments generalize the “classical” environments: if $E$ is the relational environment with the domain of singleton multisets, and $\Pi$ is a “classical” environment mapping relations to the same formulas, then for all $\Sigma \cup R$-formulas $\varphi$, $\llbracket \varphi \rrbracket_E$ is logically equivalent to $\llbracket \varphi \rrbracket_{\Pi}$.

B. Relational safety proofs

Given a safety problem $\langle \mathcal{P}, \varphi_{\text{safe}} \rangle$, a relational environment $E$ is a relational safety proof, if it is safe and inductive:

$$\mathcal{M} \models \forall \mathcal{P}_0, E(P_0) \Rightarrow \varphi_{\text{safe}} \quad \text{(safety)}$$

for all $\mathcal{P} \in \text{dom}(E)$, $\mathcal{M} \models \forall \mathcal{P} \cup \mathcal{T}_P (\llbracket \text{body}(\mathcal{P}) \rrbracket_E \Rightarrow E(\mathcal{P})) \quad \text{(inductiveness)}$

Besides evaluating the bodies in relational environments, the main difference between the “classical” and relational safety proofs is that the latter needs to be inductive relatively to the merged bodies. That is, if $\mathcal{P} \in \text{dom}(E)$ for non-singleton multiset $\mathcal{P}$, then $E$ should be inductive relatively to $\text{body}(\mathcal{P})$.

Example 4. Although for Example 2 there is no safety proof definable in LIA, there is a relational safety proof:

$$E = \{P_0 \mapsto s = s + 2n, \quad \text{sum} \mapsto \top, \quad \text{inc} \mapsto \top, \quad \langle \text{size}, \text{sum}, \text{sum}, \text{inc} \rangle \mapsto T_{\text{size}} = T_{\text{sum}} = T_{\text{inc}} \land T_{\text{sum}2} = U_{\text{inc}2} \Rightarrow s_{\text{sum}2} = s_{\text{sum}1} + 2n_{\text{size}} \}$$

Example 1 has a quantifier-free relational safety proof as well:

$$E = \{P_0 \mapsto s' = s + 2n, \quad \text{sum} \mapsto \top, \quad \text{inc} \mapsto \top, \quad \langle \text{size}, \text{sum}, \text{inc} \rangle \mapsto T_{\text{size}} = T_{\text{inc}} \land T_{\text{sum}2} = U_{\text{inc}2} \Rightarrow s_{\text{sum}2} = s_{\text{sum}1} + 2n_{\text{size}} \}$$

Example 1 has a quantifier-free relational safety proof as well:

C. Correctness

Theorem 1. If there is a relational safety proof $E$ for a safety problem $\langle \mathcal{P}, \varphi_{\text{safe}} \rangle$, then $\mathcal{P}$ is safe with respect to $\varphi_{\text{safe}}$.

Proof. By safety of $E$, it is sufficient to show that $[P_0]_E \subseteq \mathcal{M} (E(P_0))$. We prove it by constructing another CHC system $\mathcal{P}'$ and “classical” safety proof II using $E$. 

By (3), (4), and (5), we get:

**Example 3.**

Finally, we define a “classical” environment \( E \); from which we borrow the notation and general structure of the algorithm.

A. **Bounded assertion maps**

The algorithm stores its data in two data structures called a bounded assertion map and a relational bounded assertion map. The former maps \( P \in \mathcal{P} \) and a natural number \( b \) to a set of formulas over \( \mathcal{P} \), and the latter maps a multiset \( \overline{P} \in \mathbb{N}^\mathcal{P} \) and a natural number \( b \) to a set of formulas over \( \mathcal{P} \). Our algorithm maintains a bounded assertion map \( \rho \) and a relational bounded assertion map \( \sigma \).

Our algorithm uses \( \rho \) to witness a counterexample to safety and \( \sigma \) to build a relational safety proof. In particular, \( \rho \) stores the *reachability facts*, i.e., reachable branches of the system; \( \rho(P, b) \) is a set of formulas, under-approximating the \( b \)-bounded semantics of \( P \), i.e., the union of top-down derivations of the system \( P \) of the height \( b \). Dually, \( \sigma \) stores *summary facts* of the system, also known as *lemmas*. Formulas in \( \sigma(P, b) \) over-approximate the \( b \)-bounded semantics of \( P \); and are used for building a relational safety proof.

Finally, \( \rho \) and \( \sigma \) implicitly define the “classical” environment \( U_\rho \) and relational environment \( O_\sigma \), respectively. The former under-approximates and the latter over-approximates the bounded semantics of the system\(^1\).

\[ U_\rho(P) \equiv \bigvee \{ \delta \in \rho(P, c) \mid c \leq b \} \]

\[ O_\sigma(P) \equiv \bigwedge \{ \delta \in \sigma(\overline{R}, c) \mid \overline{R} \in \text{dom}(\sigma), \overline{R} \subseteq \overline{P}, c \geq b \} \]

Note that lemmas for a multiset \( \overline{P} \) subsume the lemmas of multisets included in \( \overline{P} \).

We abbreviate \( [\pi]_{U_\rho} \) and \( [\pi]_{O_\sigma} \) to \( [\pi]_\rho \) and \( [\pi]_\sigma \) correspondingly. For simplicity, we define \( U_{\rho}^{-1} \) and \( O_{\sigma}^{-1} \) (i.e., environments for level \(-1\)) to be relational environments mapping every multiset to \( \bot \).

B. **Outer loop**

Algorithm [1] shows a pseudo-code of the REL.RECMC procedure that iteratively weakens reachability facts \( \rho \) and strengthens summary facts \( \sigma \) until either \( \rho \) witnesses a counterexample (line 4) or \( \sigma \) becomes inductive (line 12).

\(^1\)The conjunction of an empty set is \( \top \), the disjunction is \( \bot \).
iteration $b$ of RELRECRec checks if the property violation is reachable in $b$ steps. If no bug is reachable (lines [6]–[11]), $\sigma$ contains a proof of bounded safety for $b$ steps. RELRECRec then propagates all inductive lemmas from $\sigma$ to level $b + 1$ and iterates if they are not sufficient for concluding the safety.

C. Inner loop

The RELBSAFETY algorithm shown in Algorithm 2 checks the safety of all top-down derivations of the system with all heights of derivations bounded by a given level $B$. It formulates and solves bounded reachability queries $\langle \mathcal{P}, \pi, b \rangle$, where $\mathcal{P} \in \mathbb{N}^K$, $\pi$ is the negation of a safety property for $\mathcal{P}$, and $b \in \mathbb{N}$. Intuitively, to answer $\langle \mathcal{P}, \pi, b \rangle$, we determine if $\mathcal{P}$ does not reach $\pi$ in $b$ steps.

Queries are stored in a queue $Q$ that initially contains only $\langle P_0, \neg \varphi_{\text{safe}}, B \rangle$. Each iteration begins with picking a query with the smallest $b$ (line 4), which may be answered positively (line 12) or negatively (line 8) or may give birth to child queries to be answered prior to answering this query (line 23). When all answers are returned, the algorithm answers the (UN–)REACHABLE result (line 34 or 32, respectively).

1) Inference of reachability facts: If $\pi$ is reachable in one step from predecessors bounded by $b - 1$ steps (line 4), the algorithm deduces new reachability facts for every $\mathcal{P}[i]$ (line 2). Informally, instead of exploring all branches of $\mathcal{P}[i]$, the algorithm explores only one branch $\psi_{\mathcal{P}[i]}$, chosen in a property-directed manner. Each query $\langle \mathcal{R}, \eta, c \rangle \in Q$, where $\eta$ is reachible with the updated environment $U^c_{\rho}$, is answered and removed from $Q$ (in particular, $\langle \mathcal{P}, \pi, b \rangle$).

To obtain a symbolic expression for a branch $\psi_{\mathcal{P}[i]}$, the algorithm uses a model-based projection (MBP) [6], [20]. Given a formula $\exists \pi, \tau$, where $\tau$ is quantifier-free, and a model $m$, an MBP $\langle \tau, \pi, m \rangle$ produces a quantifier-free conjunction of literals $\tau'$, such that $m \models \tau' \Rightarrow \exists \pi, \tau$, and if $\mathcal{M}$ admits quantifier elimination, then for each formula $\tau$, there is a finite number of models $m_1, \ldots, m_n$, such that $\exists \pi, \tau \models \psi_{\mathcal{P}[i]} \land \bigvee_{i=1}^n \text{MBP} (\tau, \pi, m_i)$. Intuitively, a series of model-based projections perform quantifier elimination from $\exists \pi, \tau$ lazily. Given $m$, MBP $\langle \text{body}(\mathcal{P}[i]) \rangle_{\rho}^{b-1}, \mathcal{T}_{\mathcal{P}[i]}(m)$ can be viewed as picking a branch of $\mathcal{P}[i]$, satisfied by $m$, and eliminating local variables $\mathcal{T}_{\mathcal{P}[i]}$ from it. Lazy quantifier elimination keeps the size of the reachability facts small and allows to consider only relevant behaviours of the system. In particular, although $\exists \pi, \tau$ is equivalent to a disjunction of branches, the algorithm considers only one branch per query; other branches will be considered on demand in the next iterations.

2) Inference of summary facts: If $\sigma$ is strong enough to prove the unreachability of $\pi$ (line 2), RELBSAFETY infers a new lemma by computing a Craig interpolant (denoted later ITP) of $\text{body}(\mathcal{P}) \models_{\sigma}^{b-1} \pi$. As a result, the new lemma over-approximates the $b$-bounded semantics of $P$, still proving its safety relatively to $\pi$. Note that the resulting lemma is formulated over the variables of $\mathcal{P}$, expressing relations among elements of $\mathcal{P}$. Every query $\langle \mathcal{R}, \eta, c \rangle \in Q$, such that the updated $\sigma$ proves the unsatisfiability of $\eta$, is immediately an-
served and removed from the queue (in particular, \( \langle P, \pi, b \rangle \)).

c) Query generation: If neither \([body(P)]_{\rho}^{b-1} \land \pi \) is satisfiable, nor \([body(P)]_{\sigma}^{b-1} \land \pi \) is unsatisfiable, then the reachability facts at level \( b-1 \) are too strong to witness a counterexample, while summary facts at level \( b-1 \) are too weak to prove the safety. In this case, there is a potential counterexample (counter-example to inductiveness in terms of \( IC3 \) (21)) \( cti \) assigned on line [13] which should be witnessed by \( \rho \) or blocked by \( \sigma \). The algorithm generates child queries, answers to which would help answering \( \langle P, \pi, b \rangle \) by either blocking the CTI or proving its reachability.

For each relation in \( P \), the algorithm picks a clause \( C \) witnessing \( cti \) (line [18]). Such clause is guaranteed to exist because \( cti \models [body(P)]_{\rho}^{b-1} \). Let \( R_1(\pi_1), \ldots, R_m(\pi_m) \) be all applications of relational symbols in \( C \) (line [19]). In the next step the algorithm tries to strengthen the summary facts by inferring lemmas blocking the CTI. To get this, the algorithm detects which summary facts are too coarse by splitting \( R_1(\pi_1), \ldots, R_m(\pi_m) \) into two groups: the applications witnessing \( cti \) (line [22]) and other applications \( apps \) (line [24]). As reachability facts of applications in the first group are already weak enough to witness the counterexample, it doesn’t make sense to strengthen their summary facts, so the algorithm proceeds with strengthening \( apps \).

At this step the algorithm behaves differently from the one in [5]. Instead of strengthening summary facts for each relation separately, it tries to infer the lemma for the group of predicates in \( apps \).

The algorithm is parametrized by an oracle \( PARTITION \) (line [25]) that splits the input set of atoms into a list of multisets of symbols and a list of vectors of variables from the corresponding applications. For example, the set of atoms \( \{ f(\pi_1), f(\pi_2), g(\pi) \} \) could be split into the list of multisets \([f \mapsto 2], [g \mapsto 1]\) of relations and the list of vectors the \([\pi_1, \pi_2, \pi]\) of variables. As a result, the \( rels \) list contains all multisets of symbols to be explored in the child queries.

In our implementation, \( PARTITION \) splits applications into a recursive and a non-recursive group. If the recursive group has more than \( |P| \) elements, it gets partitioned into the groups of size at most \( |P| \). This blocks the size of the queried multisets from the unbounded growth. \( PARTITION \) guesses a suitable partitioning of the recursive applications by detecting the synchronizations that preserve the inductiveness of the property in spirit of \( [16] \) (Sect. IV-D) demonstrates its work on Example [1].

For each multiset in \( rels \) the algorithm generates its own safety property, underapproximating the set of bad states of the group. The safety property for \( j \)-th group is the conjunction of the parent safety property \( \pi \) (line [15]), the constraints of all clauses witnessing \( cti \) (line [20]), the reachable child states witnessing \( cti \) (line [23]) and summary facts of the remaining multisets in the partition (line [29]). Intuitively,

RELBNDSAFETY strengthens the safety property \( \pi \) with the reachability information and child lemmas related to \( cti \). Afterward, the algorithm projects away all variables except \( vars[j] \) from the child safety property using MBP (line [30]), renames the variables and places a new bounded reachability query into the queue (line [31]). Note that the variables of child safety property renamed to formulate the property in terms of child relations (line [31]). Similar to the inference of reachability facts, using MBP does not break the correctness, while keeping the size of query formulas small.

D. Example

In this subsection, we demonstrate the several iterations of our algorithm for the problem in Example [1]. We prefer brevity to accuracy, so we simplify formulas wherever possible. For instance, we write \( \bot \) instead of \( n = 0 \land \bot \) and \( P_0 \) instead of the multiset \( \{ P_0 \rightarrow 1 \} \).

After the syntactic preprocessing, the algorithm handles the following system of CHCs, in which all variables are disjoint:

\[
T_1 = \text{leaf} \land n = 0 \Rightarrow \text{size}(T_1, n)
\]

\[
T_2 = \text{node}(v_1, L_1, R_1) \land n = 1 + n_1 + n_1^n \land \text{size}(L_1, n_1) \land \text{size}(R_1, n_1^n) \Rightarrow \text{size}(T_1, n)
\]

\[
T_3 = \text{leaf} \land n = 0 \Rightarrow \text{sum}(T_3, n)
\]

\[
T_4 = \text{node}(v_2, L_2, R_2) \land n = 1 + n_2 + n_2^n \land \text{sum}(L_2, n_2^2) \land \text{sum}(R_2, n_2^2) \Rightarrow \text{sum}(T_4, n)
\]

\[
T_4 = \text{leaf} \land U = \text{leaf} \Rightarrow \text{inc}(T_4, U)
\]

\[
A = T_0 \land B = T_0 \land C = T_0 \land D = E \land \text{size}(A, n_0) \land \text{sum}(B, n_0) \land \text{inc}(C, D) \land \text{sum}(E, s_0) \Rightarrow P_0(T_0, T_0, s_0, s_0')
\]

\[
\psi_{safe} \equiv s_0' = s_0 + 2n_0
\]

a) Level 0: The algorithm begins with calling RELBND-SAFETY for level 0 that puts query \( \{ P_0, s_0' \neq s_0 + 2n_0, 0 \} \) into \( Q \). Both \( [\text{body}(P_0)]_{\rho}^{b-1} \) and \( [\text{body}(P_0)]_{\sigma}^{b-1} \) are \( \bot \), so the algorithm gets into line [11] where it adds \( \text{ITP}(\bot, \bot, \bot) \) into \( \sigma(P_0) \). RELBND-SAFETY terminates with the result UNREACHABLE, but since the added lemma is not inductive, RELREC-MC proceeds to level 1.

b) Level 1: The RELBND-SAFETY algorithm begins with \( Q = \{ \{ P_0, s_0' \neq s_0 + 2n_0, 0 \} \} \). Here, \( [\text{body}(P_0)]_{\rho}^{b-1} \equiv \bot \) and \( [\text{body}(P_0)]_{\sigma}^{b-1} \equiv \psi_{safe} \equiv A = T_0 \land B = T_0 \land C = T_0 \land D = E. \) Since \( \psi_{safe} \) is satisfiable, the algorithm extracts \( cti = \{ A, B, C, D, E, T_0 \rightarrow \text{leaf}; n_0 \rightarrow 1; s_0 \rightarrow 0; s_0' \rightarrow 1 \} \) and goes to line [14]. Then it picks the only possible rule for \( P_0 \) with the body \( \psi_{safe} \land \text{size}(A, n_0) \land \text{sum}(B, s_0) \land \text{inc}(C, D) \land \text{sum}(E, s_0') \) Now \( cti \not\models [\text{size}(A, n_0)]_{\rho}^{b-1} \equiv A = \text{leaf} \land n_0 = 0, cti \models [\text{sum}(B, s_0)]_{\rho}^{b-1}, cti \models [\text{inc}(C, D)]_{\rho}^{b-1}, cti \not\models [\text{sum}(E, s_0')]_{\rho}^{b-1} \) so we get \( apps = \{ \text{size}(A, n_0), \text{sum}(E, s_0') \} \) and \( \psi = \psi_{safe} \land \psi_{safe} \land \text{size}(A, n_0) \land \text{sum}(E, s_0') \land \text{sum}(B, s_0) \land \text{inc}(C, D) \land \text{sum}(E, s_0') \) and \( \psi_{safe} = \psi_{safe} \land \text{size}(A, n_0) \land \text{sum}(E, s_0') \land \text{sum}(B, s_0) \land \text{inc}(C, D) \land \text{sum}(E, s_0') \) Since both \( inc \) and \( sum \) are non-recursive with \( P_0 \), the PARTITION oracle does nothing: \( \text{PARTITION}(apps) = \{ 1, A \} \).

Let \( \alpha_1 = \{ \text{size} \rightarrow 1, \text{sum} \rightarrow 1 \} \). To obtain the child bounded reachability query for \( \alpha_1 \), the algorithm projects away all variables from \( \psi \) except \( A, n_0, E, \) and \( s_0' \), obtaining
RecMc applies the technique described in Sect. III-D for the property, the algorithm projects away not inductive, so R
ITP(leaf) \neq 0. Thus, \sigma(\alpha_1, 0, 0) = \{1\}, and the query at level 0 is answered and removed from Q.

At the third iteration, the algorithm picks a query \langle\alpha_1, \psi_1, 0\rangle again. This time, \langle body(\alpha_1) \rangle_{\sigma} = \varphi_0 \land \delta_1(A, n, B, s_0) \land \delta_1(A, n, E, s_0). Since now \langle body(PO) \rangle_{\sigma} \land \neg \varphi_0 is unsatisfiable, \sigma(\alpha_1, 1) is updated to ITP(\langle body(PO) \rangle_{\sigma} \land \neg \varphi_0, \psi_0). The new environment is not inductive, so RELRECMe proceeds to level 2.

c) Level 2: Query \langle PO, \varphi_0, \psi_1, 2\rangle is picked from Q. Now, cti = \{A, B, C, T_0 \Rightarrow node(0, leaf, leaf); D, E \Rightarrow leaf; n_0, s_0 \Rightarrow 1\}. As cti \not\models \langle size(\alpha_2, n_0), \varphi_0 \land \langle sum(B, s_0) \rangle_{\rho}, cti \not\models \langle sum(C, D) \rangle_{\rho}, cti \not\models \langle E, s_0 \rangle_{\rho}\rangle, we get apps = \{size(\alpha_2, n_0), sum(B, s_0), inc(C, D), sum(E, s_0)\}. As none of the symbols is recursive with \rho, we get Groups = \{1, 2, 3, 4\}. To get the child safety property, the algorithm projects away T_0 and iterates with \{\langle PO, \varphi_0, \psi_1, 2\rangle, \langle \alpha_2, \psi_2, 1\rangle\}, where \alpha_2 \equiv (size \mapsto 1, \varphi_0 \land \langle sum \mapsto 2, inc \mapsto 1\rangle \land \psi_2 \models T_1 \equiv T_2 \equiv T_3 \equiv T_4 \equiv U \equiv T_1 \land s_3 \neq s_2 \neq 2n.

At the next iteration, \langle \alpha_2, \psi_2, 1\rangle is picked from Q. RELRECMe applies the technique described in Sect. III-D for the fast evaluation in \sigma. Let \beta_{size} = \begin{cases} T_1 = \text{leaf} \land n = 0 \lor \\
T_2 = \text{node}(v_1, L_1, R_1) \land n = 1 + n^L + n^R \land a_L \land a_R 
\end{cases} \beta_{sum_1} = \begin{cases} T_3 = \text{leaf} \land s_3 = 0 \lor \\
T_2 = \text{node}(v_2, L_2, R_2) \land s_2 = v_2 + s_2^L + s_2^R \land b_L \land b_R 
\end{cases} \beta_{sum_2} = \begin{cases} T_3 = \text{leaf} \land s_3 = 0 \lor \\
T_3 = \text{node}(v_3, L_3, R_3) \land s_3 = v_3 + s_3^L + s_3^R \land c_L \land c_R 
\end{cases} \beta_{inc} = \begin{cases} T_4 = \text{leaf} \land U = \text{leaf} \lor \\
T_4 = \text{node}(v_4, L_4, R_4) \land U = \text{node}(v_4 + 2L, R') \land d_L \land d_R 
\end{cases}

Here a_L, a_R, b_L, b_R, c_L, c_R, d_L, and d_R are fresh Boolean abstractions of relational symbol applications. Then:

\langle body(\alpha_2) \rangle_{\sigma} \models \begin{cases} \beta_{size} \land \beta_{sum_1} \land \beta_{sum_2} \land \beta_{inc} \land \\
(a_L \land b_L \Rightarrow \delta_1(L_1, n_1, L_2, s_2^L)) \land \\
(a_L \land b_R \Rightarrow \delta_1(L_1, n_1, R_2, s_2^R)) \land \\
(a_L \land c_L \Rightarrow \delta_1(L_1, n_1, L_2, s_2^L)) \ldots 
\end{cases}

If instead we straightforwardly convert the \langle body(\alpha_2) \rangle_{\sigma} into DNF and replace each possible combination of relational applications with \delta_1, we would get \sum_{k} \text{times larger formula.}

Suppose now that none of the child reachability facts is satisfied by cti. Then we get apps = \{size(L_1, n^L), size(R_1, n^R), sum(L_2, s_2^L), sum(R_2, s_2^R), sum(L_3, s_3^L), sum(R_3, s_3^R), inc(L_4, L'), inc(R_4, R')\}, with only recursive relations, and \psi \models \psi_2 \land T_1 = \text{node}(v_1, L_1, R_1) \land n = 1 + n^L + n^R \land \ldots

If PARTITION merges all applications into one group, we get the bounded reachability query for 8 relations, which may result in the query for 16 relations, and so on; in result, RELBNDSAFETY diverges. To control the size of multisets, PARTITION splits apps into groups of the size less or equal than \alpha_2 = 4. But there is already C_4^1 = 70 different variants of splitting 8 applications into two groups of size 4. To pick the best combination, PARTITION applies the following heuristic.

Since each bounded reachability query is created using MBP, it is a conjunction of literals. For each subset of apps, PARTITION detects the maximal inductive subset of literals.

In this case, the set of literals in \psi_2 is \{T_1 = T_2, T_3 = T_4, U = T_3, s_3 \neq s_2 \neq 2n\}. For example, \psi_1 \models T_1 \land T_2 is inductive relatively to size(R_1, n^R) and sum(R_2, s_2^R).

To verify this, we rename T_1 to T_3 and R_1 to R_2 (as T_3, R_1 and T_2, R_2 are the first arguments in applications of respectively size and sum), and check \psi \models R_1 = R_2. In our case, the whole \psi_2 is inductive relatively to the groups \{size(L_1, n^L), sum(L_2, s_2^L), sum(L_3, s_3^L), inc(L_4, L')\} and \{size(R_1, n^R), sum(R_2, s_2^R), sum(R_3, s_3^R), inc(R_4, R')\}, so PARTITION outputs Groups = \{\{1, 3, 5, 7\}, \{2, 4, 6, 8\}\}. For both groups, the bounded reachability queries for \alpha_2 at level 0 are added into Q.

At the following iterations, the algorithm infers the lemma T_1 = T_2 = T_4 \land U = T_4 \Rightarrow s_3 = s_2 + 2n and accomplishes the construction of a relational safety proof.

E. General properties

We now state the important properties of RELRECMe and RELBNDSAFETY. For the proof sketches, the reader is referred to [5].

Theorem 2. RELRECMe and RELBNDSAFETY are sound.

Theorem 3. RELBNDSAFETY is complete relatively to an oracle for satisfiability in \mathcal{M}.

Theorem 4. Given an oracle for satisfiability in \mathcal{M}, RELBNDSAFETY terminates.

By Theorem 4 RELRECMe is a co-semidecision procedure for safety problems, i.e., if \mathcal{P} is unsafe, the procedure is guaranteed to find a counterexample to safety. For finite-state systems, RELRECMe is a complete decision procedure; in this case, the algorithm is polynomial in the number of states.

If Algorithm 2 executes line 25 and Groups contains only singleton sets, then the behavior of RELBNDSAFETY is consistent with the behavior of BNDSAFETY [5]. In other words, our algorithm behaves the same as BNDSAFETY on linear CHC systems and generalizes its behavior on non-linear CHC systems. If PARTITION groups the input applications into singleton sets, then the algorithm infers the “classic” safety proofs, behaving similarly to the original algorithm.
We have compared our implementation with SPACER problems from [8], not arising from relational verification. We have evaluated the implementation against SPACER and the HOICE tool [8] on two benchmark suites. We have run the experiments on an Arch Linux machine Intel(R) Core(TM) i5-6200U CPU @ 2.30GHz processor with a 30-second timeout.

The first benchmark suite contains 840 “classic” safety problems from [8], not arising from relational verification. We have compared our implementation with SPACER and HOICE [8] and demonstrated its viability. SPACER solved 788 out of 840 problems with 50 timeouts and 2 runtime errors. Our implementation solved 806 problems with 34 timeouts. The overhead on solved problems is insignificant (less than 0.1 sec on 87% of problems). Our implementation solved most of the problems solved by SPACER. However, there are 10 problems solved by SPACER, but not by our implementation, currently. HOICE solved 808 problems with 26 timeouts and 6 runtime errors, but both SPACER and our implementation of RELREC/MC outperformed HOICE on the solved problems.

The second benchmark suite contains 37 relational verification problems adapted from [23]. We have evaluated our implementation against SPACER, HOICE, and the CHCPRODUCT algorithm [16] implementing a syntactic transformation of the input system with the subsequent solution by SPACER. A schematic comparison is shown in Fig. 1.

Both SPACER and HOICE solved only 11 of 37 problems within a 5-minute timeout. CHCPRODUCT solved 24 problems, and RELREC/MC solved 32 out of 37 problems. Note that RELREC/MC solved some problems that SPACER provided with syntactically merged clauses did not solve. For large unsafe problems, e.g., point-location*, CHCPRODUCT generates the exponential amount of rules and times out, but other solvers detect a counterexample in a few seconds.

VI. RELATED WORK

Various relational verification techniques are based on automated or semi-automated analysis of product programs [9–17], [24]. All these approaches treat a verification engine for functional specification as black-box. Thus they have to predetermine the synchronization strategies. In contrast, our approach does not construct a product program explicitly but leverages an SMT solver while discovering both synchronization strategies and relational invariants at the same time.

Cartesian Hoare Logic [25], [26] for proving k-safety properties consists in a set of rules and heuristics for aligning loops in programs under comparison. These techniques analyze loop guards, conditionals, relational pre- and postconditions. To detect a synchronization strategy, [26] identifies a maximal group of loops, where the termination of one loop implies the termination of others (otherwise, it fails to find relational invariants, even if they exist). In contrast, our approach is agnostic to termination properties and can discover relational invariants for loops with unequal numbers of iterations.

There are some transformation techniques for nonlinear CHC systems that enable existing solvers to discover a relational invariant automatically [15], [16]. The CHCPRODUCT transformation [16] resembles a Cartesian product over a set of relation symbols of the CHC system. When the relational symbols that are being transformed have clauses with more than one recursive reference, the CHCPRODUCT is not uniquely determined. In order to tackle this, an extension of the technique called synchronous CHCPRODUCT tries to select a product that joins structurally similar recursive references together. Alternatively, [15] proposes a transformation based on well-known FOLD/UNFOLD rules. Although the resulting CHC systems are easier to solve, the cost of these transformations grows exponentially with the number of merged predicates. By comparison, our approach transforms recursive references on demand using models of SMT queries and thus does not lead to an exponential explosion in complexity.

A recent technique [17] is the closest to our work. It analyzes counterexamples to identify a non-lockstep synchronization strategy, but it uses a given set of predicates to discover relational invariants. In contrast, our approach does not require predicates and obtain invariants using interpolation, effectively exploiting the features inherited from [5]. We plan to support non-lockstep synchronization strategies in our future work.

VII. CONCLUSION

We have presented a novel approach based on PDR to solve non-linear CHCs. Its key feature is the ability to discover relational invariants that safely over-approximate semantics of groups of uninterpreted predicates. More importantly, our approach identifies automatically which predicates should be considered in groups. We have implemented the algorithm on top of the SPACER tool and confirmed its practical success on a set of benchmarks arising from relational verification tasks.
REFERENCES


