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 Σ be the first-order signature with equality and let $\mathcal{M}$ be some Σ-structure with the domain $|\mathcal{M}|$. For a Σ-sentence φ

ISBN: 978-0-9835678-9-9.

(i.e. Σ-formula without free variables), $\mathcal{M}$ is a *model* of φ if $\mathcal{M}$ satisfies φ, written $\mathcal{M} \models \varphi$. Throughout the paper, we refer to the first-order language defined by Σ as an *assertion* language. For Σ-formula with n free variables $\varphi(x_1, \ldots, x_n)$, by $\mathcal{M}(\varphi)$ we denote a set of free variable valuations satisfying φ, i.e., $\mathcal{M}(\varphi) \stackrel{\text{def}}{=} \{\langle a_1, \ldots, a_n\rangle \mid \mathcal{M} \models \varphi(a_1, \ldots, a_n)\} \subseteq |\mathcal{M}|^n$.

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

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

where φ is a quantifier-free Σ-formula, $R, R_i \in \mathcal{R}$, $\overline{v}_R$ 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 $R(\overline{v}_R)$ 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 φ_{safe} is a Σ-formula over $\overline{v}_{P_0}$, 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 $R(\overline{v}_R)$ and $R(\overline{v}'_R)$, then $\overline{v}_R = \overline{v}'_R$.

By $rules(R)$ we denote a set of clauses in $\mathcal{P}$ with the heads $R(\overline{v}_R)$. By $body(R)$, we denote a disjunction of rules for R:

$$body(R) \stackrel{\text{def}}{=} \bigvee_{C \in rules(R)} body(C)$$

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

$$body(R_1, \ldots, R_m) \stackrel{\text{def}}{=} body(R_1) \stackrel{+}{\wedge} \ldots \stackrel{+}{\wedge} body(R_m),$$

where $\varphi \stackrel{+}{\wedge} \psi$ is a conjunction of φ and ψ that guarantees the disjointedness of free variables of $\varphi \wedge \psi$:

$$\varphi(\overline{x}) \stackrel{+}{\wedge} \psi(\overline{y}) \stackrel{\text{def}}{=} (\varphi \wedge \psi)\,(\overline{x} \uplus \overline{y})$$

We denote by $\overline{\ell}_R$ the vector of *existential* (or *local*) variables of R, i.e. free variables of $body(R)$ without variables of the heads $\overline{v}_R$.

Example 1. Let Σ be a signature, and $\mathcal{M}$ be the model of algebraic data types (ADT) where sort $tree$ is defined with uninterpreted functions $leaf : tree$ and $node : \mathbb{N} \times tree \times tree \to tree$. 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):

$$\begin{aligned}
T = leaf \wedge n = 0 &\Rightarrow size(T,n)\\
T = node(v,L,R) \wedge n = 1 + n^L + n^R \wedge size(L,n^L) \wedge size(R,n^R) &\Rightarrow size(T,n)\\
T = leaf \wedge s = 0 &\Rightarrow sum(T,s)\\
T = node(v,L,R) \wedge s = v + s^L + s^R \wedge & \\
sum(L,s^L) \wedge sum(R,s^R) &\Rightarrow sum(T,s)\\
T = leaf \wedge U = leaf &\Rightarrow inc(T,U)\\
T = node(v,L,R) \wedge U = node(v+2,L',R') \wedge & \\
inc(L,L') \wedge inc(R,R') &\Rightarrow inc(T,U)\\
size(T,n) \wedge sum(T,s) \wedge inc(T,T') \wedge sum(T',s') &\Rightarrow P_0(T,n,s,s')\\
\varphi_{safe} &\stackrel{\text{def}}{=} s' = s + 2n
\end{aligned}$$

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, $\mathcal{R} = \{P_0, size, sum, inc\}$, $\overline{v}_{P_0} = \{T, n, s, s'\}$, $\overline{\ell}_{P_0} = \{T'\}$, $\overline{v}_{size} = \{T, n\}$, $\overline{\ell}_{size} = \{v, L, R, n^L, n^R\}$, $body(size) = (T = leaf \wedge n = 0) \vee \big(T = node(v,L,R) \wedge n = 1 + n^L + n^R \wedge size(L,n^L) \wedge size(R,n^R)\big)$.

D. Fixedpoint Semantics

Let arities of $P_0, P_1, \ldots, P_n$ be $k_0, k_1, \ldots, k_n$ correspondingly. Let $\overline{X} = \langle X_0, X_1, \ldots, X_n\rangle$ be a tuple of relations with $X_i \subseteq |\mathcal{M}|^{k_i}$. We denote the expansion $\mathcal{M}\{P_0 \mapsto X_0, P_1 \mapsto X_1, \ldots, P_n \mapsto X_n\}$ by $\big(\mathcal{M}, \overline{X}\big)$.

A *semantics* of a CHC system $\mathcal{P}$ in structure $\mathcal{M}$ is the pointwise least $(n+1)$-tuple of relations $\overline{X}$ such that for all $P \in \mathcal{R}$, $\big(\mathcal{M}, \overline{X}\big) \models \forall \overline{v}_P \cup \overline{\ell}_P.\,(body(P) \Rightarrow P(\overline{v}_P))$. 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 $\big\langle [\![P_0]\!]_{\mathcal{M}}, [\![P_1]\!]_{\mathcal{M}}, \ldots, [\![P_n]\!]_{\mathcal{M}}\big\rangle$.

A CHC system is *safe* with respect to φ_{safe} if $[\![P_0]\!]_{\mathcal{M}} \subseteq \mathcal{M}(\varphi_{safe})$. The CHC system in Example 1 is safe with respect to $s' = s + 2n$.

E. Safety Proofs

An *Environment* Π maps $P \in \mathcal{R}$ to Σ-formulas over $\Sigma \cup \overline{v}_P$. For a $\Sigma \cup \mathcal{R}$-formula ψ, $[\![\psi]\!]_\Pi$ is a formula obtained by instantiating all applications of relational symbols in ψ by their Π-images.

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

$$\mathcal{M} \models \forall \overline{v}_{P_0}.\Pi(P_0) \Rightarrow \varphi_{safe} \qquad \text{(safety)}$$

$$\text{for all } P \in \mathcal{R},\ \mathcal{M} \models \forall \overline{v}_P \cup \overline{\ell}_P.([\![body(P)]\!]_\Pi \Rightarrow \Pi(P)) \qquad \text{(inductiveness)}$$

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 φ_{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,

ISBN: 978-0-9835678-9-9.

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:

$$x=0 \wedge z=0 \Rightarrow mul(x,y,z)$$
$$x>0 \wedge x'=x-1 \wedge z=z'+y \wedge mul(x',y,z') \Rightarrow mul(x,y,z)$$
$$x=x' \wedge y=y' \wedge mul(x,y,z) \wedge mul(x',y',z') \Rightarrow P_0(x,y,z,x',y',z')$$
$$\varphi_{safe} \stackrel{\text{def}}{=} z=z'$$

An invariant $mul(x,y,z) = (z = x \cdot y)$ 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 $\overline{x} = x_1, \ldots, x_n$ is a vector of elements of X (possibly repeating), we identify it with a multiset $\{x_k \mapsto \#x_k\}$, where $\#x_i$ is a number of occurrences of x_i in $\overline{x}$. 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 $\mathbb{N}^{\mathcal{R}}$ to formulas that maps multiset $\overline{R} = \{R_1 \mapsto n_1, \ldots, R_k \mapsto n_k\}$ to a formula over $\overline{v}_{\overline{R}} \stackrel{\text{def}}{=} \underbrace{\overline{v}_{R_1} \uplus \ldots \uplus \overline{v}_{R_1}}_{n_1 \text{ times}} \uplus \ldots \uplus \underbrace{\overline{v}_{R_k} \uplus \ldots \uplus \overline{v}_{R_k}}_{n_k \text{ times}}$.

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

$$[\![\varphi \wedge R_1(\overline{x}_1) \wedge \ldots \wedge R_m(\overline{x}_m)]\!]_E \stackrel{\text{def}}{=} \quad (1)$$
$$\varphi \wedge \bigwedge_{\substack{R_{i_1},\ldots,R_{i_k} \in \{R_1,\ldots,R_m\} \\ \langle R_{i_1},\ldots,R_{i_k}\rangle \in dom(E)}} E(R_{i_1},\ldots,R_{i_k})(\overline{x}_{i_1},\ldots,\overline{x}_{i_k})$$

and

$$\left[\!\!\left[\bigwedge_{i=1}^{k} \bigvee_{j=1}^{m_i} F_{i,j} \right]\!\!\right]_E \stackrel{\text{def}}{=} \bigvee_{\substack{1 \leq j_1 \leq m_1 \\ \ldots \\ 1 \leq j_k \leq m_k}} [\![F_{1,j_1} \wedge \ldots \wedge F_{k,j_k}]\!]_E \quad (2)$$

Intuitively, (1) gathers all possible variants of "grouped" substitutions into (possibly merged) clause body $R_1(\overline{x}_1) \wedge \ldots \wedge R_m(\overline{x}_m)$. 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 $m_1 \cdot \ldots \cdot m_k$ possible ways, performing grouped substitution into merged clause bodies.

Example 3. Consider the following CHCs:

$$\varphi_1 \Rightarrow f(x_1,x_2)$$
$$\varphi_2 \wedge f(x_1',x_2') \wedge f(x_1'',x_2'') \Rightarrow f(x_1,x_2)$$
$$\psi_1 \Rightarrow g(y)$$
$$\psi_2 \wedge g(y') \Rightarrow g(y)$$

and the following relational environment:

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

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

$$[\![body(f,g)]\!]_E = \left[\!\!\left[\Big(\varphi_1 \vee \big(\varphi_2 \wedge f(x_1',x_2') \wedge f(x_1'',x_2'')\big)\Big) \wedge \Big(\psi_1 \vee \big(\psi_2 \wedge g(y')\big)\Big) \right]\!\!\right]_E =$$
$$[\![\varphi_1 \wedge \psi_1]\!]_E \vee [\![\varphi_1 \wedge \psi_2 \wedge g(y')]\!]_E \vee$$
$$[\![\varphi_2 \wedge \psi_1 \wedge f(x_1',x_2') \wedge f(x_1'',x_2'')]\!]_E \vee$$
$$[\![\varphi_2 \wedge \psi_2 \wedge f(x_1',x_2') \wedge f(x_1'',x_2'') \wedge g(y')]\!]_E =$$
$$(\varphi_1 \wedge \psi_1) \vee (\varphi_1 \wedge \psi_2 \wedge \eta_1(y')) \vee (\varphi_2 \wedge \psi_1) \vee$$
$$(\varphi_2 \wedge \psi_2 \wedge \eta_1(y') \wedge \eta_2(x_1',x_2',y') \wedge \eta_2(x_1'',x_2'',y'))$$

The relational environments generalize the "classical" environments: if E is the relational environment with the domain of *singleton* multisets, and Π is a "classical" environment mapping relations to the same formulas, then for all $\Sigma \cup \mathcal{R}$-formulas φ, $[\![\varphi]\!]_E$ is logically equivalent to $[\![\varphi]\!]_\Pi$.

B. Relational safety proofs

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

$$\mathcal{M} \models \forall \overline{v}_{P_0}.E(P_0) \Rightarrow \varphi_{safe} \quad \text{(safety)}$$

for all $\overline{P} \in dom(E)$, $\mathcal{M} \models \forall \overline{v}_{\overline{P}} \cup \overline{\ell}_{\overline{P}}.\big([\![body(\overline{P})]\!]_E \Rightarrow E(\overline{P})\big)$ (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 $\overline{P} \in dom(E)$ for non-singleton multiset $\overline{P}$, then E should be inductive relatively to $body(\overline{P})$.

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

$$E = \{P_0 \mapsto z = z', mul \mapsto \top,$$
$$\langle mul, mul\rangle \mapsto (x^{mul_1} = x^{mul_2} \wedge y^{mul_1} = y^{mul_2} \Rightarrow z^{mul_1} = z^{mul_2})\}$$

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

$$E = \{P_0 \mapsto s' = s+2n, \quad sum \mapsto \top, \quad inc \mapsto \top,$$
$$\langle size, sum, sum, inc\rangle \mapsto T^{size} = T^{sum_1} = T^{inc} \wedge$$
$$T^{sum_2} = U^{inc_2} \Rightarrow s^{sum_2} = s^{sum_1} + 2n^{size}\}$$

C. Correctness

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

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

ISBN: 978-0-9835678-9-9.

For each $\overline{P} \in dom(E)$, we introduce a fresh relational symbol $R_{\overline{P}}$. We define a relational environment E' that maps $\overline{P}$ to $R_{\overline{P}}(\overline{v}_{\overline{P}})$. Now we define $\mathcal{P}'$ over the relational symbols $\{R_{\overline{P}} \mid \overline{P} \in dom(E)\}$. For each $R_{\overline{P}}$, we put $body(R_{\overline{P}}) \stackrel{\text{def}}{=} [\![body(\overline{P})]\!]_{E'}$; the head of each rule for $R_{\overline{P}}$ is $R_{\overline{P}}(\overline{v}_{\overline{P}})$.

For instance, for system $\mathcal{P}$ and relational environment E in Example 3, $\mathcal{P}'$ is as follows:

$$\begin{aligned}
\varphi_1 &\Rightarrow R_f(x_1, x_2)\\
\varphi_2 \wedge R_f(x_1', x_2') \wedge R_f(x_1'', x_2'') &\Rightarrow R_f(x_1, x_2)\\
\psi_1 &\Rightarrow R_g(y)\\
\psi_2 \wedge R_g(y') &\Rightarrow R_g(y)\\
\varphi_1 \wedge \psi_1 &\Rightarrow R_{f,g}(x_1, x_2, y)\\
\varphi_1 \wedge \psi_2 \wedge R_g(y') &\Rightarrow R_{f,g}(x_1, x_2, y)\\
\varphi_2 \wedge \psi_1 \wedge R_f(x_1', x_2') \wedge R_f(x_1'', x_2'') &\Rightarrow R_{f,g}(x_1, x_2, y)\\
\varphi_2 \wedge \psi_2 \wedge R_f(x_1{}', x_2{}') \wedge R_f(x_1{}'', x_2{}'') \wedge R_g(y') \wedge & \\
R_{f,g}(x_1{}', x_2{}', y) \wedge R_{f,g}(x_1{}'', x_2{}'', y) &\Rightarrow R_{f,g}(x_1, x_2, y)
\end{aligned}$$

The semantics of $R_{\overline{P}}$ in $\mathcal{P}'$ is the Cartesian product $\bigtimes_{p \in \overline{P}} [\![p]\!]_{\mathcal{M}}$, as $R_{\overline{P}}$ uses the disjoint variables and "reaches" every state reached by each $p \in \overline{P}$. In particular,

$$[\![P_0]\!]_{\mathcal{M}} = [\![R_{P_0}]\!]_{\mathcal{M}} \tag{3}$$

Finally, we define a "classical" environment Π mapping $R_{\overline{P}}$ to $E(\overline{P})$. Π is a safety proof for $\mathcal{P}'$: it is safe and inductive by construction. Note that

$$\Pi(R_{P_0}) = E(P_0) \tag{4}$$

As the semantics of $\mathcal{P}'$ is the pointwise least inductive tuple of relations, we have

$$[\![R_{P_0}]\!]_{\mathcal{M}} \subseteq \mathcal{M}(\Pi(R_{P_0})) \tag{5}$$

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

$$[\![P_0]\!]_{\mathcal{M}} = [\![R_{P_0}]\!]_{\mathcal{M}} \subseteq \mathcal{M}(\Pi(R_{P_0})) = \mathcal{M}(E(P_0)) \subseteq \mathcal{M}(\varphi_{safe})$$

□

D. Fast evaluation

By (2), a naive implementation of $[\![\varphi]\!]_E$ requires to calculate $m_1 \cdot \ldots \cdot m_k$ rules, which has the exponential complexity with the growing k. In this subsection, we demonstrate an alternative method allowing to avoid an explicit enumeration of all combinations of rules being merged, which is inherent to the syntactic transformation [16]. The key insight is to build an *equisatisfiable* formula instead of precise calculation using rules (1) and (2).

Let $R_1(\overline{x}_1), \ldots, R_m(\overline{x}_m)$ be all applications of relational symbols in φ. For each application $R_i(\overline{x}_i)$, we introduce a fresh propositional atom (i.e., a nullary predicate symbol) a_i. Let φ' be a formula obtained by replacing all occurrences of $R_i(\overline{x}_i)$ with a_i for all i in φ. Note that $[\![\varphi]\!]_E$ is equisatisfiable with:

$$\varphi' \wedge \bigwedge_{\substack{R_{i_1},\ldots,R_{i_k} \in \{R_1,\ldots,R_m\}\\ \langle R_{i_1},\ldots,R_{i_k}\rangle \in dom(E)}} \big(a_{i_1} \wedge \ldots \wedge a_{i_k} \Rightarrow E(R_{i_1}, \ldots, R_{i_k})(\overline{x}_{i_1}, \ldots, \overline{x}_{i_k})\big)$$

Example 5. Recall Example 3. The following formula is equisatisfiable with $[\![body(f, g)]\!]_E$:

$$\big(\varphi_1 \vee (\varphi_2 \wedge a_1 \wedge a_2)\big) \wedge \big(\psi_1 \vee (\psi_2 \wedge a_3)\big) \wedge \big(a_3 \Rightarrow \eta_1(y')\big) \wedge \\ \wedge \big(a_1 \wedge a_3 \Rightarrow \eta_2(x_1', x_2', y')\big) \wedge \big(a_2 \wedge a_3 \Rightarrow \eta_2(x_1'', x_2'', y')\big)$$

We use this method in our implementation of RELRECMC (see Sect. IV); it allows us to preserve compositionality in the relational lemma inference and significantly improves the speed of the evaluation in relational environments compared to the naive implementation.

IV. ALGORITHM

In this section, we formulate a property-directed algorithm for automatically inferring relational invariants. Our algorithm extends the RECMC algorithm [5]; from where 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 $P \in \mathcal{R}$ and a natural number b to a set of formulas over $\overline{v}_P$, and the latter maps a multiset $\overline{P} \in \mathbb{N}^{\mathcal{R}}$ and a natural number b to a set of formulas over $\overline{v}_{\overline{P}}$. Our algorithm maintains a bounded assertion map ρ and a relational bounded assertion map σ.

Our algorithm uses ρ to witness a counterexample to safety and σ to build a relational safety proof. In particular, ρ 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, σ 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, ρ and σ implicitly define the "classical" environment U_ρ^b and relational environment O_σ^b, respectively. The former under-approximates and the latter over-approximates the bounded semantics of the system:[1]

$$U_\rho^b(P) \stackrel{\text{def}}{=} \bigvee \{\delta \in \rho(P, c) \mid c \le b\}$$
$$O_\sigma^b(\overline{P}) \stackrel{\text{def}}{=} \bigwedge \big\{\delta \in \sigma(\overline{R}, c) \mid \overline{R} \in dom(\sigma), \overline{R} \subseteq \overline{P}, c \ge b\big\}$$

Note that lemmas for a multiset $\overline{P}$ subsume the lemmas of multisets included into $\overline{P}$.

We abbreviate $[\![\pi]\!]_{U_\rho^b}$ and $[\![\pi]\!]_{O_\sigma^b}$ to $[\![\pi]\!]_\rho^b$ and $[\![\pi]\!]_\sigma^b$ correspondingly. For simplicity, we define U_ρ^{-1} and O_σ^{-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 RELRECMC procedure that iteratively weakens reachability facts ρ and strengthens summary facts σ until either ρ witnesses a counterexample (line 4) or σ becomes inductive (line 12). An

[1]The conjunction of an empty set is $\top$, the disjunction is $\bot$.

ISBN: 978-0-9835678-9-9.

Algorithm 1: Pseudocode of RELRECMC

Input: Safety problem $\langle \mathcal{P}, \varphi_{safe} \rangle$
Output: $out \in \langle SAFE / UNSAFE$, relational proof, counterexample$\rangle$

1 $b \leftarrow 0$; $\rho \leftarrow \emptyset$; $\sigma \leftarrow \emptyset$;
2 **while** *true* **do**
3 $\quad \langle res, \rho, \sigma \rangle \leftarrow$ RELBNDSAFETY$(\langle \mathcal{P}, \varphi_{safe} \rangle, b, \rho, \sigma)$;
4 $\quad$ **if** $res = REACHABLE$ **then return** $\langle UNSAFE, \_, \rho \rangle$;
5 $\quad$ **else**
6 $\quad\quad inductive \leftarrow true$;
7 $\quad\quad$ **foreach** $\overline{P}$ *such that* $\langle \overline{P}, b \rangle \in dom(\sigma)$ **do**
8 $\quad\quad\quad$ **foreach** $\delta \in \sigma\left(\overline{P}, b\right)$ **do**
9 $\quad\quad\quad\quad$ **if** $\left[\!\left[body(\overline{P}) \right]\!\right]^{b}_{\sigma} \wedge \neg\delta \Rightarrow \bot$ **then**
10 $\quad\quad\quad\quad\quad \sigma \leftarrow \sigma \cup \{\langle \overline{P}, b+1 \rangle \mapsto \delta\}$;
11 $\quad\quad\quad\quad$ **else** $inductive \leftarrow false$;
12 $\quad\quad$ **if** $inductive$ **then return** $\langle SAFE, O^{b}_{\sigma}, \_ \rangle$;
13 $\quad\quad b \leftarrow b + 1$;

iteration b of RELRECMC checks if the property violation is reachable in b steps. If no bug is reachable (lines 6 - 11), σ contains a proof of bounded safety for b steps. RelRecMc then propagates all inductive lemmas from σ to level $b+1$ and iterates if they are not sufficient for concluding the safety.

C. Inner loop

The RELBNDSAFETY 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* $\langle \overline{P}, \pi, b \rangle$, where $\overline{P} \in \mathbb{N}^{\mathcal{R}}$, π is the negation of a safety property for $\overline{P}$, and $b \in \mathbb{N}$. Intuitively, to answer $\langle \overline{P}, \pi, b \rangle$, we determine if $\overline{P}$ does not reach π in b steps.

Queries are stored in a queue Q that initially contains only $\langle P_0, \neg\varphi_{safe}, B \rangle$. Each iteration begins with picking a query with the smallest b (line 3), 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 31). When all queries are answered, the algorithm returns the $(UN-)REACHABLE$ result (line 34 or 32, respectively).

a) Inference of reachability facts: If π is reachable in one step from predecessors bounded by $b-1$ steps (line 4), the algorithm deduces new reachability facts for every $\overline{P}[i]$ (line 7). Informally, instead of exploring all branches of $\overline{P}[i]$, the algorithm explores only a single branch $\psi_{\overline{P}[i]}$, chosen in a property-directed manner. Each query $\langle \overline{R}, \eta, c \rangle \in Q$, where η is reachable with the updated environment U^{c}_{ρ}, is answered and removed from Q (in particular, $\langle \overline{P}, \pi, b \rangle$).

To obtain a symbolic expression for a branch $\psi_{\overline{P}[i]}$, the algorithm uses a *model-based projection* (MBP) [5], [20]. Given a formula $\exists \overline{x}.\tau$, where τ is quantifier-free, and a model m, an $\mathrm{MBP}\,(\tau, \overline{x}, m)$ produces a quantifier-free conjunction of literals τ', such that $m \models \tau'$, $\tau' \Rightarrow \exists \overline{x}.\tau$, and if $\mathcal{M}$ admits quantifier elimination, then for each formula τ, there is a finite number of models $m_1, \ldots, m_k$, such that $\exists \overline{x}.\tau \Leftrightarrow \bigvee_{i=1}^{k} \mathrm{MBP}\,(\tau, \overline{x}, m_i)$. Intuitively, a series of model-based projections perform quantifier elimination from $\exists \overline{x}.\tau$ lazily. Given m, $\mathrm{MBP}\left(\left[\!\left[body(\overline{P}[i]) \right]\!\right]^{b-1}_{\rho}, \overline{\ell}_{\overline{P}[i]}, m\right)$ can be viewed as picking a branch of $\overline{P}[i]$, satisfied by m, and eliminating local variables $\overline{\ell}_{\overline{P}[i]}$ 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 $\exists \overline{x}.\tau$ 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.

Algorithm 2: Pseudocode of RELBNDSAFETY

Input: Safety problem $\langle \mathcal{P}, \varphi_{safe} \rangle$, level B, bounded assertion maps ρ, σ
Output: $out \in \langle REACHABLE / UNREACHABLE, \rho, \sigma \rangle$
Data: Queue of bounded reachability queries Q

1 $Q \leftarrow \{\langle P_0, \neg\varphi_{safe}, B \rangle\}$;
2 **while** $(Q \neq \emptyset)$ **do**
3 $\quad$ pick $\langle \overline{P}, \pi, b \rangle$ from Q;
4 $\quad$ **if** $\exists m \,.\, m \models \left[\!\left[body(\overline{P}) \right]\!\right]^{b-1}_{\rho} \wedge \pi$ **then**
5 $\quad\quad$ **for** $i \leftarrow 1$ **to** $\left|\overline{P}\right|$ **do**
6 $\quad\quad\quad \psi_{\overline{P}[i]} \leftarrow \mathrm{MBP}\left(\left[\!\left[body(\overline{P}[i]) \right]\!\right]^{b-1}_{\rho}, \overline{\ell}_{\overline{P}[i]}, m\right)$;
7 $\quad\quad\quad \rho \leftarrow \rho \cup \{\langle \overline{P}[i], b \rangle \mapsto \psi_{\overline{P}[i]} \mid 1 \le i \le n\}$;
8 $\quad\quad Q \leftarrow Q \setminus \{\langle \overline{R}, \eta, c \rangle \mid \overline{R} \subseteq \overline{P}, c \ge b, \bigwedge_{i=1}^{+|\overline{R}|} \psi_{\overline{R}[i]} \wedge \eta \not\Rightarrow \bot\}$;
9 $\quad$ **else if** $\left[\!\left[body(\overline{P}) \right]\!\right]^{b-1}_{\sigma} \wedge \pi \Rightarrow \bot$ **then**
10 $\quad\quad$ let ψ be s.t. $\left[\!\left[body(\overline{P}) \right]\!\right]^{b-1}_{\sigma} \Rightarrow \psi$ and $\psi \wedge \pi \Rightarrow \bot$;
11 $\quad\quad \sigma \leftarrow \sigma \cup \{\langle \overline{P}, b \rangle \mapsto \psi\}$;
12 $\quad\quad Q \leftarrow Q \setminus \left\{\langle \overline{R}, \eta, c \rangle \mid \overline{P} \subseteq \overline{R}, c \le b, \left[\!\left[\bigwedge_i \overline{R}[i](\overline{v}_{\overline{R}[i]}) \right]\!\right]^{c}_{\sigma} \wedge \eta \Rightarrow \bot\right\}$;
13 $\quad$ **else**
14 $\quad\quad$ let $cti \models \left[\!\left[body(\overline{P}) \right]\!\right]^{b-1}_{\sigma} \wedge \pi$;
15 $\quad\quad \psi \leftarrow \pi$;
16 $\quad\quad apps \leftarrow \emptyset$;
17 $\quad\quad$ **for** $i \leftarrow 1$ **to** $\left|\overline{P}\right|$ **do**
18 $\quad\quad\quad$ let $C \in rules(\overline{P}[i])$ be s.t. $cti \models \left[\!\left[body(C) \right]\!\right]^{b-1}_{\sigma}$;
19 $\quad\quad\quad \eta \wedge R_1(\overline{x}_1) \wedge \ldots \wedge R_m(\overline{x}_m) \leftarrow body(C)$;
20 $\quad\quad\quad \psi \leftarrow \psi \wedge \eta$;
21 $\quad\quad\quad$ **for** $j \leftarrow 1$ **to** m **do**
22 $\quad\quad\quad\quad$ **if** $cti \models \left[\!\left[R_j(\overline{x}_j) \right]\!\right]^{b-1}_{\rho}$ **then**
23 $\quad\quad\quad\quad\quad \psi \leftarrow \psi \wedge \left[\!\left[R_j(\overline{x}_j) \right]\!\right]^{b-1}_{\rho}$;
24 $\quad\quad\quad\quad$ **else** $apps \leftarrow apps \cup \{R_j(\overline{x}_j)\}$;
25 $\quad\quad Groups \leftarrow$ PARTITION$(apps, \pi, \psi)$;
26 $\quad\quad$ **for** $\{i_1, \ldots, i_k\} \in Groups$ **do**
27 $\quad\quad\quad rels \leftarrow \langle R_{i_1}, \ldots, R_{i_k} \rangle$;
28 $\quad\quad\quad vars \leftarrow \overline{x}_{i_1} \ldots \overline{x}_{i_k}$;
29 $\quad\quad\quad \psi' \leftarrow \psi \wedge \left[\!\left[\bigwedge_{\substack{R_j(\overline{x}_j) \in apps \\ j \notin \{i_1, \ldots, i_k\}}} R_j(\overline{x}_j) \right]\!\right]^{b-1}_{\sigma}$;
30 $\quad\quad\quad \psi' \leftarrow \mathrm{MBP}\left(\psi', \overline{v}_{\overline{P}} \cup \overline{\ell}_{\overline{P}} \setminus vars, cti\right)$;
31 $\quad\quad\quad Q \leftarrow Q \cup \{\langle rels, \psi'[vars \leftarrow \overline{v}_{rels}], b-1 \rangle\}$;
32 **if** $\left[\!\left[P_0 \right]\!\right]^{n}_{\rho} \wedge \neg\varphi_{safe} \not\Rightarrow \bot$ **then return** $\langle REACHABLE, \rho, \sigma \rangle$;
33 **assert** $\left[\!\left[P_0 \right]\!\right]^{n}_{\sigma} \wedge \neg\varphi_{safe} \Rightarrow \bot$;
34 **return** $\langle UNREACHABLE, \rho, \sigma \rangle$;

b) Inference of summary facts: If σ is strong enough to prove the unreachability of π (line 9), RELBNDSAFETY infers a new lemma by computing a Craig interpolant (denoted later ITP) of $\left[\!\left[body(\overline{P}) \right]\!\right]^{b-1}_{\sigma}$ and $\neg\pi$. As a result, the new lemma over-approximates the b-bounded semantics of P, still proving its safety relatively to π. Note that the resulting lemma is formulated over the variables of $\overline{P}$, expressing *relations* among elements of $\overline{P}$. Every query $\langle \overline{R}, \eta, c \rangle \in Q$, such that the updated σ proves the unsatisfiability of η, is immediately an-

swered and removed from the queue (in particular, $\langle \overline{P}, \pi, b \rangle$).

c) Query generation: If neither $[\![body(\overline{P})]\!]_{\rho}^{b-1} \wedge \pi$ is satisfiable, nor $[\![body(\overline{P})]\!]_{\sigma}^{b-1} \wedge \pi$ 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 14, which should be witnessed by ρ or blocked by σ. The algorithm generates child queries, answers to which would help answering $\langle \overline{P}, \pi, b \rangle$ by either blocking the CTI or proving its reachability.

For each relation in $\overline{P}$, the algorithm picks a clause C witnessing cti (line 18). Such clause is guaranteed to exist because $cti \models [\![body(\overline{P})]\!]_{\sigma}^{b-1}$. Let $R_1(\overline{x}_1), \ldots, R_m(\overline{x}_m)$ be all applications of relational symbols in C (line 19). In the next steps 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 $R_1(\overline{x}_1), \ldots, R_m(\overline{x}_m)$ 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 $\{f(\overline{x_1}), f(\overline{x_2}), g(\overline{y})\}$ could be split into the list of multisets $[\{f \mapsto 2\}, \{g \mapsto 1\}]$ of relations and the list of vectors the $[\langle \overline{x}_1, \overline{x}_2 \rangle, \overline{y}]$ 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[2]. If the recursive group has more than $|\overline{P}|$ elements, it gets partitioned into the groups of size at most $|\overline{P}|$. 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 π (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 partitioning (line 29). Intuitively, RELBNDSAFETY strengthens the safety property π 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.

[2]Two symbols are (mutually) recursive if they belong to the same strongly connected component in the directed graph (V, E) with $V = \mathcal{R}$ and $(P, R) \in E$ iff R occurs in the body of some rule for P.

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 $n = 0 \wedge \bot$ and P_0 instead of the multiset $\{P_0 \mapsto 1\}$.

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

$$
\begin{aligned}
T_1 = leaf \wedge n = 0 &\Rightarrow size(T_1, n)\\
T_1 = node(v_1, L_1, R_1) \wedge n = 1 + n^L + n^R \wedge & \\
size(L_1, n^L) \wedge size(R_1, n^R) &\Rightarrow size(T_1, n)\\
T_2 = leaf \wedge s_2 = 0 &\Rightarrow sum(T_2, s_2)\\
T_2 = node(v_2, L_2, R_2) \wedge s_2 = v_2 + s_2^L + s_2^R \wedge & \\
sum(L_2, s_2^L) \wedge sum(R_2, s_2^R) &\Rightarrow sum(T_2, s_2)\\
T_4 = leaf \wedge U = leaf &\Rightarrow inc(T_4, U)\\
T_4 = node(v_4, L_4, R_4) \wedge U = node(v_4 + 2, L', R') \wedge & \\
inc(L_4, L') \wedge inc(R_4, R') &\Rightarrow inc(T_4, U)\\
A = T_0 \wedge B = T_0 \wedge C = T_0 \wedge D = E \wedge size(A, n_0) \wedge & \\
sum(B, s_0) \wedge inc(C, D) \wedge sum(E, s_0') &\Rightarrow P_0(T_0, n_0, s_0, s_0')\\
\varphi_{safe} &\stackrel{\text{def}}{=} s_0' = s_0 + 2n_0
\end{aligned}
$$

a) Level 0*:* The algorithm begins with calling RELBNDSAFETY for level 0 that puts query $\langle P_0, s_0' \neq s_0 + 2n_0, 0 \rangle$ into Q. Both $[\![body(P_0)]\!]_{\rho}^{-1}$ and $[\![body(P_0)]\!]_{\sigma}^{-1}$ are $\bot$, so the algorithm gets into line 11, where it adds $\mathrm{ITP}(\bot, \neg\varphi_{safe}) = \bot$ into $\sigma(P_0, 0)$. RELBNDSAFETY terminates with the result $UNREACHABLE$, but since the added lemma is not inductive, RELRECMC proceeds to level 1.

b) Level 1*:* The RELBNDSAFETY algorithm begins with $Q = \{\langle P_0, s_0' \neq s_0 + 2n_0, 1 \rangle\}$. Here, $[\![body(P_0)]\!]_{\rho}^{0} \equiv \bot$ and $[\![body(P_0)]\!]_{\sigma}^{0} \equiv \varphi_0 \stackrel{\text{def}}{=} A = T_0 \wedge B = T_0 \wedge C = T_0 \wedge D = E$. Since $\varphi_0 \wedge \neg\varphi_{safe}$ is satisfiable, the algorithm extracts $cti = \{A, B, C, D, E, T_0 \mapsto leaf; n_0 \mapsto 1; s_0 \mapsto 0; s_0' \mapsto 1\}$ and goes to line 14. Then it picks the only possible rule for P_0 with the body $\varphi_0 \wedge size(A, n_0) \wedge sum(B, s_0) \wedge inc(C, D) \wedge sum(E, s_0')$. Now $cti \not\models [\![size(A, n_0)]\!]_{\rho}^{0} \equiv A = leaf \wedge n_0 = 0$, $cti \models [\![sum(B, s_0)]\!]_{\rho}^{0}$, $cti \models [\![inc(C, D)]\!]_{\rho}^{0}$, $cti \not\models [\![sum(E, s_0')]\!]_{\rho}^{0}$, so we get $apps = \{size(A, n_0), sum(E, s_0')\}$ and $\psi \equiv \neg\varphi_{safe} \wedge \varphi_0 \wedge B = leaf \wedge s_0 = 0 \wedge C = leaf \wedge D = leaf$. Since both inc and sum are non-recursive with P_0, the PARTITION oracle does nothing: $\mathrm{PARTITION}(apps) = \{\{1, 4\}\}$.

Let $\alpha_1 = \{size \mapsto 1, sum \mapsto 1\}$. To obtain the child bounded reachability query for α_1, the algorithm projects away all variables from ψ except A, n_0, E, and s_0', obtaining

$\psi' \equiv \text{MBP}(\psi, \{T_0, B, C, D, s_0\}, cti) \equiv A = leaf \wedge E = leaf \wedge s_0' \neq 2n_0$, which becomes after renaming $\psi_1 \stackrel{\text{def}}{=} T_1 = leaf \wedge T_2 = leaf \wedge s_2 \neq 2n$. Thus, Q becomes $\{\langle P_0, \neg\varphi_{safe}, 1\rangle, \langle \alpha_1, \psi_1, 0\rangle\}$, and the algorithm iterates.

At the second iteration, the algorithm picks a query $\langle \alpha_1, \psi_1, 0\rangle$ as the one having the minimal level. Let $\beta_1 \stackrel{\text{def}}{=} [\![body(\alpha_1)]\!]_\rho^{-1} \equiv [\![body(\alpha_1)]\!]_\sigma^{-1} \equiv T_1 = leaf \wedge n = 0 \wedge T_2 = leaf \wedge s_2 = 0$. Since $\beta_1 \wedge \psi_1$ is unsatisfiable, the algorithm derives a new lemma $\delta_1 \stackrel{\text{def}}{=} \text{ITP}(\beta_1, \psi_1) \equiv T_1 = T_2 = leaf \wedge n = s_2 = 0$. Thus, $\sigma(\alpha_1, 0) = \{\delta_1\}$, and the query at level 0 is answered and removed from Q.

At the third iteration, the algorithm picks a query $\langle P_0, \neg\varphi_{safe}, 1\rangle$ again. This time, $[\![body(P_0)]\!]_\sigma^0 \equiv \varphi_0 \wedge \delta_1(A, n, B, s_0) \wedge \delta_1(A, n, E, s_0')$. Since now $[\![body(P_0)]\!]_\sigma^0 \wedge \neg\varphi_{safe}$ is unsatisfiable, $\sigma(P_0, 1)$ is updated to $\text{ITP}([\![body(P_0)]\!]_\sigma^0, \neg\varphi_{safe}) \equiv \varphi_{safe}$. The new environment is not inductive, so RELRECMC proceeds to level 2.

c) Level 2: Query $\langle P_0, \neg\varphi_{safe}, 2\rangle$ is picked from Q. Now, $cti = \{A, B, C, T_0 \mapsto node(0, leaf, leaf); D, E \mapsto leaf; n_0, s_0, s_0' \mapsto 1\}$. As $cti \not\models [\![size(A, n_0)]\!]_\rho^1$, $cti \not\models [\![sum(B, s_0)]\!]_\rho^1$, $cti \not\models [\![inc(C, D)]\!]_\rho^1$, $cti \not\models [\![sum(E, s_0')]\!]_\rho^1$, we get $apps = \{size(A, n_0), sum(B, s_0), inc(C, D), sum(E, s_0')\}$. As none of the symbols is recursive with P_0, we get $Groups = \{\{1, 2, 3, 4\}\}$. To get the child safety property, the algorithm projects away T_0 and iterates with $\{\langle P_0, \neg\varphi_{safe}, 2\rangle, \langle \alpha_2, \psi_2, 1\rangle\}$, where $\alpha_2 \stackrel{\text{def}}{=} \{size \mapsto 1, sum \mapsto 2, inc \mapsto 1\}$ and $\psi_2 \stackrel{\text{def}}{=} T_1 = T_2 = T_4 \wedge U = T_3 \wedge s_3 \neq s_2 + 2n$.

At the next iteration, $\langle \alpha_2, \psi_2, 1\rangle$ is picked from Q. RelRecMc applies the technique described in Sect. III-D for the fast evaluation in σ. Let

$$\begin{aligned}
\beta_{size} &\stackrel{\text{def}}{=} (T_1 = leaf \wedge n = 0) \vee \\
&\quad (T_1 = node(v_1, L_1, R_1) \wedge n = 1 + n^L + n^R \wedge a_L \wedge a_R) \\
\beta_{sum_1} &\stackrel{\text{def}}{=} (T_2 = leaf \wedge s_2 = 0) \vee \\
&\quad (T_2 = node(v_2, L_2, R_2) \wedge s_2 = v_2 + s_2^L + s_2^R \wedge b_L \wedge b_R) \\
\beta_{sum_2} &\stackrel{\text{def}}{=} (T_3 = leaf \wedge s_3 = 0) \vee \\
&\quad (T_3 = node(v_3, L_3, R_3) \wedge s_3 = v_3 + s_3^L + s_2^R \wedge c_L \wedge c_R) \\
\beta_{inc} &\stackrel{\text{def}}{=} (T_4 = leaf \wedge U = leaf) \vee \\
&\quad (T_4 = node(v_4, L_4, R_4) \wedge U = node(v_4 + 2, L', R') \wedge d_L \wedge d_R)
\end{aligned}$$

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:

$$\begin{aligned}
[\![body(\alpha_2)]\!]_\sigma^0 \equiv\ & \beta_{size} \wedge \beta_{sum_1} \wedge \beta_{sum_2} \wedge \beta_{inc} \wedge \\
& (a_L \wedge b_L \Rightarrow \delta_1(L_1, n_L, L_2, s_2^L)) \wedge \\
& (a_L \wedge b_R \Rightarrow \delta_1(L_1, n_L, R_2, s_2^R)) \wedge \\
& (a_L \wedge c_L \Rightarrow \delta_1(L_1, n_L, L_3, s_3^L)) \ldots
\end{aligned}$$

If instead we straightforwardly convert the $[\![body(\alpha_2)]\!]_\sigma^0$ into DNF and replace each possible combination of relational applications with δ_1, we would get 2^4 times larger formula.

$[\![body(\alpha_2)]\!]_\sigma^0 \wedge \psi_2$ is satisfiable, and $cti = \{T_k \mapsto node(1, leaf, leaf); T_3, U \mapsto node(3, leaf, leaf); \ldots\}$ for $k \in \{1, 2, 4\}$. Thus at line 18, the algorithm picks the second (recursive) rule for each relation in $\mathcal{R}$.

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 $\psi \equiv \psi_2 \wedge T_1 = node(v_1, L_1, R_1) \wedge n = 1 + n^L + n^R \wedge \ldots$.

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 $C_8^4 = 70$ 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 ψ_2 is $\{T_1 = T_2, T_1 = T_4, U = T_3, s_3 \neq s_2 + 2n\}$. For example, $T_1 \wedge T_2$ is inductive relatively to $size(R_1, n^R)$ and $sum(R_2, s_2^R)$. To verify this, we rename $T_1 = T_2$ to $R_1 = R_2$ (as T_1, R_1 and T_2, R_2 are the first arguments in applications of respectively $size$ and sum), and check $\psi \Rightarrow R_1 = R_2$. In our case, the whole ψ_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 α_2 at level 0 are added into Q.

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

E. General properties

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

Theorem 2. RELRECMC *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, RELRECMC 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, RELRECMC 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.

ISBN: 978-0-9835678-9-9.

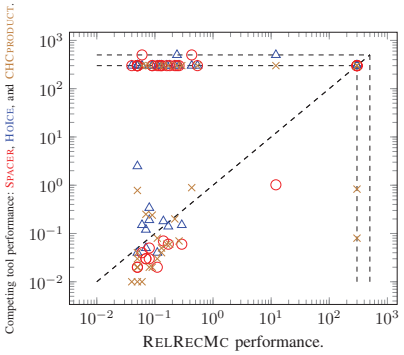

Figure 1: RELRECMC vs competitors. Each point in a plot represents a pair of the run times (sec × sec) of RELRECMC (x-axis) and a competitor (y-axis). Timeouts are placed on the inner dashed lines; and crashes are on the outer dashed lines.

While in some cases the RECMC algorithm [5] fails to infer a safety proof for nonlinear systems simply because every model of the system is undefinable in the assertion language, our algorithm manages to infer a relational safety proof. Conversely, whenever the RECMC algorithm succeeds to prove or disprove the safety, our algorithm succeeds as well.

V. EVALUATION

We have implemented our algorithm on the top of SPACER, a state-of-the-art CHC solving engine in Z3 SMT-solver [22][3]. We have evaluated the implementation against SPACER and the HOICE tool [8] on two benchmark suites[4]. 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 RELRECMC 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 RELRECMC solved 32 out of 37 problems. Note that RELRECMC 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.

[3]The implementation is available at https://github.com/dvvrd/z3.

[4]Benchmarks are available at https://github.com/dvvrd/spacer-benchmarks.

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.

ISBN: 978-0-9835678-9-9.

REFERENCES

[1] N. Eén, A. Mishchenko, and R. K. Brayton, "Efficient implementation of property directed reachability," in *FMCAD*. IEEE, 2011, pp. 125–134.

[2] K. Hoder and N. Bjørner, "Generalized property directed reachability," in *SAT*, ser. LNCS, vol. 7317. Springer, 2012, pp. 157–171.

[3] A. Cimatti, A. Griggio, S. Mover, and S. Tonetta, "IC3 modulo theories via implicit predicate abstraction," in *TACAS*, ser. LNCS, vol. 8413. Springer, 2014, pp. 46–61.

[4] K. L. McMillan, "Lazy annotation revisited," in *CAV*, ser. LNCS, vol. 8559. Springer, 2014, pp. 243–259.

[5] A. Komuravelli, A. Gurfinkel, and S. Chaki, "SMT-Based Model Checking for Recursive Programs," in *CAV*, ser. LNCS, vol. 8559, 2014, pp. 17–34.

[6] D. Jovanovic and B. Dutertre, "Property-directed k-induction," in *FMCAD*. IEEE, 2016, pp. 85–92.

[7] G. Fedyukovich and R. Bodík, "Accelerating Syntax-Guided Invariant Synthesis," in *TACAS, Part I*, ser. LNCS, vol. 10805. Springer, 2018, pp. 251–269.

[8] A. Champion, N. Kobayashi, and R. Sato, "HoIce: An ICE-Based Non-linear Horn Clause Solver," in *Asian Symposium on Programming Languages and Systems*. Springer, 2018, pp. 146–156.

[9] S. K. Lahiri, K. L. McMillan, R. Sharma, and C. Hawblitzel, "Differential assertion checking," in *FSE*. ACM, 2013, pp. 345–355.

[10] J. B. Almeida, M. Barbosa, G. Barthe, F. Dupressoir, and M. Emmi, "Verifying constant-time implementations," in *USENIX*. USENIX Association, 2016, pp. 53–70.

[11] M. Kiefer, V. Klebanov, and M. Ulbrich, "Relational program reasoning using compiler IR," in *VSTTE*, ser. LNCS, vol. 9971. Springer, 2016, pp. 149–165.

[12] B. Beckert, T. Bingmann, M. Kiefer, P. Sanders, M. Ulbrich, and A. Weigl, "Relational Equivalence Proofs Between Imperative and MapReduce Algorithms," ser. LNCS, vol. 11294. Springer, 2018, pp. 248–266.

[13] G. Barthe, J. M. Crespo, and C. Kunz, "Relational verification using product programs," in *FM*, ser. LNCS, vol. 6664. Springer, 2011, pp. 200–214.

[14] D. Felsing, S. Grebing, V. Klebanov, P. Rümmer, and M. Ulbrich, "Automating regression verification," in *ASE*. ACM, 2014, pp. 349–360.

[15] E. De Angelis, F. Fioravanti, A. Pettorossi, and M. Proietti, "Relational Verification Through Horn Clause Transformation," in *SAS*, ser. LNCS, vol. 9837, 2016, pp. 147–169.

[16] D. Mordvinov and G. Fedyukovich, "Synchronizing Constrained Horn Clauses," in *LPAR*, ser. EPiC Series in Computing, vol. 46. EasyChair, 2017, pp. 338–355.

[17] R. Shemer, A. Gurfinkel, S. Shoham, and Y. Vizel, "Property directed self composition," in *CAV, Part I*, vol. 11561. Springer, 2019, pp. 161–179.

[18] E. M. Clarke, "Program invariants as fixedpoints," *Computing*, vol. 21, no. 4, pp. 273–294, 1979.

[19] K. R. Apt *et al.*, *From logic programming to Prolog*. Prentice Hall London, 1997, vol. 362.

[20] N. Bjørner and M. Janota, "Playing with quantified satisfaction." *LPAR (short papers)*, vol. 35, pp. 15–27, 2015.

[21] A. R. Bradley, "SAT-Based Model Checking without Unrolling," in *VMCAI*, ser. LNCS, vol. 6538. Springer, 2011, pp. 70–87.

[22] L. M. de Moura and N. Bjørner, "Z3: An Efficient SMT Solver," in *TACAS*, ser. LNCS, vol. 4963. Springer, 2008, pp. 337–340.

[23] D. Mordvinov and G. Fedyukovich, "Verifying Safety of Functional Programs with Rosette/Unbound," *CoRR*, vol. abs/1704.04558, 2017, https://github.com/dvvrd/rosette.

[24] O. Strichman and M. Veitsman, "Regression verification for unbalanced recursive functions," in *FM*, ser. LNCS, vol. 9995, 2016, pp. 645–658.

[25] M. Sousa and I. Dillig, "Cartesian Hoare Logic for verifying k-safety properties," in *PLDI*. ACM, 2016, pp. 57–69.

[26] L. Pick, G. Fedyukovich, and A. Gupta, "Exploiting Synchrony and Symmetry in Relational Verification," in *CAV, Part I*, ser. LNCS, vol. 10981. Springer, 2018, pp. 164–182.

ISBN: 978-0-9835678-9-9.