Property Directed Inference of Relational Invariants

Dmitry Mordvinov\textsuperscript{1} \hspace{1cm} Grigory Fedyukovich\textsuperscript{2}

\textsuperscript{1}JetBrains Research, Saint Petersburg State University, Russia

\textsuperscript{2}Florida State University, USA

FMCAD, Oct 25, 2019, San Jose, CA
Motivation

- Verification of program safety
- Constrained Horn Clauses (CHC):
  - Representation of a wide range of programs and assertions
- Model checker = CHC-translation + CHC-solving
  - E.g., SeaHorn + Spacer, or JayHorn + Eldarica
- Invariant generation for CHC-solving:
  - Construct proofs iteratively using SMT solvers
- Relational verification
  - Program equivalence,
  - Non-interference, ...
Constrained Horn Clauses

Constrained Horn Clause (CHC):

\[ \varphi \land f_1(\overline{x}_1) \land \ldots \land f_n(\overline{x}_n) \rightarrow H \]

- \( \varphi \) — constraint (quantifier-free formula in first-order constraints language)
- \( f_i \) — uninterpreted predicate symbol
- either \( H = \bot \) or \( H = f(\overline{x}) \) for some \( f \)
- \( \overline{x}_i, \overline{x} \) — vectors of (distinct) variables
- \( H \) is a head of clause (\textit{head}(C))
- the premise of the implication is the body of the clause
Systems of Constrained Horn Clauses

- **CHC system** $S$ is an arbitrary set of clauses
- Clauses with heads $\bot$ are called **queries**, otherwise **rules**
  - **Rules** define the “semantics” of uninterpreted symbols
  - **Queries** define the specification
  - Queries can be rewritten to $f_1(\overline{x}_1) \land \ldots \land f_n(\overline{x}_n) \rightarrow P$
- A set of rules for uninterpreted symbol $f$ is a set of clauses with applications of $P$ in the head
- $\text{body}(f)$ is the disjunction of bodies of rules for $f$
Linear and Nonlinear CHCs

• CHC is called **linear**, if it has at most one application of uninterpreted symbol in its body
• Otherwise it is **nonlinear**

**Linear:**

\[ \varphi \land f(x) \rightarrow h(x) \]

**Nonlinear:**

\[ \varphi \land f(x) \land g(x) \rightarrow h(x, y) \]
Linear Queries

Linear query is a clause of the form

\[ f(\bar{x}) \rightarrow P(\bar{x}) \]

Uninterpreted symbol  Safety property

To answer the query, we should find the subset of \( P(\bar{x}) \), which is inductive for \( f \)
Property Directed Reachability

“SAT-based model checking without unrolling” (Bradley, 2011)
“Efficient Implementation of Property Directed Reachability” (Eén et al., 2011)

\[ P(\overline{x}) \]

Constrained facts
(initial states)
Property Directed Reachability

“SAT-based model checking without unrolling” (Bradley, 2011)
“Efficient Implementation of Property Directed Reachability” (Eén et al., 2011)

Constrained facts
(initial states)

\[ P(\overline{x}) \]
Property Directed Reachability

“SAT-based model checking without unrolling” (Bradley, 2011)
“Efficient Implementation of Property Directed Reachability” (Eén et al., 2011)
Property Directed Reachability

“SAT-based model checking without unrolling” (Bradley, 2011)
“Efficient Implementation of Property Directed Reachability” (Eén et al., 2011)
Property Directed Reachability

“SAT-based model checking without unrolling” (Bradley, 2011)
“Efficient Implementation of Property Directed Reachability” (Eén et al., 2011)
Property Directed Reachability

“SAT-based model checking without unrolling” (Bradley, 2011)
“Efficient Implementation of Property Directed Reachability” (Eén et al., 2011)
Property Directed Reachability

“SAT-based model checking without unrolling” (Bradley, 2011)
“Efficient Implementation of Property Directed Reachability” (Eén et al., 2011)
Property Directed Reachability

“SAT-based model checking without unrolling” (Bradley, 2011)
“Efficient Implementation of Property Directed Reachability” (Eén et al., 2011)
PDR for Linear CHC Systems

- Start with root query $f(\bar{x}) \rightarrow P$
- Get counterexample to inductiveness (CTI) $s$
- Detect a prime implicant of $\text{body}(f)$ satisfied by $s$
- As the system is linear, the prime implicant is linear as well!
  - Let it be $\psi \land g(\bar{y})$
- Use $s$ form a new query $P'$
  - Project away all variables except $\bar{y}$ from $\psi \rightarrow P$
PDR for Linear CHC Systems

- Start with root query $f(\overline{x}) \rightarrow P$
- Get counterexample to inductiveness (CTI) $s$
- Detect a prime implicant of $\text{body}(f)$ satisfied by $s$
- As the system is linear, the prime implicant is linear as well!
  - Let it be $\psi \land g(\overline{y})$
- Use $s$ form a new query $P'$
  - Project away all variables except $\overline{y}$ from $\psi \rightarrow P$

Thus we (recursively) turn the linear safety problem

$$f(\overline{x}) \rightarrow P$$

to a (simpler) linear subproblem

$$g(\overline{y}) \rightarrow P'$$

At the previous level
PDR for Linear CHC Systems

Given a CHC system, PDR terminates, if

- CTI satisfies some constrained fact
  - then the system is unsafe
- Summary facts at two subsequent levels are identical
  - then the system is safe
  - summary facts form a safety proof $\mathcal{I}$
  - $\mathcal{I}$ maps each uninterpreted symbol $f$ to a first-order formula
PDR for Linear CHC Systems

Given a CHC system, PDR terminates, if

- CTI satisfies some constrained fact
  - then the system is unsafe
- Summary facts at two subsequent levels are identical
  - then the system is safe
  - summary facts form a safety proof $I$
  - $I$ maps each uninterpreted symbol $f$ to a first-order formula

Linear problems are easier for modern PDR implementations (and other CHC solvers) than nonlinear
Results: LIA-Lin

<table>
<thead>
<tr>
<th>Solver</th>
<th>Score</th>
<th>#SAT</th>
<th>#UNSAT</th>
<th>Avg time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Spacer</td>
<td>279</td>
<td>194</td>
<td>85</td>
<td>28.90</td>
</tr>
<tr>
<td>Rebus</td>
<td>267</td>
<td>188</td>
<td>79</td>
<td>41.85</td>
</tr>
<tr>
<td>Eldarica</td>
<td>209</td>
<td>129</td>
<td>80</td>
<td>24.55</td>
</tr>
<tr>
<td>Ultimate Unihorn Automizer</td>
<td>133</td>
<td>63</td>
<td>70</td>
<td>23.05</td>
</tr>
<tr>
<td>Hoice</td>
<td>129</td>
<td>65</td>
<td>64</td>
<td>7.09</td>
</tr>
<tr>
<td>Ultimate Tree Automizer</td>
<td>107</td>
<td>42</td>
<td>65</td>
<td>29.15</td>
</tr>
<tr>
<td>PCSat</td>
<td>45</td>
<td>33</td>
<td>12</td>
<td>23.74</td>
</tr>
</tbody>
</table>

* 325 instances total
Nonlinear Problems

Nonlinear CHCs naturally appear in various verification problems

- Verification of programs with multiple procedures

**Program:**

1. `int x = f(n);`
2. `x = g(x);`
3. `assert(P(n,x));`

**Verification conditions:**

- `... → f(n,x)`
- `... → g(x,x')`
- `f(n,x) ∧ g(x,x') → P(n,x')`
Nonlinear Problems

Nonlinear CHCs naturally appear in various verification problems

- Relational verification
  
  “Relational Verification Using Product Programs”
  (Barthe et al., 2011)

Goal: prove that a relational specification \((pre, post)\) holds for \(k\) programs
Nonlinear Problems

Nonlinear CHCs naturally appear in various verification problems

- Relational verification
  “Relational Verification Using Product Programs” (Barthe et al., 2011)

Given programs $f_1$ and $f_2$, that respectively have inputs $\overline{x_1}, \overline{x_2}$ and outputs $\overline{y_1}, \overline{y_2}$,
prove $\overline{x_1} = \overline{x_2} \Rightarrow \overline{y_1} = \overline{y_2}$

Example: program equivalence
Nonlinear Problems: Undefinability

Consider the following nonlinear CHC system in Linear Integer Arithmetic (LIA) ($\mathbb{Z} \cup \{\leq, +, -\}$):

\[
\begin{align*}
\text{mul}(x, y, z) & \leftarrow x = 0 \land z = 0 \\
\text{mul}(x, y, z) & \leftarrow x > 0 \land x' = x - 1 \land z = z' + y \land \text{mul}(x', y, z') \\
\bot & \leftarrow x = x' \land y = y' \land \text{mul}(x, y, z) \land \text{mul}(x', y', z') \land z \neq z'
\end{align*}
\]
Nonlinear Problems: Undefinability

Consider the following nonlinear CHC system in Linear Integer Arithmetic (LIA) ($\mathbb{Z} \cup \{\leq, +, -\}$):

\[
\begin{align*}
\text{mul}(x, y, z) & \leftarrow x = 0 \land z = 0 \\
\text{mul}(x, y, z) & \leftarrow x > 0 \land x' = x - 1 \land z = z' + y \land \text{mul}(x', y, z') \\
\bot & \leftarrow x = x' \land y = y' \land \text{mul}(x, y, z) \land \text{mul}(x', y', z') \land z \neq z'
\end{align*}
\]

Although the system is safe, no safety proof is definable in linear integer arithmetic!

- The obvious invariant

\[
\mathcal{I} = \{ \text{mul} \mapsto (x \geq 0 \land y \geq 0 \rightarrow z = x \cdot y) \}
\]

is undefinable in LIA!
Traditional Safety Proofs

The problem is that “classical” safety proofs are not expressive enough for nonlinear systems:

\[ \mathcal{I}(f) \equiv F(\bar{x}) \]
\[ \mathcal{I}(g) \equiv G(\bar{y}) \]

For nonlinear bodies like \( f(\bar{x}) \land g(\bar{y}) \rightarrow H \), traditional interpretations define only **Cartesian** relations.
Traditional vs Relational Proofs

Problem: $f(\overline{x}) \land g(\overline{y}) \rightarrow P(\overline{x}, \overline{y})$

- $P(x, y)$ — safety property
- $F(\overline{x})$ is the invariant for $f$
- $G(\overline{y})$ is the invariant for $g$

$I(f) \equiv F(\overline{x})$
$I(g) \equiv G(\overline{y})$

Not too expressive proofs!
Problem: \( f(x) \land g(y) \rightarrow P(x, y) \)

- \( P(x, y) \) — safety property
- \( H(x, y) \) is the invariant for product program \( f \times g \)
Relational Proofs

- Symbolic relational proofs interpret multisets of uninterpreted symbols instead of singleton symbols
- But they should be an invariant for product programs!
Relational Proofs

- **Symbolic relational proofs** interpret multisets of uninterpreted symbols instead of singleton symbols.
- But they should be an invariant for product programs!

\[
\begin{align*}
\text{mul}(x, y, z) & \leftarrow x = 0 \land z = 0 \\
\text{mul}(x, y, z) & \leftarrow x > 0 \land x' = x - 1 \land z = z' + y \land \text{mul}(x', y, z') \\
\bot & \leftarrow x = x' \land y = y' \land \text{mul}(x, y, z) \land \text{mul}(x', y', z') \land z \neq z'
\end{align*}
\]
Relational Proofs

- Symbolic relational proofs interpret multisets of uninterpreted symbols instead of singleton symbols.
- But they should be an invariant for product programs!

\[
\begin{align*}
\text{mul}(x, y, z) & \leftarrow x = 0 \land z = 0 \\
\text{mul}(x, y, z) & \leftarrow x > 0 \land x' = x - 1 \land z = z' + y \land \text{mul}(x', y, z') \\
\bot & \leftarrow x = x' \land y = y' \land \text{mul}(x, y, z) \land \text{mul}(x', y', z') \land z \neq z'
\end{align*}
\]

This system does not have “traditional” definable in LIA, but has a relational one:

\[
\mathcal{J} = \left\{ \begin{array}{c}
\text{mul} \\
\langle \text{mul}, \text{mul} \rangle
\end{array} : \begin{array}{c}
\top \\
(x_1 = x_2 \land y_1 = y_2 \rightarrow z_1 = z_2)
\end{array} \right\}
\]
Relational Proofs

Let $S$ be a CHC system

- If $S$ has a relational proof, then it is safe
- If $S$ has a definable “traditional” safety proof, then it has a relational safety proof as well
- The converse is not true!
- Both forms of proofs evaluate linear systems identically
  - The relational invariants are useful only for nonlinear systems!
Problem: \( f(x) \land g(y) \rightarrow P(x, y) \)

“SMT-based model checking for recursive programs” (Komuravelli et al., 2016)

use lemmas from previous levels to **split** the safety property

- Spawns **two** proof obligations \((f, \text{level}, P_1)\) and \((g, \text{level}, P_2)\)
  
  - \([P_1(x)] \equiv \exists y.\sigma(g, \text{level} - 1)(y) \rightarrow P(x, y)\)
  
  - \([P_2(y)] \equiv \exists x.\sigma(f, \text{level} - 1)(x) \rightarrow P(x, y)\)

- If no model of a safe system is definable, never terminates
**PDR vs Relational PDR**

**Problem**: \( f(\overline{x}) \land g(\overline{y}) \rightarrow P(\overline{x}, \overline{y}) \)

Our algorithm partitions the predicates into the groups, tending to obtain the linear problems, but for the relational specifications

- Tries to answer the relational proof obligation

\[ (\langle f, g \rangle, \text{level}, P(\overline{x}, \overline{y})) \]

- Detects and blocks CTIs in product program \( f \times g \)
- Doesn’t explicitly construct a product program
- Obtains relational lemmas by interpolation
To Merge or Not to Merge?

- Let $s$ be a CTI at some level for a nonlinear problem.
- Unlike the linear case, some predicates may reach the projection of $s$, while some may not:

![Diagram showing reachable states of $f$ and lemmas for $g$.]
To Merge or Not to Merge?

• If some predicate reaches the projection of CTI $s$, then there is no need to strengthen its lemmas
  • We can’t exclude the point anyway
• Thus all predicates witnessing $s$ are not “merged”
  • Instead, we use the reachability information to weaken the safety property
• Different CTI may result in different groups of predicates
  • But larger groups “subsume” lemmas of smaller groups
• The procedure was implemented in SPACER kernel
  • The relational generalization of PDR
  • Given constraints satisfiability oracle, the algorithm is a co-decision procedure for safety and complete for the systems with finite state space
Experiments: HOICE benchmarks

“HoIce: An ICE-Based Non-linear Horn Clause Solver” (Champion et al., 2018)

840 tests, different sources (mostly generated by MoCHi)

<table>
<thead>
<tr>
<th>The amount of tests</th>
<th>HoICE</th>
<th>Spacer</th>
<th>RelSpacer</th>
</tr>
</thead>
<tbody>
<tr>
<td>840</td>
<td>808</td>
<td>788</td>
<td>807</td>
</tr>
</tbody>
</table>

- Small timeout (30 sec.)
- \texttt{Spacer} has more timeouts than \texttt{HoICE} on nonlinear benchmarks
- Our implementation is more competitive, outperforming (timewise) \texttt{HoICE} on solved benchmarks
Experiments: Relational Benchmarks

“Synchronizing Constrained Horn Clauses” (Mordvinov et al., 2017)

- 37 relational problems
- A lot of problems with undefinable «classical» invariants
- New competitor: \texttt{CHCProduct} implemented as a syntactical preprocessing of clauses in \texttt{Z3}, then solving with \texttt{SPACER}
- Timeout: 5 min.

<table>
<thead>
<tr>
<th>Total</th>
<th>\texttt{HOICE}</th>
<th>\texttt{SPACER}</th>
<th>\texttt{CHCProduct}</th>
<th>\texttt{RELSPACER}</th>
</tr>
</thead>
<tbody>
<tr>
<td>37</td>
<td>11</td>
<td>11</td>
<td>24</td>
<td>32</td>
</tr>
</tbody>
</table>

For some large unsafe problems syntactical transformation fails to deliver a counterexample within the timeout!
- Generates exponential amount of rules
- \texttt{RelSpacer} does the job in seconds
Experiments: Relational Benchmarks

“Synchronizing Constrained Horn Clauses” (Mordvinov et al., 2017)

- 37 relational problems
- A lot of problems with undefinable «classical» invariants
- New competitor: CHC-product implemented as a syntactical preprocessing of clauses in Z3, then solving with Spacer
- Timeout: 5 min.

<table>
<thead>
<tr>
<th>Total</th>
<th>HoIce</th>
<th>Spacer</th>
<th>CHC-Product</th>
<th>REL-Spacer</th>
</tr>
</thead>
<tbody>
<tr>
<td>37</td>
<td>11</td>
<td>11</td>
<td>24</td>
<td>32</td>
</tr>
</tbody>
</table>

For some large unsafe problems syntactical transformation fails to deliver a counterexample within the timeout!

- Generates exponential amount of rules
- REL-Spacer does the job in seconds
Conclusion

• Traditional safety proofs are not expressive enough for nonlinear CHC systems
• Relational safety proofs is a better form of their solutions
• Traditional PDR works well for linear systems
• It can be easily generalized to infer relational proofs!
  • Generate relational proof obligations instead of splitting safety property
  • Block CTI for product programs
• Relational PDR is a better choice than syntactical transformations!