Verifying relational program properties by transforming constrained Horn clauses

E. De Angelis¹, F. Fioravanti¹, A. Pettorossi², and M. Proietti³

¹ University of Chieti-Pescara `G. d'Annunzio'
² University of Rome `Tor Vergata'
³ CNR - Istituto di Analisi dei Sistemi ed Informatica, Rome
Relational verification

Relational properties
1. modified code still complies with its specification
2. unmodified code has not been affected by the changeset

Example: program equivalence
Verification methods
State-of-the-art

- Verification condition generator
- (semi) automated
- Verification conditions
- Satisfiability Modulo Theories (SMT) solver
Weakeness

specific for

verification condition generator

programming language

properties

...
Our goal & contribution

Achieve a higher level of parametricity with respect to programming language & properties

Verification method based on Transformation of Constrained Horn Clauses (CHCs)

- **CHCs** as a metalanguage for representing and as a set of implications of the form $A_0 \leftarrow c, A_1, \ldots, A_n$
- **Transformations** of CHCs to compose clauses representing the programs and the relational property
- **Transformations** increase the effectiveness of satisfiability provers
Relational Verification by CHCs transformation

CHC encoder

Interpreter for SRC (operational semantics)

(fully automated) CHC transformer

Transformed CHC

CHC solver (SAT provers)
Specifying Relational Properties using CHCs

The relational property \( \{\varphi\} \ P_1 \sim P_2 \ \{\psi\} \) is translated into the clause

\[
\text{false} \leftarrow \text{pre}(X, Y), p_1(X, X'), p_2(Y, Y'), \text{neg\_post}(X', Y')
\]

<table>
<thead>
<tr>
<th>pre-relation</th>
<th>( \varphi )</th>
<th>( \text{pre}(X, Y) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>input/output relation</td>
<td>( P_1 )</td>
<td>( p_1(X, X') )</td>
</tr>
<tr>
<td>input/output relation</td>
<td>( P_2 )</td>
<td>( p_2(Y, Y') )</td>
</tr>
<tr>
<td>negation of post-relation</td>
<td>( \neg\psi )</td>
<td>( \text{neg_post}(X', Y') )</td>
</tr>
</tbody>
</table>

check the **validity** of a relational property reduces to check the **satisfiability** of its CHCs representation
**Interpreter (a glimpse)**
Operational semantics of the programming language

\[
\text{prog}(X, X') \leftarrow \text{initConf}(C, X), \quad \text{reach}(C, C'), \quad \text{finalConf}(C', X')
\]

- **input/output relation**
- **initial C and final C' configurations**
  \[\text{cf}(\text{cmd}(\text{Label,Command}), \text{Environment})\]

\[
\begin{align*}
\text{reach}(C, C') & \leftarrow \text{tr}(C, C1), \text{reach}(C1, C2) \\
\text{tr}(\text{cf}(\text{cmd}(L, \text{asgn}(X, \text{expr}(E))), \text{Env}), \text{cf}(\text{cmd}(L1, C), \text{Env1})) & \leftarrow \\
\text{eval}(E, \text{Env}, V), \text{update}(\text{Env}, X, V, \text{Env1}), \text{nextlab}(L, L1), \text{at}(L1, C)
\end{align*}
\]

- \[x = e;\]
Transformation of CHCs

CHS solvers are often unable to prove satisfiability

A technique that
- manipulates clauses
- preserves their satisfiability

Transformation strategies:
1. CHC Specialization
2. Predicate Pairing

\[ S_1 \text{ is satisfiable } \iff S_2 \text{ is satisfiable} \]
Interpreters & CHC Specialization

Take advantage of static information, that is,
- actual programs
- relational property

to customize the interpreter

By specializing the interpreter w.r.t. the static input, we get CHCs with no references to
- reach
- tr
- complex terms representing configurations
Example

\[ P_1: \text{void sum_upto()} \{ \]
\[ \quad z_1 = f(x_1); \]
\[ \}
\[ \quad \text{int } f(\text{int } n1)\{ \]
\[ \quad \quad \text{int } r1; \]
\[ \quad \quad \text{if } (n1 \leq 0) \{ \]
\[ \quad \quad \quad r1 = 0; \]
\[ \quad \quad \} \text{ else } \{ \]
\[ \quad \quad \quad r1 = f(n1 - 1) + n1; \]
\[ \quad \}
\[ \quad \text{return } r1; \]
\[ \}
\]
\[ \text{non-tail recursive} \]

\[ P_2: \text{void prod()} \{ \]
\[ \quad z_2 = g(x_2, y_2); \]
\[ \}
\[ \quad \text{int } g(\text{int } n2, \text{int } m2)\{ \]
\[ \quad \quad \text{int } r2; \]
\[ \quad \quad r2 = 0; \]
\[ \quad \quad \text{while } (n2 > 0) \{ \]
\[ \quad \quad \quad r2 += m2; \]
\[ \quad \quad \quad n2--; \]
\[ \quad \}
\[ \quad \text{return } r2; \]
\[ \}
\]
\[ \text{iterative} \]

global variables \( \mathcal{V}_1 = \{ x_1, z_1 \} \) of \( P_1 \)

global variables \( \mathcal{V}_2 = \{ x_2, y_2, z_2 \} \) of \( P_2 \)

\[ z_1 = \sum_{i=0}^{x_1} i \]

\[ \mathcal{V}_1 \cap \mathcal{V}_2 = \emptyset \]

\[ z_2 = x_2 \times y_2 \]

\[ Leq: \{ x_1 = x_2, \ x_2 \leq y_2 \} \ \text{sum_upto} \sim \text{prod} \ \{ z_1 \leq z_2 \} \]
CHCs specialization

\( P_1: \) void sum_upto() {
    z1=f(x1);
}

int f(int n1){
    int r1;
    if (n1 <= 0) {
        r1 = 0;
    } else {
        r1 = f(n1 - 1) + n1;
    }
    return r1;
}

CHCs

\( su(X1, Z1') \leftarrow f(X1, Z, X1, R, N1, Z1') \)
\( f(X, Z, N, R, N, 0) \leftarrow N \leq 0 \)
\( f(X, Z, N, R, N, Z1) \leftarrow N \geq 1, N1 = N - 1, Z1 = R2 + N, f(X, Z, N1, R1, N2, R2) \)
CHCs specialization

\( P_2: \) void prod() {
    z2 = g(x2, y2);
}

int g(int n2, int m2) {
    int r2;
    r2 = 0;
    while (n2 >= 1) {
        r2 += m2;
        n2--;
    }
    return r2;
}

CHCs

\[
\begin{align*}
    p(X_2, Y_2, Z_2') & \leftarrow g(X_2, Y_2, Z, X_2, Y_2, 0, N, P, Z_2') \\
g(X, Y, Z, N, P, R, N, P, R) & \leftarrow N \leq 0 \\
g(X, Y, Z, N, P, R, N_2, P_2, R_2) & \leftarrow N \geq 1, N_1 = N - 1, R_1 = P + R,
\end{align*}
\]
Satisfiability of CHCs

\[
\{ \varphi \} \quad P_1 \sim P_2 \quad \{ \psi \}
\]

\[
\text{false} \leftarrow X_1 = X_2, \text{ } X_2 \leq Y_2, Z_1' > Z_2', \quad \text{su}(X_1, Z_1'), \quad \text{p}(X_2, Y_2, Z_2') \quad \text{(*)}
\]

\[
P_1
\]

\[
f(X, Z, N, R, N, 0) \leftarrow N \leq 0
\]

\[
f(X, Z, N, R, N, Z_1) \leftarrow N \geq 1, N_1 = N - 1, Z_1 = R_2 + N, \quad f(X, Z, N_1, R_1, N_2, R_2)
\]

\[
g(X_2, Y_2, Z_2') \leftarrow g(X_2, Y_2, Z, X_2, Y_2, 0, N, P, Z_2')
\]

\[
P_2
\]

\[
g(X, Y, Z, N, P, R, N, P, R) \leftarrow N \leq 0
\]

\[
g(X, Y, Z, N, P, R, N_2, P_2, R_2) \leftarrow N \geq 1, N_1 = N - 1, R_1 = P + R,
\]

\[
g(X, Y, Z, N_1, P, R_1, N_2, P_2, R_2)
\]

- state-of-the-art solvers for CHCs with Linear Integer Arithmetic (LIA) are unable to prove their satisfiability to prove their satisfiability, that is, the premise of clause (\text{(*)}) unsatisfiable, LIA solvers should discover quadratic relations

\[
Z_1' = X_1 \times (X_1 - 1)/2 \quad Z_2' = X_2 \times Y_2
\]

- reasoning on \text{su} and \text{p} separately is unhelpful
Predicate Pairing

• **Solution 1:** use a solver for non-linear integer arithmetic
drawback: satisfiability of constraints is undecidable

• **Solution 2:** use the transformation, again!

**Predicate Pairing transformation strategy**

• **composes** the predicates $f$ and $g$ into a new predicate $fg$
equivalent to their conjunction

• Objective: to **discover** relations among variables occurring
  in $f$ and $g$ to help the solvers in proving the satisfiability
  of the CHCs
Predicate Paring in action

false $\leftarrow X_1 = X_2, X_2 \leq Y_2, Z_1' > Z_2', su(X_1, Z_1'), p(X_2, Y_2, Z_2')$

1. Unfold
   
   $false \leftarrow X_1 \leq Y_2, Z_1' > Z_2', f(X_1, Z, X_1, R, N_1, Z_1'), g(X_1, Y_2, Z, X_1, Y_2, 0, N_2, P_2, Z_2')$

2. Define
   
   $fg(X_1, Z, N, R, N_1, Z_1', Y_2, V, W, N_2, P_2, Z_2') \leftarrow f(X_1, Z, N, R, N_1, Z_1'), g(X_1, Y_2, V, N, Y_2, W, N_2, P_2, Z_2')$

3. Fold
   
   $false \leftarrow X_1 \leq Y_2, Z_1' > Z_2', fg(X_1, Z, X_1, R, N_1, Z_1', Y_2, Z, 0, N_2, P_2, Z_2')$
Satisfiability of CHCs

Transformed CHCs

\[ \text{false} \leftarrow X_1 \leq Y_2, \ Z_1' > Z_2', \ fg(X_1, Z, X_1, R, N_1, Z_1', Y_2, Z, 0, N_2, P_2, Z_2') \]
\[ fg(X, Z, N, R, N_0, Y_2, V, Z_2, N, P_2, Z_2) \leftarrow N \leq 0 \]
\[ fg(X, Z, N, R, N, Z_1, Y_2, V, W, N_2, P_2, Z_2) \leftarrow \]
\[ N \geq 1, \ N_1 = N - 1, \ Z_1 = R_2 + N, \ M = Y_2 + W, \]
\[ fg(X, Z, N_1, R_1, S, R_2, Y_2, V, M, N_2, P_2, Z_2) \]

Predicate Pairing makes it possible to infer linear relations among variables in the conjunction \( fg \) of predicates \( f \) and \( g \)

\[ fg(X_1, Z, N, R, N_1, Z_1', Y_2, V, W, N_2, P_2, Z_2') \leftarrow \]
\[ f(X_1, Z, N, R, N_1, Z_1'), g(X_1, Y_2, V, N, Y_2, W, N_2, P_2, Z_2') \]

The conjunction \( fg \) enforces the linear constraint

\[ (X_1 > Y_2) \lor (Z_1' \leq Z_2') \]

Hence the satisfiability of the first clause
Implementation & Experimental Evaluation

\[
\{\varphi\} \ P_1 \sim P_2 \ \{\psi\}
\]

**VErIMAP**
1. CHC Encoding;
2. Predicate Pairing;
3. Constraint Propagation;

CHC Interpreter

**CHC Solver**
- ELDARICA or Z3
- or MathSAT

<table>
<thead>
<tr>
<th>Properties</th>
<th>n</th>
<th>Enc+Eld</th>
<th>Enc+Z3</th>
<th>PP+Eld</th>
<th>PP+MS</th>
<th>PP+Z3</th>
<th>CP+Eld</th>
<th>CP+MS</th>
<th>CP+Z3</th>
</tr>
</thead>
<tbody>
<tr>
<td>ITERATIVE</td>
<td>21</td>
<td>7</td>
<td>6</td>
<td>13</td>
<td>19</td>
<td>6</td>
<td>17</td>
<td>20</td>
<td>21</td>
</tr>
<tr>
<td>RECURSIVE</td>
<td>18</td>
<td>7</td>
<td>8</td>
<td>7</td>
<td>11</td>
<td>6</td>
<td>14</td>
<td>11</td>
<td>13</td>
</tr>
<tr>
<td>ITERATIVE-RECURSIVE</td>
<td>4</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td>0</td>
<td>4</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>ARRAYS</td>
<td>5</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>4</td>
<td>2</td>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>LEQ</td>
<td>6</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>6</td>
<td>1</td>
<td>3</td>
<td>6</td>
<td>4</td>
</tr>
<tr>
<td>MONOTONICITY</td>
<td>18</td>
<td>4</td>
<td>4</td>
<td>11</td>
<td>16</td>
<td>8</td>
<td>11</td>
<td>17</td>
<td>14</td>
</tr>
<tr>
<td>INJECTIVITY</td>
<td>11</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>11</td>
<td>4</td>
<td>7</td>
<td>11</td>
<td>10</td>
</tr>
<tr>
<td>FUNCTIONALITY</td>
<td>7</td>
<td>5</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>7</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td>90</td>
<td>24</td>
<td>25</td>
<td>39</td>
<td>74</td>
<td>34</td>
<td>64</td>
<td>78</td>
<td>78</td>
</tr>
</tbody>
</table>
Conclusions

• A method for proving relational properties
  • translating the property in CHCs
  • transforming the CHCs to better exploit the interaction between predicates

• Independent of the programming language
  • The only language specific element is the interpreter
  • Can be applied to prove relations between programs that may be written in different programming languages

• Improves effectiveness of state-of-the art CHC solvers