Progetto GR 2016-17

Verifica formale di modelli e programmi basata sulla trasformazione di clausole di Horn con vincoli

Emanuele De Angelis
Università degli Studi `G. d'Annunzio' Chieti-Pescara

joint work with
F. Fioravanti (UdA), M. C. Meo (UdA),
A. Pettorossi (U. Tor Vergata), and M. Proietti (IASI-CNR)

Convegno GNCS – Montecatini Terme, 14–16 febbraio 2018
Automatic Verification of (software) artifacts

Providing a proof that an artifact (for instance, a model or a program) satisfies its specification.

Use a mathematical formalism to:

- model artifacts, and
- derive specifications as theorems.
Constrained Horn Clauses (CHCs)

First order formulas of the form

\[ B_1 \land \ldots \land B_n \land c \rightarrow H \]

where:
- \( B_1, \ldots, B_n \), and \( H \) are atomic formulas, and
- \( c \) is a formula in a theory of constraints.

Formulas are universally quantified in front.

We use the syntax of Logic Programming

\[ H \leftarrow c, B_1, \ldots, B_n \quad (\text{Head } \leftarrow \text{Body}) \]
Relational Verification
Proving relations between programs

Program Equivalence
If $P_1$ terminates on the input $i_1$ producing $o_1$ & $P_2$ terminates on the input $i_2$ producing $o_2$ & $i_1$ equals to $i_2$
then $o_1$ equals to $o_2$
Verification of Relational Properties using transformation of CHCs

Programs

$P_1$  
$P_2$

Specification

$\varphi$

CHC encoder

$\forall$

CHCs

Transformation strategy

1. unfold
2. define
3. fold

Rule-based CHC transformer

Transformed CHCs

CHC solver

$\varphi$

Interpreter

Semantics of C & Specification Logic

$\vdash$

$\forall$
Example

**P1**

```c
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;
}
```

*non-tail recursive*

global variables of **P1**: \{x1, z1\}

\[ z1 = \sum_{i=0}^{x1} i \]

**P2**

```c
void prod() {
    z2 = g(x2, y2);
}
int g(int n2, int m2){
    int r2;
    r2=0;
    while (n2 > 0) {
        r2 += m2;
        n2--;
    }
    return r2;
}
```

*iterative*

global variables of **P2**: \{x2, y2, z2\}

\[ z2 = x2 \times y2 \]

**Relational property**  \\[
\{x1 = x2, \; x2 \leq y2\} \; \text{sum\_upto} \sim \text{prod} \; \{z1 \leq z2\}\]
Specifying relational properties using CHCs

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

\[
post(X', Y') \leftarrow pre(X, Y), p_1(X, X'), p_2(Y, Y')
\]

<table>
<thead>
<tr>
<th>pre-relation</th>
<th>( \varphi )</th>
<th>( 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>post-relation</td>
<td>( \psi )</td>
<td>( post(X', Y') )</td>
</tr>
</tbody>
</table>

Relational property: \( \{ x_1 = x_2, \ x_2 \leq y_2 \} \ \text{sum}_\text{upto} \sim \\text{prod} \ \{ z_1 \leq z_2 \} \)

CHC translation:

\[
Z_1 \leq Z_2 \leftarrow X_1 = X_2, \ X_2 \leq Y_2, \ \text{su}(X_1, Z_1), \ \text{pr}(X_2, Y_2, Z_2)
\]
Interpreter (a glimpse)
Operational semantics of the programming language

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

Input/output relation

Initial \( C \) and final \( C' \) configurations
\( \text{cf}(\text{cmd}(\text{Label}, \text{Command}), \text{Environment}) \)

\[ \text{reach}(C, C') \]
\[ \text{reach}(C, C2) \leftarrow \text{tr}(C, C1), \text{reach}(C1, C2) \]

\[ \text{x=e;} \]
\[ \text{tr}(\text{cf(cmd(L, asgn( V, expr(E))), Env), cf(cmd(L1, S), Env1)) \leftarrow eval(E, Env, V), update(Env, X, V, Env1), nextlab(L, L1), at(L1, S)} \]
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
- \textit{reach}
- \textit{tr}
- complex terms representing \textit{configurations}
```c
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;
}
```

**Input/Output relation of P1**

\[
\begin{align*}
\text{su}(X,Z') & \leftarrow f(X,Z') \\
\text{f}(X,Z) & \leftarrow N\leq 0, Z=0 \\
\text{f}(N,Z) & \leftarrow N\geq 1, N1=N-1, Z=R+N, f(N1,R)
\end{align*}
\]
prove the validity of a relational property reduces to
prove the satisfiability of CHCs
Satisfiability of CHCs

\[
\text{false} \leftarrow X_1 = X_2, \ X_2 \leq Y_2, \ Z_1 > Z_2, \ su(X_1, Z_1), \ pr(X_2, Y_2, Z_2), \ su(X, Z) \leftarrow f(X, Z), \ f(N, Z) \leftarrow N \leq 0, \ Z = 0, \ f(N, Z) \leftarrow N \geq 1, \ N_1 = N - 1, \ Z = R + N, \ f(N_1, R), \ pr(X, Y, Z) \leftarrow W = 0, \ g(X, Y, W, Z), \ g(N, P, R, R) \leftarrow N \leq 0, \ g(N, P, R, R_2) \leftarrow N \geq 1, \ N_1 = N - 1, \ R_1 = P + R, \ g(N_1, P, R_1, R_2)
\]

State-of-the-art solvers for CHCs with **Linear Integer Arithmetic (LIA)** look for **models of single atoms**: \( su \) and \( pr \).

Hence, LIA solvers should discover quadratic relations:

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

“Solution 1”: use a solver for non-linear integer arithmetic
drawback: satisfiability of constraints is \textit{undecidable}
(decide satisfiability of Diophantine equations)

Solution 2: \textit{predicate pairing} transformation

- composes the predicates \( f \) and \( g \) into a new predicate
  \( fg \) equivalent to their \textit{conjunction}
- objective: discover linear relations among variables
  occurring in \( f \) and \( g \) may help solvers in proving the
  satisfiability of CHCs
Satisfiability of CHCs

Transformed CHCs

\[ \text{false} \leftarrow N \leq Y, \ W = 0, \ Z1 > Z2, \ fg(N, Z1, Y, W, Z2) \]
\[ fg(N, Z1, Y, Z2, Z2) \leftarrow N \leq 0, \ Z1 = 0 \]
\[ fg(N, Z1, Y, W, Z2) \leftarrow N \geq 1, \ N1 = N - 1, \ Z1 = R + N, \ M = Y + W, \]
\[ fg(N1, R, Y, M, Z2) \]

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

\[ fg(N, Z1, Y, W, Z2) \leftarrow f(N, Z1), \ g(N, Y, W, Z2) \]

Whenever \( W = 0 \) the conjunction \( fg \) enforces the linear constraint

\[ (N > Y) \lor (Z1 \leq Z2) \]

Hence the satisfiability of the first clause
Implementation

CHCs
(encoding the verification problem)

VeriMAP

Transformation strategy

1. unfold
2. define
3. fold

Transformed CHCs

CHC solver

Z3

true
unknown
false

Interpreter

∀

∀

http://map.uniroma2.it/VeriMAP
Results

<table>
<thead>
<tr>
<th>Property</th>
<th>Constraints</th>
</tr>
</thead>
<tbody>
<tr>
<td>Equivalence</td>
<td>( p_1(X, X'), p_2(Y, Y'), X = Y \rightarrow X' = Y' )</td>
</tr>
<tr>
<td>Monotonicity</td>
<td>( p(X, X'), p(Y, Y'), X \leq Y \rightarrow X' \leq Y' )</td>
</tr>
<tr>
<td>Injectivity</td>
<td>( p(X, X'), p(Y, Y'), X' = Y' \rightarrow X = Y )</td>
</tr>
<tr>
<td>Functionality</td>
<td>( p(X, f(X), X'), p(Y, f(Y), Y'), X = Y \rightarrow X' = Y' )</td>
</tr>
</tbody>
</table>

Relational properties

- **Encoding**: Blue
- **Predicate Pairing**: Orange
- **Predicate Pairing + Constraint Propagation**: Yellow
Verification of Models using transformation of CHCs

BP Model

M

BPMN

Specification

φ

CHC encoder

∀

CHCs

Transformation strategy

1. unfold
2. define
3. fold

Rule-based CHC transformer

Transformed CHCs

CHC solver

Interpreter

Semantics BPMN & Specification Logic

∀

φ

φ

φ

φ
A Business Process (BP) **coordinates** the activities of an organization towards a business goal. A BP can be represented using the **Business Process Modeling Notation**.

### Purchase Order
A customer adds one or more items to the shopping cart and pays. Then, the vendor sends the invoice and delivers the order.

No quantitative time information (such as the durations of tasks).
Time-aware Business Process

Specify intervals of task duration $D \in [d_{min}, d_{max}] \subset \mathbb{N}$

Properties

- **Reachability**
  The time to reach ‘end’ from ‘start’ is less than $T$.

- **Controllability**
  It is possible to determine the durations of some (controllable) tasks so that a given reachability property holds.
**Interpreter (a glimpse)**

Semantics of Business Process Modeling Notation

**Reachability**

\[
reach(S, S', U, C).
\]

\[
reach(S, S_2, U, C) \leftarrow tr(S, S_1, U, C), \ reach(S_1, S_2, U, C)
\]

where \( tr \) encodes the semantics of BPMN (Interpreter \( I \)).

**Reachability Property**

\[
reachProp(U, C') \leftarrow c(T, U, C), \ reach(init, \ fin(T), U, C')
\]

where \( c(T, U, C) \) is a constraint.

**Controllability Property**

**Weak**

\[
I \cup LIA \models \forall U. \ adm(U) \rightarrow \exists C \ reachProp(U, C')
\]

**Strong**

\[
I \cup LIA \models \exists C \forall U. \ adm(U) \rightarrow reachProp(U, C')
\]

where \( adm(U) \) iff the durations in \( U \) belong to the given intervals.
Applying CHCs solvers

Validity of weak and strong controllability properties

- **cannot be proved** by CHC solvers over LIA (such as Z3), because of complex terms occurring in the interpreter
- **cannot be proved** by CLP systems, because of $\exists \forall$ and $\forall \exists$
- CHC solvers and CLP system **may not terminate**, because of recursive definition of reach

Transformation techniques can be applied

- to get CHCs with no complex terms, and
- to avoid expensive quantifier elimination by reducing the problem of verifying controllability to the problem of verifying simpler properties where quantification is restricted to LIA constraints
Conclusions

A method for **proving** correctness of (software) artifacts

- **Independent of the formalism** used to represent the **artifact** and its **specification**

  The only language specific element is the **interpreter**

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

Future work:

- more formalisms (programming and modeling languages)
- more properties
Publications


- Emanuele De Angelis, Fabio Fioravanti, Maria Chiara Meo, Alberto Pettorossi, Maurizio Proietti: *Verification of Time-Aware Business Processes using Constrained Horn Clauses*. LOPSTR 2016: 38-55


- Emanuele De Angelis, Fabio Fioravanti, Alberto Pettorossi, Maurizio Proietti: *Enhancing Predicate Pairing with Abstraction for Relational Verification*. LOPSTR 2017 (to appear)