Solving non-linear Horn clauses using a linear Horn clause solver

Bishoksan Kafle, John Gallagher and Pierre Ganty

Roskilde University, Denmark and IMDEA Software Institute, Spain

HCVS’16 Eindhoven
03/04/2016
Number of solvers based on Constrained Horn Clauses (CHCs) are available:

After fixing a constraint theory, the (Horn clause) solvers are:
- linear e.g., VeriMap, Sally etc.
- non-linear e.g., RAHFT, SeaHorn, QARMC, ELDARICA, Z3 etc.

since the underlying engine of linear solver can handle only linear clauses which restricts, in principle, their applicability

Can we solve non-linear CHCs using a linear Horn clause solver?

**Notation:** solver = Horn clause solver, linear solver = Horn clause solver for linear Horn clauses
Yes, by interleaving program transformation (Horn clause linearisation) with linear Horn solving in an incremental manner to handle non-linear clauses.
Example: CHCs defining the Fibonacci function (*Fib*)

\[
\begin{align*}
c1. & \quad \text{fib}(A, B) :- A \geq 0, \ A \leq 1, \ B = A. \\
c2. & \quad \text{fib}(A, B) :- A > 1, \ A2 = A - 2, \ \text{fib}(A2, B2), \\
& \quad A1 = A - 1, \ \text{fib}(A1, B1), \ B = B1 + B2. \\
c3. & \quad \text{false} :- A > 5, \ \text{fib}(A,B), \ B < A.
\end{align*}
\]

c1 and c3 are linear clauses, c2 non-linear

The Horn clause verification to show that there is no successful derivation of *false*.
Program transformation (1)
is based on the idea of **tree dimension** of Horn clause derivations

The dimension of tree is a measure of its (non)-linearity
Given a set of clauses (program) \( P \), the notion of *tree dimension* allows us to derive a program \( P^{\leq k} \) (\( P \) at most \( k \) or simply \( k \)-dim program) whose derivations trees have dimension \( \leq k \) (\( k \geq 0 \)).

The Horn clause verification problem based on tree dimension:

- show that there is no successful derivation of \( \text{false} \) – of any dimension.

- It is known that \( P^{\leq k} \) is linearisable [Afrati et al., 2003].

this allows us to generate programs for increasing value of \( k \), linearise and solve them.
dimension of Fib’s derivation trees depends on the input number.

c1. \( \text{fib}(A, B) : - \ A \geq 0, \ A = < 1, \ B = A. \)
c2. \( \text{fib}(A, B) : - \ A > 1, \ A_2 = A - 2, \ \text{fib}(A_2, B_2), \)
\[ A_1 = A - 1, \ \text{fib}(A_1, B_1), \ B = B_1 + B_2. \]
c3. \( \text{false} : - \ A > 5, \ \text{fib}(A, B), \ B < A. \)

\( \text{Fib}^{\leq 0} \) (linear)

\( \text{fib}(0)(A, B) : - \ A \geq 0, \ A = < 1, \ B = A. \)
\( \text{false}(0) : - \ A > 5, \ B < A, \ \text{fib}(0)(A, B). \)
\( \text{false}[0] : - \ \text{false}(0). \)
\( \text{fib}[0](A, B) : - \ \text{fib}(0)(A, B). \)

the atom \( p(k)(X) \) means any derivation tree rooted at \( p(0)(X) \) will have

tree dimension \( k \)
\( p[k](X) - \text{tree dimension} \leq k \)
fib(0)(A,B) :- B=A, A=<1, A>=0.
false(0) :- B<A, A>5, fib(0)(A,B).
false[0] :- false(0).
fib[0](A,B) :- fib(0)(A,B).

fib(1)(A,B) :- B=F+D, C=A-2, E=A-1, A>1, fib[0](E,F), fib(1)(C,D).
fib(1)(A,B) :- B=F+D, C=A-2, E=A-1, A>1, fib[0](C,D), fib(1)(E,F).
fib(1)(A,B) :- B=F+D, C=A-2, E=A-1, A>1, fib(0)(C,D), fib(0)(E,F).

All clauses of $Fib^{\leq 0}$ are in $Fib^{\leq 1}$
by construction all clauses of $P^{\leq k}$ are included in $P^{\leq k+1}$ ($k \geq 0$)

As a result

- it provides a basis for iterative strategy for dimension bounded programs.
- reuse the solution obtained for lower dimension to linearise/solve clauses of higher dimension
based on partial evaluation (PE). PE is a source-source program transformation.

- P: non-linear clauses, I: an interpreter (linear in our case, written in some language L (as Horn clauses))
- I’ is a specialised interpreter for P, which can be regarded as the transformation of P.

- same as predicate tuppling (Invited talk).
Assume that the following is the solution for $Fib_{\leq 0}$

\[
\begin{align*}
\text{fib}(0)(A,B) & : \quad B=A, \quad A=\langle 1, A \rangle = 0. \\
\text{false}(0) & : \quad \text{FALSE}.
\end{align*}
\]

Given $Fib_{\leq 1}$

\[
\begin{align*}
\text{fib}(0)(A,B) & : \quad B=A, \quad A=\langle 1, A \rangle = 0. \\
\text{false}(0) & : \quad B<A, \quad A>5, \text{fib}(0)(A,B). \\
\text{false}[0] & : \quad \text{false}(0). \\
\text{fib}[0](A,B) & : \quad \text{fib}(0)(A,B).
\end{align*}
\]

\[
\begin{align*}
\text{fib}(1)(A,B) & : \quad B=F+D, \quad C=A-2, \\
& \quad E=A-1, \quad A>1, \text{fib}[0](E,F), \text{fib}(1)(C,D).
\end{align*}
\]

\[
\begin{align*}
\text{fib}(1)(A,B) & : \quad B=F+D, \quad C=A-2, \quad A>1, \text{fib}[0](C,D), \text{fib}(1)(E,F).
\end{align*}
\]

\[
\begin{align*}
\text{fib}(1)(A,B) & : \quad B=F+D, \quad C=A-2, \quad E=A-1, \\
& \quad A>1, \text{fib}(0)(C,D), \text{fib}(0)(E,F).
\end{align*}
\]
Fib$^\leq_1$ after solution reuse

fib(0)(A,B) :- B=A, A=<1, A>=0.
fib[0](A,B) :- B=A, A=<1, A>=0.

fib(1)(A,B) :- B=F+D, C=A-2, E=A-1, A>1, fib[0](E,F), fib(1)(C,D).
fib(1)(A,B) :- B=F+D, C=A-2, E=A-1, A>1, fib[0](C,D), fib(1)(E,F).
fib(1)(A,B) :- B=F+D, C=A-2, E=A-1, A>1, fib(0)(C,D), fib(0)(E,F).

.
.
.
Then we can linearise this program.
Assumption about a linear solver

1. linear solver is a black box and is sound
2. capable of producing a tuple $\text{Status} \times \text{Result}$ where $\text{Status} \in \{\text{safe or unsafe}\}$ and $\text{Result} \in \{\text{solution, counterexample}\}$

A solution for $P$ is a set of constrained facts of the form: $p(X) \leftarrow \phi$ for each predicate $p$ occurring in $P$
Our approach

**Lin – Linearisation procedure**

**LS – Linear Horn clause solver**

**CC – Counterexample Analyser**

**Abstraction**

CHC $P$

$k = 0, S = \emptyset$

$P', S, k$

R solution $P$?

No

$S \leftarrow R, k = k + 1$

**Refinement**

(safe, $R'^P$)

$S, R, k$

(unsafe, $R$)

$S \cap S|_R = \emptyset$?

$S \leftarrow S \setminus S|_R, k$

**Figure**: Abstraction-refinement scheme for solving non-linear Horn clauses using a linear solver. $P'$ is a linearised version of $P$'s k-dim program. $S|_R$ is a set of constrained facts from $S$ appearing in a counterexample.
Example: counterexample using approximate solution

\begin{verbatim}
c1. false:- X=0, p(X).
c2. false:- q(X).
c3. p(X):- X>0.
c4. q(X):-X=0.
\end{verbatim}

Suppose $S = \{ p(X) : \neg TRUE \}$ for the predicate $p(X)$. Using this solution, we get

\begin{verbatim}
c1. false:- X=0, p(X).
c2. false:- q(X).
c3. p(X):- TRUE. (approximate solution)
c4. q(X):-X=0.
\end{verbatim}

$c_1(c_3)$ is a spurious counterexample for the original program

$c_2(c_4)$ is a real counterexample
Experimental settings

1. **Linear solver:** Convex polyhedral analyser (CPA) – terminates but may generate *false alarms*
2. **Partial Evaluator:** Logen [Leuschel et al., 2003]
3. **Benchmarks:** 44 problems (SV-COMP’15, QARMC, Repository of Horn clauses)
4. **Tool:** LHornSolver
   (https://github.com/bishoksan/LHornSolver)

**Goal**

1. Whether solving non-linear Horn clauses can be done using a solver for linear Horn clauses?
2. The relation between the solvability of a program with tree dimension
3. Comparison with tools for non-linear Horn clauses
61% of the problems are solved
we found that the solution of $P^{\leq k}$ (for $k = 1, 2$) becomes a solution of $P$ or counterexample was found

The results on this set of benchmarks show that
it is feasible to solve non-linear Horn clauses using a linear solver and
the solvability of a problem is shallow wrt. tree dimension of its derivations.
Comparison with RAHFT (whose underlying engine is also CPA), Horn solver for non-linear clauses

- **RAHFT solves all the problems** unlike LHornSolver

This could mean

- **linearisation strategy** we use is not useful for solving non-linear Horn clauses
- the use of CPA in LHornSolver: no refinement is done when CPA produces a *false alarm*. In this case LHornSolver returns **unknown**
What next?

- experiment with **different linearisation strategies** for Horn clause
- **use different linear solver within LHornSolver**, which if returns returns with a solution or a counterexample wrt. the original program
Thanks for your attention!