Analysis and transformation tools for constrained Horn clause verification

John Gallagher **Bishoksan Kafle**

Roskilde University

ICLP 2014, Vienna

Constrained Horn clause verification

Constrained Horn clause (CHC)

A predicate logic formula, $H(X) \leftarrow \phi \wedge B_1(X_1), \ldots, B_k(X_k)$

- ϕ - a conjunction of constraints with respect to some background theory,

Integrity constraints

false $\leftarrow \phi \wedge B_1(X_1), \ldots, B_k(X_k)$.
where false is always interpreted as *false*.

CHC verification problem

- given a set of CHCs P (including integrity constraints encoding safety properties),
- does P have a model?

An Example

```
false :- A>0, B=0, C=0, D=0, l(B,C,D,A).

l(A,B,C,D) :- -A+D>0, A-G= -1, l_body(B,C,E,F), l(G,E,F,D).
l(A,B,C,D) :- A-D>=0, B+C-3*D>0.
l(A,B,C,D) :- A-D>=0, -B-C+3*D>0.

l_body(A,B,C,D) :- A-C= -1, B-D= -2.
l_body(A,B,C,D) :- A-C= -2, B-D= -1.
```

CHCs verification techniques and tools

- CHC has gained interest from CLP and software verification communities

CLP

- approximation of the minimal model of a CLP program using abstract interpretation (AI)
- specialisation wrt a goal
- model preserving transformations etc.

Verification

- Abstract interpretation
- counter example guided abstraction refinement (CEGAR) etc.

- Tools: VeriMAP, HSF(C) , TRACER etc.

CHCs is the software verification community's terminology for CLP
From now on set of CHCs, CLP are used interchangeably

Characteristics

	Commonalities	Characteristics	Issues
Abstract interpretation	derive invariants	scalable, not property guided	domain choice, false alarms
Specialisation	by transformation	model preserving, property guided	generalisation operators
CEGAR	by cEx analysis	property guided	cEx generalisation

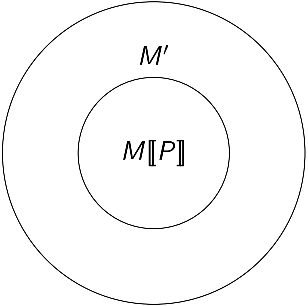
In essence, CHC verification boils down to deriving required program invariants but each of these techniques usually miss the aspect of each others

The idea of this paper is to investigate:

- whether the off-the-shelf CLP tools are suitable for this purpose
- whether we can fill the gap betweeen CLP and verification techniques

Proof by over-approximation of the minimal model

- There exists a minimal model, $M[\![P]\!]$, wrt the subset ordering,
- $M[\![P]\!]$ is equivalent to the set of atomic consequences of P (model vs. proof)
- It is sufficient to find a set of constrained facts M' such that $M[\![P]\!] \subseteq M'$, where false $\notin M'$.



Proof by specialisation / Transformation

Given P_0 and an atom A, we wish to prove A is not a consequence of P_0

$\boxed{P_0}$ $\boxed{P_1}$ ...... $\boxed{P_k}$

P_k contains no clause with head A

we wish to prove A is a consequence of P_0

$\boxed{P_0}$ $\boxed{P_1}$ ...... $\boxed{P_k}$

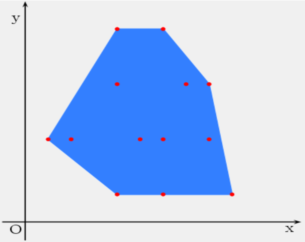
P_k contains a clause with head $A \leftarrow \mathit{true}$

- $P \models A$ if and only if $P' \models A$, P' is a specialisation of P.

Analysis

Convex polyhedra approximation (CPA)

- when applied to P it constructs an over-approximation M' of the minimal model of P, where M' contains at most one constrained fact $p(X) \leftarrow \mathcal{C}$ for each predicate p.
- where the constraint $\mathcal{C}$ is a conjunction of linear inequalities, representing a convex polyhedron.



$$\begin{cases} 6x + y \leq 111 \\ 3x + 2y \leq 78 \\ x + y \geq 11 \\ 2x - y \geq -5 \\ y \geq 3 \\ y \leq 21 \end{cases}$$

source: http://bugseng.com/products/ppl/abstractions

Our approach

We carry out following steps in an iterative manner:

1. apply CLP transformation and specialisation ,
2. analyse the resulting program by AI (over convex polyhedra),
3. refine (drawing idea from CEGAR)

Transformation tools and their roles (I)

1. Redundant argument filtering (Leuschel and Sørensen 1996)
   - specialise a program by removing redundant variables (equivalent to live variable analysis)
   - needed for scalability
2. Unfolding (Pettorossi and Proietti 1999)
   - can improve the structure of a program, by removing some case of mutual recursion, or propagating constraints upwards towards the integrity constraints
3. Specialisation / partial evaluation (Gallagher 1993; Leuschel 1999)
   - can remove parts of theories not relevant to the verification problem

Transformation tools and their roles

1. Predicate splitting (Pettorossi and Proietti 1999)
    - splits a predicate wrt to criteria
    - its role is to improve the precision loss by the analyser

$P \models A$ if and only if $P' \models A$, P' is a transformation of P.

Convex Polyhedral Analysis of P'

- produces an overapproximation
- presence of constrained fact for $false \rightarrow P$ may not be safe
- generate counterexamples

Refinement

- analysis of counterexamples
  - satisfiable $\rightarrow$ bug
  - unsat $\rightarrow$: the reason for refinement
- refinement of the program
  - the clauses appearing in counterexamples say which predicates to split
  - refine program by specialisation (Gallagher (1993))

Summary of our approach

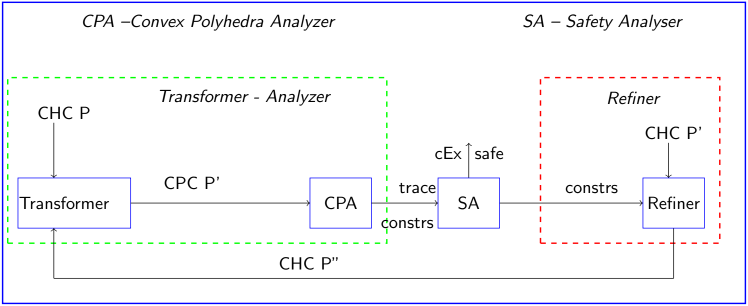


Figure : *Tool chain overview (CHC verification).*

Transformation sequence: Redudant argument filtering, unfolding, specialisation, predicate splitting.

Experiments

benchmarks

- repository of SV benchmarks [a] and
- other sources including Gupta et al. (2009) [Invgen], Beyer (2013) [SV-COMP 2013], Jaffar et al. (2012) [TRACER], De Angelis et al. (2014) [VeriMap] etc.

[a] https://svn.sosy-lab.org/software/sv-benchmarks/trunk/clauses/

environment

- Implementation: 32-bit Ciao Prolog [a] with Parma Polyhedra Library (Bagnara et al. (2008))
- Computer: Intel(R) X5355 having 4 processors (each @ 2.66GHz) and total memory of 6 GB. Debian 5 (64 bit) - OS,
- we set 2 minutes of timeout for each experiment.

[a] http://ciao-lang.org/

Experimental results

	Toolchain	Toolchain w-ref
solved (safe/unsafe)	162 (144/18)	180 (158/22)
unknown / timeout	54 (46/8)	36 (-/36)
average time (secs)	8.62	20.7

Table : Experimental results on 216 (179/37) CHC verification problems

- this shows the feasibility of our approach
- compares favourably with other tools in the literature (HSF(C), VeriMap, Tracer etc.)

Summary and Future Works

- a combination of off-the-shelf tools from CLP transformation and analysis is surprisingly effective in CHC verification
- component based approach give insights for further development of automatic CHC verification tools.
- understand better the connection between different techniques for verification

The End!

Thanks for your attention!

Query answer transformation (QA)

Simulates goal-directed computation in a goal-independent framework.

Given P and an atom A, the QA for P wrt. A, denoted P_A^{qa}, contains:

Answer clauses

For each clause $H \leftarrow C, B_1, \ldots, B_n$ $(n \geq 0)$ in P, P_A^{qa} contains the clause $H^{\mathsf{a}} \leftarrow C, H^{\mathsf{q}}, B_1^{\mathsf{a}}, \ldots, B_n^{\mathsf{a}}$.

Query clauses

For each clause $H \leftarrow C, B_1, \ldots, B_i, \ldots, B_n$ $(n \geq 0)$ in P, P_A^{qa} contains:

$B_1^{\mathsf{q}} \leftarrow C, H^{\mathsf{q}}$.

$\ldots$

$B_i^{\mathsf{q}} \leftarrow C, H^{\mathsf{q}}, B_1^{\mathsf{a}}, \ldots, B_{i-1}^{\mathsf{a}}$.

$\ldots$

$B_n^{\mathsf{q}} \leftarrow C, H^{\mathsf{q}}, B_1^{\mathsf{a}}, \ldots, B_{n-1}^{\mathsf{a}}$.

Goal clause

$A^{\mathsf{q}} \leftarrow \mathrm{true}$.

Specialisation by constraint propagation

The procedure is as follows: the inputs are a set of CHCs P and an atomic formula A.

1. Compute a P^{qa}_A, containing predicates p^{q} and p^{a} for each predicate p in P.
2. Compute an over-approximation of the model of P^{qa}_A, expressed as a set of constrained facts $p^*(X) \leftarrow C$, where $*$ is q or a. We assume that each predicate p^* has exactly one constrained fact in the model
3. For each clause $p(X) \leftarrow \mathcal{B}$ in P, let the model of p^{a} be $p^{\mathsf{a}}(X) \leftarrow C^{\mathsf{a}}$ (where X is the same tuple of variables in $p(X)$ and $p^{\mathsf{a}}(X)$).
4. Replace the clause $p(X) \leftarrow \mathcal{B}$ in P by $p(X) \leftarrow C^{\mathsf{a}}, \mathcal{B}$ in P_A.

Property

If P is a set of CHCs and P_{false} is the set obtained by strengthening the clause constraints as just described, then $P \models \mathsf{false}$ if and only if $P_{\mathsf{false}} \models \mathsf{false}$.

Sound and completeness (I)

Lemma

P has a model if and only if $P \not\models \text{false}$.

- holds for arbitrary interpretations (only assuming that the predicate false is interpreted as false)
- does not depend on the constraint theory

Soundness

- $P \vdash A$ implies $P \models A$
- means that $P \vdash \text{false}$ is a sufficient condition for P to have no model, by above Lemma
- corresponds to using a sound proof procedure to find or check a counterexample

Sound and completeness (II)

But soundness is not enough for P to have a model since we need to establish $P \not\models \text{false}$

Completeness

- we approach this problem by using *approximations* to reason about the non-provability of false
- applying the theory of abstract interpretation to a complete proof procedure for atomic formulas (the "fixed-point semantics" for constraint logic programs Jaffar et al. (1994)
- In effect, we construct by abstract interpretation a proof procedure that is *complete* (but possibly not sound) for proofs of atomic formulas
- $P \not\vdash \text{false}$ implies $P \not\models \text{false}$ and thus establishes that P has a model