CHC-COMP 2018

Arie Gurfinkel

Philipp Ruemmer, Grigory Fedyukovich, Adrien Champion

1st Competition on Solving Constrained Horn Clauses
CHC-COMP: CHC Solving Competition

First edition on July 13, 2018 at HVCS@FLOC

Constrained Horn Clauses (CHC) is a fragment of First Order Logic (FOL) that is sufficiently expressive to describe many verification, inference, and synthesis problems including inductive invariant inference, model checking of safety properties, inference of procedure summaries, regression verification, and sequential equivalence. The CHC competition (CHC-COMP) will compare state-of-the-art tools for CHC solving with respect to performance and effectiveness on a set of publicly available benchmarks. The winners among participating solvers are recognized by measuring the number of correctly solved benchmarks as well as the runtime.

Web: https://chc-comp.github.io/
Gitter: https://gitter.im/chc-comp/Lobby
GitHub: https://github.com/chc-comp
Constrained Horn Clauses (CHC)

A Constrained Horn Clause (CHC) is a FOL formula of the form

$$\forall V \cdot (\varphi \land p_1[X_1] \land \cdots \land p_n[X_n] \rightarrow h[X])$$

where

- $\mathcal{T}$ is a background theory (e.g., Linear Arithmetic, Arrays, Bit-Vectors, or combinations of the above)
- $\varphi$ is a constraint in the background theory $\mathcal{T}$
- $p_1, \ldots, p_n, h$ are $n$-ary predicates
- $p_i[X]$ is an application of a predicate to first-order terms
CHC Satisfiability

A $\mathcal{T}$-model of a set of a CHCs $\Pi$ is an extension of the model $M$ of $\mathcal{T}$ with a first-order interpretation of each predicate $p_i$ that makes all clauses in $\Pi$ true in $M$.

A set of clauses is **satisfiable** if and only if it has a model

- This is the usual FOL satisfiability

A $\mathcal{T}$-solution of a set of CHCs $\Pi$ is a substitution $\sigma$ from predicates $p_i$ to $\mathcal{T}$-formulas such that $\Pi\sigma$ is $\mathcal{T}$-valid

In the context of program verification

- a program satisfies a property iff corresponding CHCs are satisfiable
- solutions are inductive invariants
- refutation proofs are counterexample traces
benchmark ::= logic fun_decl+ (assert chc_assert)* (assert chc_query) (check-sat)

logic ::= (set-logic HORN) fun_decl ::= (declare-fun symbol ( sort* ) Bool)

chc_assert ::= ;; a well-formed first-order sentence of the form | (forall ( var_decl+ ) (=> chc_tail chc_head)) | chc_head
var_decl ::= (symbol sort)
chc_head ::=  
              | (u_predicate var*) , where all variables are
            DISTINCT
chc_tail ::=  
             | (u_predicate var*)
             | i_formula
             | (and (u_predicate var*)+ i_formula*)
chc_query ::= ;; a well-formed first-order sentence of the form
              | (forall ( var_decl+ ) (=> chc_tail false)

u_predicate ::= uninterpreted predicate (i.e., Boolean function)
i_formula ::= an SMT-LIB formula over variables, and
                 interpreted functions and predicates
Example
(set-logic HORN)
(declare-fun Inv (Int) Bool)

;; fact
(assert (forall ((x Int)) (=> (= x 0) (Inv x))))

;; chc
(assert (forall ((x Int) (y Int))
  (=> (and (Inv x) (<= x 10) (= y (+ x 1))) (Inv y))))

;; query
(assert (forall ((x Int))
  (=> (and (Inv x) (> x 15)) false))
(check-sat)
Benchmark Selection

chc-comp.github.io as collection for CHC benchmarks

- Collected from participants
- Generated with SeaHorn from SV-COMP Device Driver problems

Converted to competition format using format script (Adriene)

- https://github.com/chc-comp/scripts

Out of successfully formatted benchmarks selected

- 339 CHC-LINEAR(LIA)
- 132 CHC-LINEAR(LRA)

- Random selection, favoring hard-for-spacer benchmarks

Benchmarks released after competition:

Participants

Eldarica
Hoice
PECOS
TransfHORNER
Ultimate TreeAutomizer
Ultimate Unihorn Automizer

Philipp Rümmer
Adriene Champion
John Gallagher
Fabio Fioravanti
Alexander Nutz
Alexander Nutz

Hors Concours

spacer
rebus
Arie Gurfinkel
unnamed solver not entered in the competition
Hoice [Champion et al., 2018]

ICE-based Refinement Type Discovery for Higher-Order Functional Programs

Adrien Champion¹, Tomoya Chiba¹, Naoki Kobayashi¹, Ryosuke Sato²

¹ The University of Tokyo
² Kyushu University

https://github.com/hopv/hoice

• machine-learning-based Horn clause solver: generalized ICE framework [Garg et al., 2014]
• context: higher-order program verification
• supports Int, Real, and Array
• support for datatypes coming soon
Hoice [Champion et al., 2018]

- learner produces candidates for the predicates
- teacher checks each clause is respected
- \( \Rightarrow \) each check is a quantifier-free (non-Horn) formula
- using Z3 [de Moura and Bjørner, 2008] (separate process)
E. De Angelis (1), F. Fioravanti (1),
A. Pettorossi (2), M. Proietti (3)

(1) DEC, University ”G. d’Annunzio” of Chieti-Pescara, Italy
(2) DICII, University of Rome Tor Vergata, Rome, Italy
(3) CNR-IASI, Rome, Italy

HCVS 2018                        Oxford – July 13, 2018
TransfHORNer distinctive features

Apply satisfiability-preserving CHC transformations (Unfold/Define/Fold)
Discover invariants and try to increase the effectiveness of backend CHC solvers

Convert from SMT-LIB format to Prolog format;
Try to transform non-linear recursive CHCs into linear-recursive CHCs;
if linearization successful then { run* Z3 (Spacer and PDR); (requires conversion to SMT-LIB)
    if sat / unsat then exit; }

while (true) do
    Propagate* constraints and discover invariants (using widening with convex-hull);
    if Propagate timed-out then use simpler widening for Propagate
    else { run* Z3 (Spacer and PDR);
        if sat / unsat then exit; }
    if CHCs are linear-recursive then change direction (backward/forward) of Propagate;
(*) with time limit

Related work
- Verification of C programs with integers and arrays (PEPM13, VMCAI14, SCP14,...)
- VeriMAP (TACAS14) https://fmlab.unich.it/VeriMAP/
- Semantics-based VC generation for C programs (PPDP15, SCP17)
- Relational program verification (SAS16, TPLP18)
- Verification of BPMN business processes (LOPSTR16, RULEML+RR17)
- Satisfiability checking of CHCs with inductively defined data structures (lists, trees, ...) without induction

ICLP 2018 on Saturday, July 14th  3 PM
PECOS: Partial evaluation and Constraint Specialisation

Developed by John P. Gallagher, Bishoksan Kafle and José F. Morales (Roskilde, IMDEA, Melbourne). Based partly on previous experiments with RAHFT (Kafle, Gallagher & Morales, CAV’2016). Oriented towards pure LRA Horn clauses (challenging to modify for mixed Boolean/numeric constraints). Iteratively performs the following transformations.


Prototype software on github.com/jpgallagher/pecos. Implemented in Ciao Prolog with interfaces to PPL and Yices.
Ultimate TreeAutomizer

Tree Automizer = Trace Abstraction + Tree Automata

Trace Abstraction (Automizer) approach:
Program is correct iff each error trace (word) is infeasible.

Tree Automizer approach:
Set of CHCs is sat iff the constraints in each derivation of false (tree) are unsat.

Under the hood:
- Ultimate Automata Library
- SMTInterpol

Contributors:
Daniel Dietsch, Alexander Nutz, Mostafa M. Mohamed, Daniel Tischner, Jochen Hoenicke, Matthias Heizmann, Andreas Podelski
Ultimate Unihorn Automizer

Input: Set of constrained Horn clauses $\Phi$

Approach:
- Construct (possibly recursive) program $P_\Phi$ such that:
  $P_\Phi$ is safe iff $\Phi$ is sat
- Apply off-the-shelf program verifier

Under the hood:
- Program verifier: Ultimate Automizer
- Predicate providers: Newton-style interpolation, SMTInterpol
- SMT Solvers: CVC4, MathSAT5, SMTInterpol, Z3
- Ultimate Automata Library

Contributors:
Daniel Dietsch, Matthias Heizmann, Jochen Hoenicke, Alexander Nutz, Andreas Podelski
The *Eldarica* Horn Solver

Hossein Hojjat\textsuperscript{1} Philipp Rümmer \textsuperscript{2}

\textsuperscript{1}Rochester Institute of Technology
\textsuperscript{2}Uppsala University

HCVS 2018: 5th Workshop on Horn Clauses for Verification and Synthesis
13 July 2018
Eldarica Overview

- Horn solver developed since 2011
- Open-source, implemented in Scala, running in JVM

- **Input formats:**
  SMT-LIB, Prolog, C, timed automata

- **Theories:**
  LIA, NIA, arrays, algebraic data-types, bit-vectors

- Scala/Java API
- Support for linear + non-linear clauses

- https://github.com/uuverifiers/eldarica
Eldarica Architecture

Horn clauses
Prolog, SMT-LIB

Programs
NTS, C, Timed Automata

Accelerator
(FLATA)

Preprocessor

CEGAR Engine

Craig Interpolator
(PRINCESS)

Horn Encoder

Global Loop Analyser

SAT + Sol

UNSAT + Cex

Hojjat, Rüffer
The Eldarica Horn Solver
Spacer: Solving SMT-constrained CHC

Spacer: a solver for SMT-constrained Horn Clauses

- now the default (and only) CHC solver in Z3
  - https://github.com/Z3Prover/z3
  - dev branch at https://github.com/agurfinkel/z3

Supported SMT-Theories

- Linear Real and Integer Arithmetic
- Quantifier-free theory of arrays
- Universally quantified theory of arrays + arithmetic (work in progress)
- Best-effort support for many other SMT-theories
  - data-structures, bit-vectors, non-linear arithmetic

Support for Non-Linear CHC

- for procedure summaries in inter-procedural verification conditions
- for compositional reasoning: abstraction, assume-guarantee, thread modular, etc.
Spacer Contributors

Arie Gurfinkel
Anvesh Komuravelli

Nikolaj Bjorner
(Krystof Hoder)
Yakir Vizel
Bernhard Gleiss
Matteo Marescotti
Competition Setup

StarExec cluster environment

Dedicated Queue of 20 nodes

2 jobs per node

64GB per job (more than promised)

900s timeout enforced by runsolver

About 12 hours for one complete run of all tools and categories

Detailed results (and benchmarks) will be publicly available on StarExec
## Results: LRA

| solver     | cnt | ok | sat | uns | fld | to | mo | time  | real     | space     | uniq | uniq
|------------|-----|----|-----|-----|-----|----|----|-------|----------|-----------|------|------
| rebus      | 132 | 96 | 82  | 14  | 36  | 36 | 0  | 42838 | 42842    | 59        | 15    |      
| spacer     | 132 | 82 | 73  | 9   | 50  | 50 | 0  | 54070 | 54032    | 93        | 3     |      
| tree-aut   | 132 | 25 | 24  | 1   | 107 | 107| 0  | 101241| 92580    | 7488      | 0     |      
| eldarica   | 132 | 20 | 18  | 2   | 112 | 77 | 0  | 72526 | 35825    | 2120      | 0     |      
| TransfHORNer | 132 | 15 | 2   | 13  | 117 | 105| 1  | 103490| 103619   | 1294      | 2     |      
| uni-aut    | 132 | 9  | 9   | 0   | 123 | 90 | 0  | 82877 | 70345    | 7536      | 0     |      
| hoice      | 132 | 1  | 0   | 1   | 131 | 131| 0  | 118793| 118459   | 31        | 0     |      
| pecos      | 132 | 0  | 0   | 0   | 132 | 57 | 1  | 54743 | 54811    | 893       | 0     |      

- **cnt** – number of benchmarks
- **ok** – solved
- **sat** – solved sat
- **uns** – solved unsat
- **fld** – failed
- **to** – timeout
- **mo** – memory out
- **time** – sum of time
- **real** – sum of wall
- **Space** – sum of mem
- **uniq** – unique solved
- **front-end issues**
### Results: LIA

<table>
<thead>
<tr>
<th>solver</th>
<th>cnt</th>
<th>ok</th>
<th>sat</th>
<th>uns</th>
<th>fld</th>
<th>to</th>
<th>mo</th>
<th>time</th>
<th>real</th>
<th>space</th>
<th>uniq</th>
</tr>
</thead>
<tbody>
<tr>
<td>spacer</td>
<td>339</td>
<td>225</td>
<td>153</td>
<td>72</td>
<td>114</td>
<td>112</td>
<td>1</td>
<td>122323</td>
<td>122337</td>
<td>509</td>
<td>56</td>
</tr>
<tr>
<td>rebus</td>
<td>339</td>
<td>185</td>
<td>139</td>
<td>46</td>
<td>154</td>
<td>154</td>
<td>0</td>
<td>147539</td>
<td>147553</td>
<td>182</td>
<td>12</td>
</tr>
<tr>
<td>hoice</td>
<td>339</td>
<td>107</td>
<td>75</td>
<td>32</td>
<td>232</td>
<td>231</td>
<td>0</td>
<td>219810</td>
<td>217971</td>
<td>93</td>
<td>2</td>
</tr>
<tr>
<td>eldarica</td>
<td>339</td>
<td>105</td>
<td>97</td>
<td>8</td>
<td>234</td>
<td>234</td>
<td>0</td>
<td>216152</td>
<td>133402</td>
<td>5856</td>
<td>5</td>
</tr>
<tr>
<td>TransfHORNER</td>
<td>339</td>
<td>99</td>
<td>63</td>
<td>36</td>
<td>240</td>
<td>160</td>
<td>8</td>
<td>196339</td>
<td>196471</td>
<td>1642</td>
<td>0</td>
</tr>
<tr>
<td>pecos</td>
<td>339</td>
<td>61</td>
<td>61</td>
<td>0</td>
<td>278</td>
<td>152</td>
<td>1</td>
<td>171955</td>
<td>173970</td>
<td>3819</td>
<td>3</td>
</tr>
<tr>
<td>tree-aut</td>
<td>339</td>
<td>53</td>
<td>34</td>
<td>19</td>
<td>286</td>
<td>247</td>
<td>0</td>
<td>241323</td>
<td>220198</td>
<td>19234</td>
<td>0</td>
</tr>
<tr>
<td>uni-aut</td>
<td>339</td>
<td>51</td>
<td>43</td>
<td>8</td>
<td>288</td>
<td>211</td>
<td>0</td>
<td>199278</td>
<td>140978</td>
<td>19378</td>
<td>0</td>
</tr>
</tbody>
</table>
Big Thanks to StarExec
Discussion

CHC-COMP 2019
- Dates, format, co-location, tracks

Ranking
- Should we decide on 1st three places?

Non-Linear CHC as a category?

Benchmark storage
- Github is limited to 1-2GB per repo

Benchmark selection / availability

Common conversion / simplification utilities