Report on the Second Edition of the CHC Competition

Grigory Fedyukovich

April 7, Prague
Constrained Horn Clauses

Formula in first order logic:

\[ \varphi \land p_1(V) \land \ldots \land p_k(V) \implies H \]

• where \( A \) is a constraint language
  (e.g., (non-)linear arithmetic, arrays, bit-vectors, etc.)
• \( \varphi \) is a constraint in \( A \)
• \( p_1 \ldots p_k \) are uninterpreted relation symbols
• each \( p_i(V) \) is an application of the predicate to variables
• \( H \) is either some application \( p_i(V) \) or \textit{false}

System of CHCs

• Only one CHC with \( H = \text{false} \)
• Has a solution if there exists an interpretation for each \( p_i \)
  making each CHC valid
Example

Program in C

```c
int x, k, c = 0;
int N = NONDET();
assume (N ≥ 0);
while (c ≠ N) {
    c++;
    if (k mod 2 == 0) x++;  
k = x + c;
}
if (x ≠ N) ERROR();
```

CHC-encoding

```
x = 0 ∧ k = 0 ∧ c = 0 ∧ N ≥ 0
⇒ Inv(x, k, c, N)

Inv(x, k, c, N) ∧ c ≠ N ∧ c' = c + 1 ∧
x' = ite (k mod 2 = 0, x + 1, x) ∧
k' = x' + c'
⇒ Inv(x', k', c', N)

Inv(x, k, c, N) ∧ c = N ∧ x ≠ N  ⇒ ⊥
```
Validating Solutions of CHCs

System of CHCs

\[ x = 0 \land k = 0 \land c = 0 \land N \geq 0 \quad \Rightarrow \quad \mathit{Inv}(x, k, c, N) \]

\[ \mathit{Inv}(x, k, c, N) \land c \neq N \land c' = c + 1 \land \]
\[ x' = \text{ite} (k \mod 2 = 0, x + 1, x) \land \]
\[ k' = x' + c' \quad \Rightarrow \quad \mathit{Inv}(x', k', c', N) \]

\[ \mathit{Inv}(x, k, c, N) \land c = N \land x \neq N \quad \Rightarrow \quad \bot \]

Inductive invariant

\[ \mathit{Inv}(x, k, c, N) = (k = x + c \land x = c) \]
Validating Solutions of CHCs

System of CHCs

\[ x = 0 \land k = 0 \land c = 0 \land N \geq 0 \implies (k = x + c \land x = c) \]

\[
\begin{align*}
(k = x + c \land x = c) \land c \neq N \land c' = c + 1 \land \\
x' &= \text{ite} \ (k \mod 2 = 0, x + 1, x) \land \\
k' &= x' + c' \implies (k' = x' + c' \land x' = c') \\
(k = x + c \land x = c) \land c = N \land x \neq N \implies \bot
\end{align*}
\]

Inductive invariant

\[ \text{Inv}(x, k, c, N) = (k = x + c \land x = c) \]
CHC Solving Competition

• Second Edition: April 7 2019, HCVS@ETAPS

• The CHC competition (CHC-COMP) compares state-of-the-art tools for CHC solving with respect to performance and effectiveness on a set of publicly available benchmarks

• Web: https://chc-comp.github.io/

• Gitter: https://gitter.im/chc-comp/Lobby

• GitHub: https://github.com/chc-comp

• Format: https://chc-comp.github.io/2018/format.html
CHC Format

- bench := (set-logic HORN)
  - fun_decl+
    - (assert assert)*
    - (assert query)
    - (check-sat)
- fun_decl := (declare-fun symbol ( sort* ) Bool)
- var_decl := (symbol sort)
- head := (u_predicate var*)
- tail := (u_predicate var*) | SMT-LIB-formula | (and tail таід)
- assert := (forall ( var_decl+ ) (=> tail head)) | head
- query := (forall ( var_decl+ ) (=> tail false))
Tracks

Linear Integer Arithmetic, linear clauses (LIA-Lin)
  at most one application of an uninterpreted relation symbol in each CHC tail

Arrays + LIA-Lin
  formulas involve array variables

Linear Integer Arithmetic, nonlinear clauses (LIA-Nonlin)
  tail of some CHC has more than one application of uninterpreted relation symbols

Linear Real Arithmetic, transition systems (LRA-TS)
  one uninterpreted relation symbol, three CHCs
Benchmarks

Linear Integer Arithmetic, linear clauses (LIA-Lin)

325 instances contributed by Hoice, Ultimate, Eldarica, Sally, Kind2/Zustre, FreqHorn/FreqTerm, VMT

Arrays + LIA-Lin

361 instances contributed by Spacer, Ultimate, FreqHorn

Linear Integer Arithmetic, nonlinear clauses (LIA-Nonlin)

283 instances contributed by PCSat, Hoice, Ultimate, Eldarica

Linear Real Arithmetic, transition systems (LRA-TS)

243 instances contributed by Sally, FreqHorn/FreqTerm, VMT
Participants

PCSat
- NEW in 2019
- Yuki Satake,
  Tomoya Kashifuku,
  and Hiroshi Unno

Sally
- Dejan Jovanovic,
  Martin Blich

Eldarica
- Hossein Hojjat and
  Philipp Rümmer

Hoice
- Adrien Champion

Ultimate Tree Automizer
- Daniel Dietsch, Matthias Heizmann, Jochen Hoenicke, Mostafa M. Mohamed, Alexander Nutz, Andreas Podelski, and Daniel Tischner

Ultimate Unihorn Automizer
- Daniel Dietsch, Matthias Heizmann, Jochen Hoenicke, Alexander Nutz, and Andreas Podelski

Rebus
- unnamed solver
  not entered in the competition

Spacer
- Arie Gurfinkel, Anvesh Komuravelli, Nikolaj Bjorner,
  Krystof Hoder, Yakir Vizel, Bernhard Gleiss, and
  Matteo Marescotti

Hors Concours
Competition Setup

• StarExec cluster environment
• Dedicated Queue of 20 nodes
• 2 jobs per node
• 64GB per job
• 600s timeout
• Benchmarks will be publicly available on StarExec
• Detailed results will be available by request
The Friendliest Competition

Fair selection of benchmarks
  If a participant submitted a benchmark suit, organizers include a good (but randomly chosen) representation of it

Help with frontend issues
  Pre-processing of submitted benchmarks by CHC-COMP’s tools to match the format more closely

Introducing new tracks
  As long as there is a solver that focuses on them

Giving participants a second chance to submit solvers
  After running trial runs on selections of benchmarks to avoid discrepancies
PCSat: Predicate Constraint Satisfaction

- Developed since January 2019 by Yuki Satake, Tomoya Kashifuku, and Hiroshi Unno (University of Tsukuba, Japan) in the OCaml functional language

- Support new classes of predicate constraint satisfaction problems beyond CHC
  - $\text{pCSP}^{\text{WF}}$: (possibly non-Horn) constrained clauses with well-foundedness constraints
  - $\mu\text{CSP}$: (possibly alternating) least and greatest fixpoint constraints [Nanjo+ LICS’18, Unno HCVS’18]

- Support LIA
PCSat Architecture
(red parts for CHC-COMP’19)
Hoice [Champion et al., 2018]

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

Adrien Champion\(^1\), Tomoya Chiba\(^1\), Naoki Kobayashi\(^{1,2}\), Ryosuke Sato\(^2\)

\(^1\) 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 \texttt{Int}, \texttt{Real}, \texttt{Array}, datatypes
Hoice [Champion et al., 2018]

- learner produces candidates for the predicates
- teacher checks each clause is respected
- each check is a quantifier-free (non-Horn) formula
- using Z3 [de Moura and Bjørner, 2008] (separate process)
The **ELDARICA** Horn Solver

Hossein Hojjat\(^1\)  Philipp Rümmer \(^2\)

\(^1\)Rochester Institute of Technology  \(^2\)Uppsala University
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

- **Eldarica**
  - Accelerator (FLATA)
  - Preprocessor
  - Global Loop Analyser
  - Craig Interpolator (PRINCESS)
  - CEGAR Engine

- **SAT + Sol**
- **UNSAT + Cex**

---

Hojjat, Rümmer

The Eldarica Horn Solver
**Approach**

- Similar to *trace abstraction* for programs
  - Represent set of *all sequences of statements* that can reach an error location as *nested word automaton*.
  - Program is correct iff each word of this language is *infeasible*.

- *Trace abstraction for Horn clauses*
  - Represent set of *all derivation trees* of a set of CHCs as *tree automaton*.
  - Set of CHCs is sat iff each tree of this language is a *derivation of false*.

**Tools**

- **Ultimate Automata Library**
- **SMTInterpol**

**Contributors**
Daniel Dietsch, Matthias Heizmann, Jochen Hoenicke, Mostafa M. Mohamed, Alexander Nutz, Andreas Podelski, Daniel Tischner
Approach

1. **Encode** set of CHCs $\Phi$ as (possibly recursive) program $P_\Phi$ s.t.
   $P_\Phi$ is safe iff $\Phi$ is sat

2. Apply off-the-shelf program verifier

Tools

- Program verifier: **Ultimate Automizer**
- Predicate providers: Newton-style interpolation (Unsat. core + projection), SMTInterpol
- SMT Solvers: CVC4, MathSAT 5, SMTInterpol, Z3
- **Ultimate Automata Library**

Contributors

Daniel Dietsch, Matthias Heizmann, Jochen Hoenicke, Alexander Nutz, Andreas Podelski
Model checker for infinite state systems described as transition systems

http://sri-csl.github.io/sally/

Property-directed k-induction, Jovanović, Dutertre, FMCAD 2016

Support of different reasoning engines
  - Bounded model checking (BMC)
  - k-induction (KIND)
  - Property-directed k-induction (PDKIND)

Supported SMT solvers: Yices2, MathSAT5, OpenSMT2 (unofficially)

Developed by Dejan Jovanović

Support for CHC format (limited) and OpenSMT2 contributed by Martin Blicha
CHC-COMP configurations

- Support for CHC limited to transition systems in LRA
- PDKIND engine using Yices2 as the main reasoning engine and
  - MathSAT5 as the interpolation back-end
  - OpenSMT2 as the interpolation back-end with different interpolation algorithms
- MathSAT5 with default settings (Farkas interpolation)
- OpenSMT2 with four LRA interpolation algorithms
  1. Farkas interpolation
  2. dual Farkas interpolation
  3. decomposed interpolation
  4. dual decomposed interpolation
- *LRA Interpolants from No Man’s Land*, Alt, Hyvärinen, Sharygina, HVC 2017
- *Decomposing Farkas Interpolants*, Blicha, Hyvärinen, Kofroň, Sharygina, TACAS 2019
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](https://github.com/Z3Prover/z3)
  - dev branch at [https://github.com/agurfinkel/z3](https://github.com/agurfinkel/z3)

Supported SMT-Theories

- Linear Real and Integer Arithmetic
- Quantifier-free theory of arrays
- *Universally quantified theory of arrays + arithmetic*
- 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
Scoring Schema

• Three possible outputs
  • Sat / Unsat / Unknown

• We count \#Sat + \#Unsat
  • Solvers with equal total score are ordered w.r.t. running time

• Disqualification for wrong results

• No witness generation (yet)
## Results: LIA-Lin

<table>
<thead>
<tr>
<th>Solver</th>
<th>Score</th>
<th>#SAT</th>
<th>#UNSAT</th>
<th>Avg time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Spacer</td>
<td>279</td>
<td>194</td>
<td>85</td>
<td>28.90</td>
</tr>
<tr>
<td>Rebus</td>
<td>267</td>
<td>188</td>
<td>79</td>
<td>41.85</td>
</tr>
<tr>
<td>Eldarica</td>
<td>209</td>
<td>129</td>
<td>80</td>
<td>24.55</td>
</tr>
<tr>
<td>Ultimate Unihorn Automizer</td>
<td>133</td>
<td>63</td>
<td>70</td>
<td>23.05</td>
</tr>
<tr>
<td>Hoice</td>
<td>129</td>
<td>65</td>
<td>64</td>
<td>7.09</td>
</tr>
<tr>
<td>Ultimate Tree Automizer</td>
<td>107</td>
<td>42</td>
<td>65</td>
<td>29.15</td>
</tr>
<tr>
<td>PCSat</td>
<td>45</td>
<td>33</td>
<td>12</td>
<td>23.74</td>
</tr>
</tbody>
</table>

*325 instances total*
## Results: LIA-Nonlin

<table>
<thead>
<tr>
<th>Solver</th>
<th>Score</th>
<th>#SAT</th>
<th>#UNSAT</th>
<th>Avg time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Spacer</td>
<td>270</td>
<td>153</td>
<td>117</td>
<td>5.04</td>
</tr>
<tr>
<td>Eldarica</td>
<td>234</td>
<td>131</td>
<td>103</td>
<td>15.93</td>
</tr>
<tr>
<td>Ultimate Unihorn Automizer</td>
<td>177</td>
<td>96</td>
<td>81</td>
<td>36.94</td>
</tr>
<tr>
<td>Hoice</td>
<td>176</td>
<td>110</td>
<td>66</td>
<td>9.85</td>
</tr>
<tr>
<td>PCSat</td>
<td>123</td>
<td>81</td>
<td>42</td>
<td>24.69</td>
</tr>
<tr>
<td>Ultimate Tree Automizer</td>
<td>73</td>
<td>29</td>
<td>44</td>
<td>4.85</td>
</tr>
</tbody>
</table>

*283 instances total*
## Results: LIA+Array

<table>
<thead>
<tr>
<th>Solver</th>
<th>Score</th>
<th>#SAT</th>
<th>#UNSAT</th>
<th>Avg time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Spacer</td>
<td>159</td>
<td>76</td>
<td>83</td>
<td>9.60</td>
</tr>
<tr>
<td>Ultimate Unihorn Automizer</td>
<td>90</td>
<td>44</td>
<td>46</td>
<td>28.47</td>
</tr>
<tr>
<td>Ultimate Tree Automizer</td>
<td>71</td>
<td>39</td>
<td>32</td>
<td>44.14</td>
</tr>
<tr>
<td>Hoice</td>
<td>35</td>
<td>24</td>
<td>11</td>
<td>0.06</td>
</tr>
<tr>
<td>Eldarica</td>
<td>20</td>
<td>20</td>
<td>0</td>
<td>100.14</td>
</tr>
</tbody>
</table>

*361 instances total*
## Results: LRA-TS

<table>
<thead>
<tr>
<th>Solver</th>
<th>Score</th>
<th>#SAT</th>
<th>#UNSAT</th>
<th>Avg time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sally-y2o2-decomposed-itp</td>
<td>194</td>
<td>150</td>
<td>44</td>
<td>43.71</td>
</tr>
<tr>
<td>Sally-y2o2-Farkas-itp</td>
<td>194</td>
<td>150</td>
<td>44</td>
<td>44.34</td>
</tr>
<tr>
<td>Rebus</td>
<td>190</td>
<td>137</td>
<td>53</td>
<td>53.24</td>
</tr>
<tr>
<td>Sally-y2o2-dual-Farkas-itp</td>
<td>188</td>
<td>144</td>
<td>44</td>
<td>53.46</td>
</tr>
<tr>
<td>Sally-y2m5</td>
<td>179</td>
<td>135</td>
<td>44</td>
<td>40.07</td>
</tr>
<tr>
<td>Spacer</td>
<td>173</td>
<td>126</td>
<td>47</td>
<td>46.19</td>
</tr>
<tr>
<td>Sally-y2o2-dual-decomposed-itp</td>
<td>157</td>
<td>118</td>
<td>39</td>
<td>67.67</td>
</tr>
<tr>
<td>Ultimate Tree Automizer</td>
<td>93</td>
<td>73</td>
<td>20</td>
<td>55.15</td>
</tr>
<tr>
<td>Ultimate Unihorn Automizer</td>
<td>67</td>
<td>50</td>
<td>17</td>
<td>22.21</td>
</tr>
</tbody>
</table>

*243 instances total*
Congrats to

**Eldarica**

LIA-Lin and LIA-Nonlin categories

**Ultimate Unihorn Automizer**

LIA+Array category

**Sally**

LRA-TS category

**Spacer**

(unofficially) all LIA categories
Big thanks to StarExec
Discussion

• Any fairness / transparency concerns?
• Frontend / format issues?
• New benchmarks / new tracks?
• Solution validation?
• CHC-COMP 2020 dates / organizers?
Thank you!