Report on the Second Edition of the CHC Competition

Grigory Fedyukovich

April 7, Prague

Constrained Horn Clauses

CHC COMP

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.)
- φ 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 *false*

System of CHCs

- Only one CHC with $H = false$
- Has a solution if there exists an interpretation for each p_i
  making each CHC valid

Example

Program in 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

$$x = 0 \land k = 0 \land c = 0 \land N \geq 0 \implies \boldsymbol{Inv}(x, k, c, N)$$

$$\begin{aligned} \boldsymbol{Inv}(x,k,c,N) \land c \neq N \land c' = c+1 \land \\ x' = \texttt{ite}\ (k \ \texttt{mod}\ 2 = 0, x+1, x) \land \\ k' = x' + c' \\ \implies \boldsymbol{Inv}(x',k',c',N) \end{aligned}$$

$$\boldsymbol{Inv}(x,k,c,N) \land c = N \land x \neq N \implies \bot$$

Validating Solutions of CHCs

System of CHCs

$$x = 0 \land k = 0 \land c = 0 \land N \geq 0 \implies \boxed{\boldsymbol{Inv}(x, k, c, N)}$$

$$\begin{aligned} \boxed{\boldsymbol{Inv}(x, k, c, N)} \land c \neq N \land c' = c + 1 \land \\ x' = \texttt{ite}\ (k \ \texttt{mod}\ 2 = 0, x + 1, x) \land \\ k' = x' + c' \implies \boxed{\boldsymbol{Inv}(x', k', c', N)} \end{aligned}$$

$$\boxed{\boldsymbol{Inv}(x, k, c, N)} \land c = N \land x \neq N \implies \bot$$

Inductive invariant

$$\boldsymbol{Inv}(x, k, c, N) = (k = x + c \land x = c)$$

Validating Solutions of CHCs

System of CHCs

$$x = 0 \land k = 0 \land c = 0 \land N \geq 0 \implies \boxed{(\boldsymbol{k = x + c \land x = c})}$$

$$\boxed{(\boldsymbol{k = x + c \land x = c})} \land c \neq N \land c' = c + 1 \land \\ x' = \texttt{ite}\ (k \ \texttt{mod}\ 2 = 0, x + 1, x) \land \\ k' = x' + c' \implies \boxed{(\boldsymbol{k' = x' + c' \land x' = c'})}$$

$$\boxed{(\boldsymbol{k = x + c \land x = c})} \land c = N \land x \neq N \implies \bot$$

Inductive invariant

$$\boldsymbol{Inv}(x, k, c, N) = (k = x + c \land x = c)$$

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 taiд)
- 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 **NEW in 2019**

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

CHC COMP

PCSat

NEW in 2019

Yuki Satake,
Tomoya Kashifuku, and
Hiroshi Unno

Sally

Dejan Jovanovic,

Martin Blicha

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

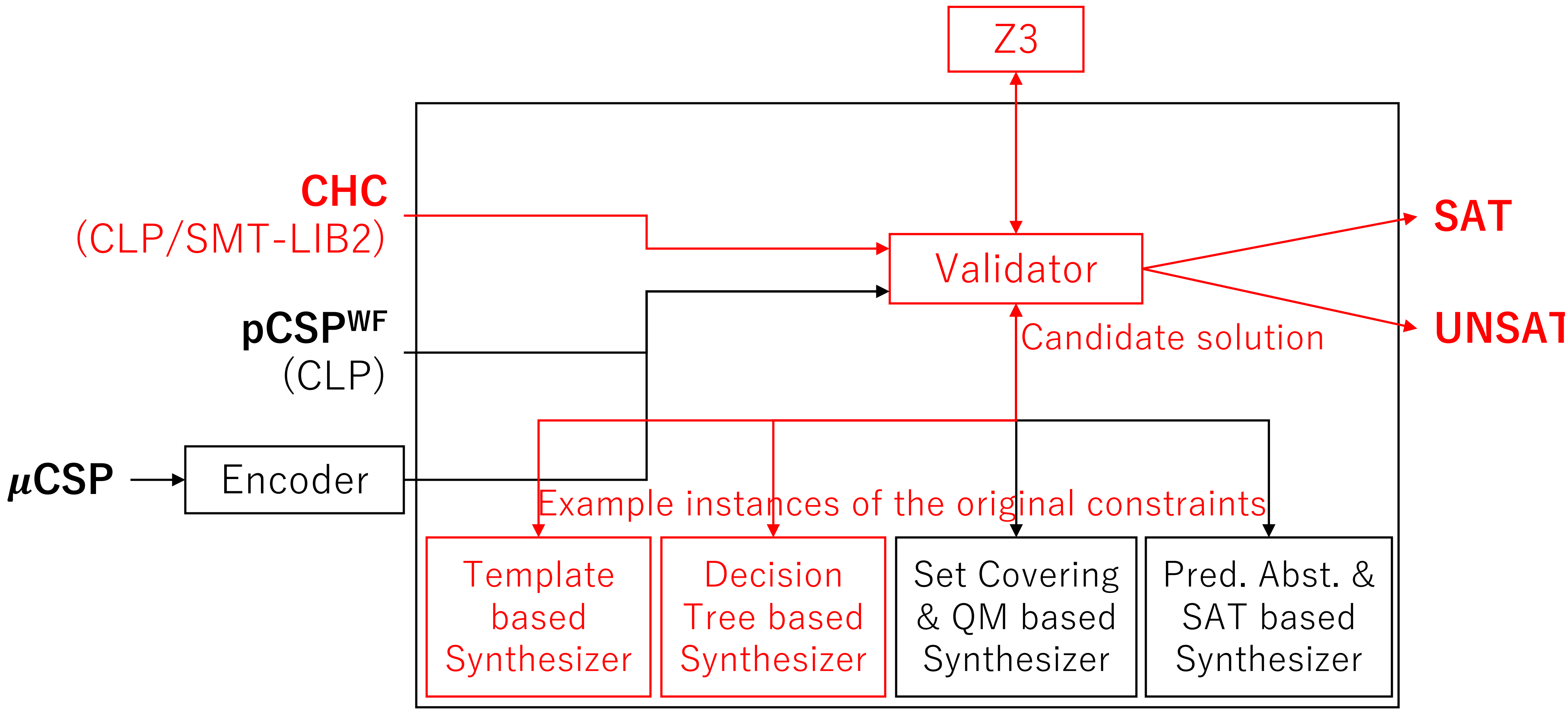
Giving participants a second chance to submit solvers

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**
  - **pCSP$^{\mathrm{WF}}$**: (possibly **non-Horn**) constrained clauses with **well-foundedness** constraints
  - **μCSP**: (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], Ryosuke Sato[2]

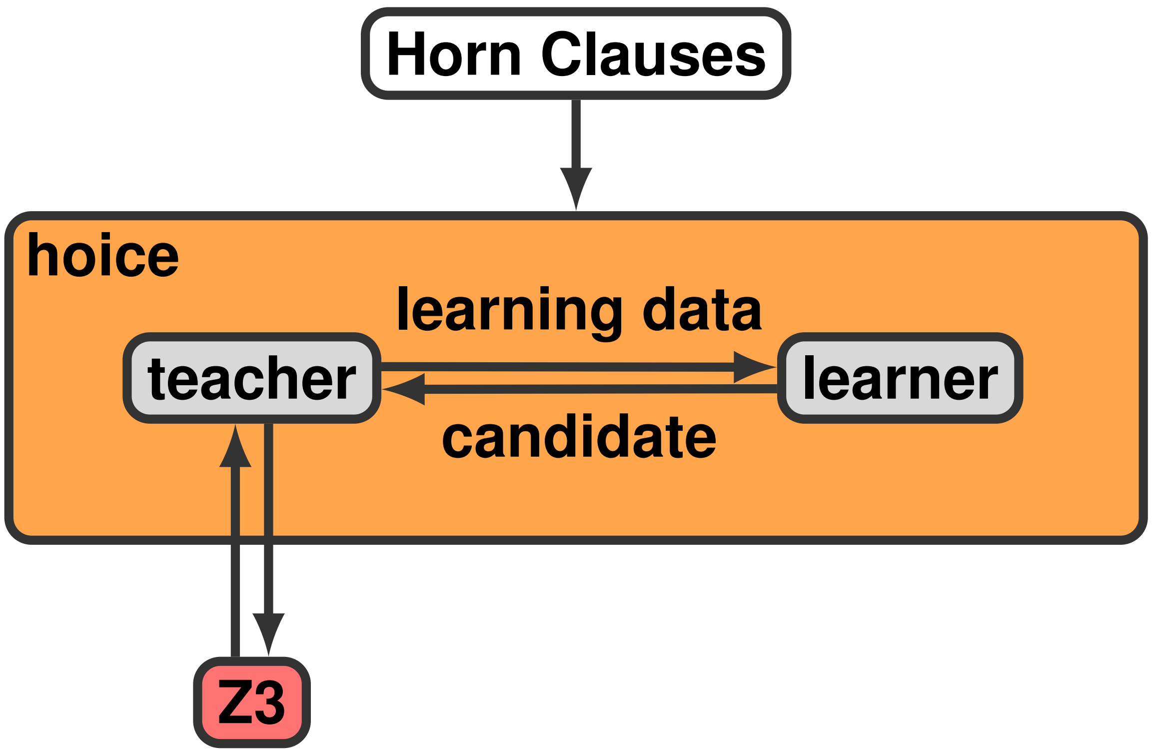
[1] The University of Tokyo
[2] Kyushu University

`https://github.com/hopv/hoice`

- **machine-learning-based Horn clause solver: generalized ICE framework [Garg et al., 2014]**
- **context: higher-order program verification**
- **supports `Int`, `Real`, `Array`, datatypes**

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)



The ELDARICA Horn Solver

Hossein Hojjat[1] Philipp Rümmer [2]

[1]Rochester Institute of Technology
[2]Uppsala University

ELDARICA Overview

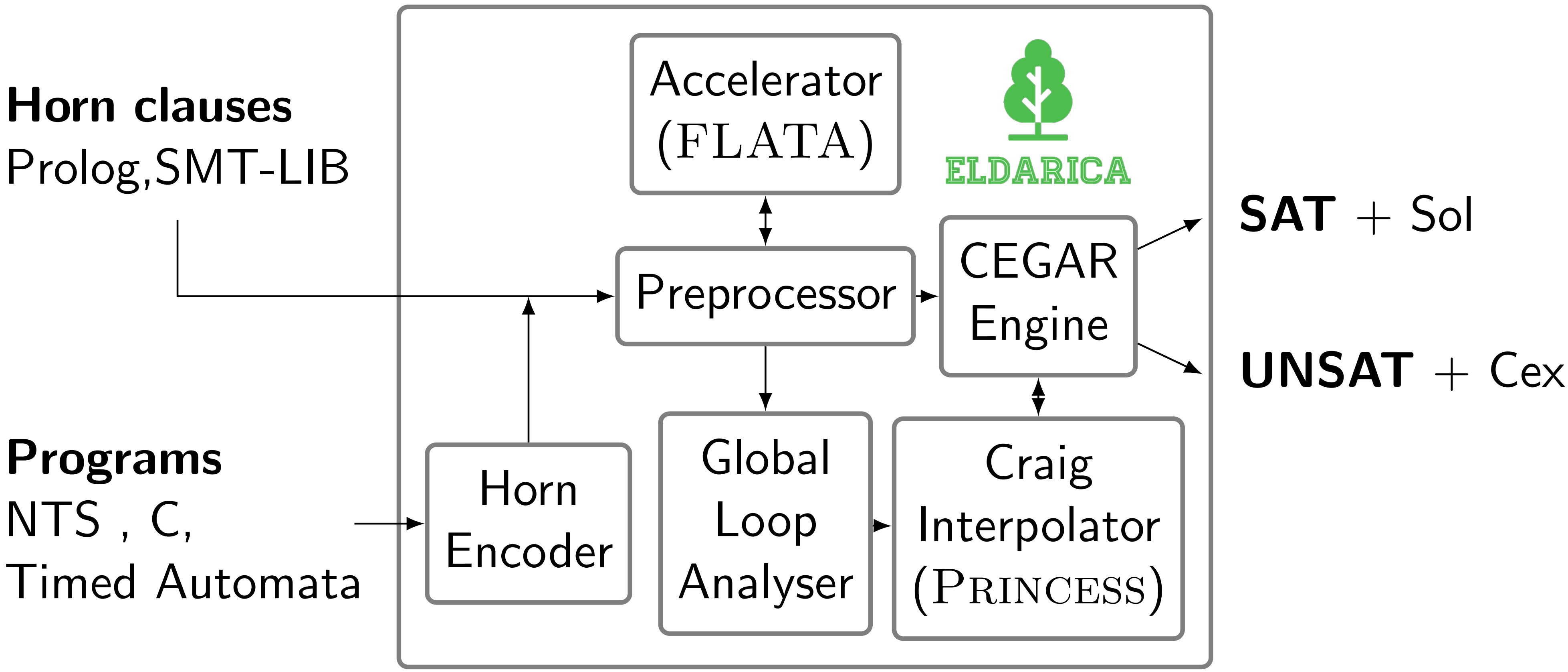
- Horn solver developed since 2011
- Open-source, implemented in Scala, running in JVM

- **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



Ultimate TreeAutomizer

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

Ultimate Unihorn

Approach

1. Encode set of CHCs Φ as (possibly recursive) program P_Φ s.t. P_Φ is safe iff Φ 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

Sally

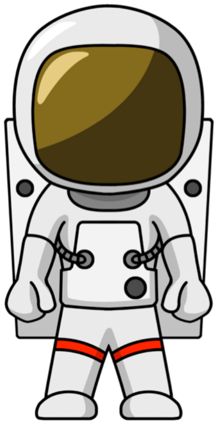
- Model checker for infinite state systems described as transition systems
- http://sri-csl.github.io/sally/
- *Property-directed k-induction*, Jovanović, Dutertre, FMCAD 2016
- Support of different reasoning engines
    - Bounded model checking (BMC)
    - k-induction (KIND)
    - Property-directed 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
  - 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*
- Best-effort support for many other SMT-theories
  - data-structures, bit-vectors, non-linear arithmetic

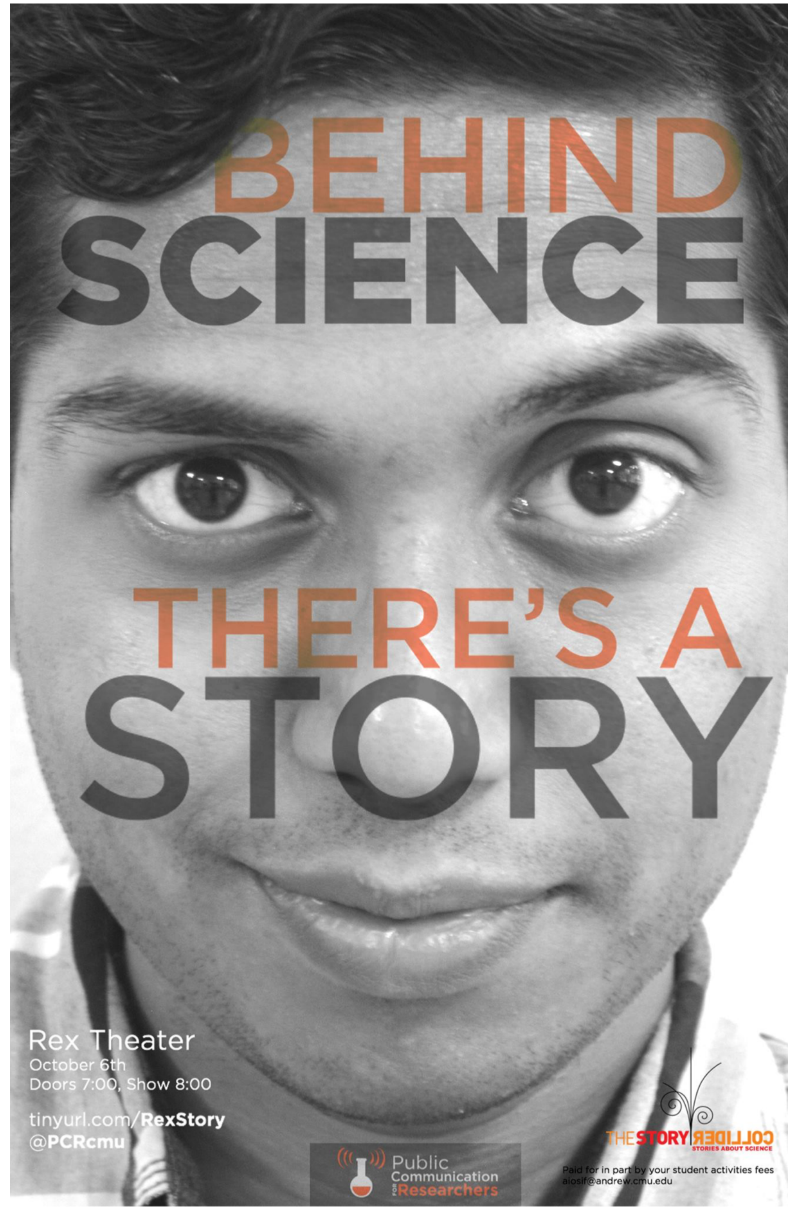
Support for Non-Linear CHC

- 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

CHC COMP

- 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**

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

* **325** instances total

Results: **LIA-Nonlin**

Solver	Score	#SAT	#UNSAT	Avg time
Spacer	**270**	153	117	5.04
Eldarica	**234**	131	103	15.93
Ultimate Unihorn Automizer	**177**	96	81	36.94
Hoice	**176**	110	66	9.85
PCSat	**123**	81	42	24.69
Ultimate Tree Automizer	**73**	29	44	4.85

***** **283** instances total

Results: **LIA+Array**

Solver	Score	#SAT	#UNSAT	Avg time
Spacer	**159**	76	83	9.60
Ultimate Unihorn Automizer	**90**	44	46	28.47
Ultimate Tree Automizer	**71**	39	32	44.14
Hoice	**35**	24	11	0.06
Eldarica	**20**	20	0	100.14

* **361** instances total

Results: **LRA-TS**

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

* **243** instances total

Congrats to

CHC COMP

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!