Program Verification with Constrained Horn Clauses

Prof. Arie Gurfinkel
Department of Electrical and Computer Engineering
University of Waterloo
Waterloo, Ontario, Canada

August 10, 2022
CAV, FLOC, 2022

joint work with **A. Komuravelli**, S. Chaki, G. Fedyukovich,
S. Shoham, N. Bjørner, **Hari Govind V. K.**, Y. (Jeff) Chen

UNIVERSITY OF WATERLOO

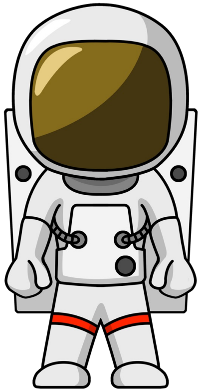
Software Model Checking of Programs / Transitions Systems / Push-down Systems

=

Satisfiability of Constrained Horn Logic (CHC) fragment of First Order Logic

---

Reduce Model Checking to FOL Satisfiability



UNIVERSITY OF WATERLOO

Constrained Horn Clauses (CHC)

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

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

- φ - constraint in a background theory $\mathcal{T}$
- $\mathcal{T}$ - background theory
  - Linear Arithmetic, Arrays, Bit-Vectors, or combinations
- V - variables, and X_i are terms over V
- $p_1, \ldots, p_n, h$ - n-ary predicates
- $p_i[X]$ - application of a predicate to first-order terms

UNIVERSITY OF WATERLOO

CHC Satisfiability

Π - set of CHCs

M - $\mathcal{T}$-**model** of a set of Π

- M satisfies $\mathcal{T}$
- M satisfies Π – through first-order interpretation of each predicate p_i

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

- This is the usual FOL satisfiability

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

In the context of program verification

Program $\vDash \boldsymbol{\varphi}$	**iff**	$CHC_{Program} \rightarrow \varphi$
Inductive Invariant	=	Solution to CHC
Counter Example Trace	=	Resolution proof of CHC

Example CHC: Is this SAT?

$$\forall x \cdot x \leq 0 \implies P(x)$$
$$\forall x, x' \cdot P(x) \wedge x < 5 \wedge x' = x + 1 \implies P(x')$$
$$\forall x \cdot P(x) \wedge x \geq 10 \implies false$$

Yes! This set of clauses is satisfiable

The **model** is an extension of the standard model of arithmetic with:

$$P(x) \equiv \{x \mid x \leq 5\}$$
$$\equiv \{5, 4, 3, 2, \ldots\}$$

Note that P(x) is definable by LIA predicate x <= 5

Validating the solution

Original CHC

$$\forall x \cdot x \leq 0 \implies P(x)$$

$$\forall x, x' \cdot P(x) \wedge x < 5 \wedge x' = x + 1 \implies P(x')$$

$$\forall x \cdot P(x) \wedge x \geq 10 \implies false$$

Validation of P(x) = {x | x <= 5}

$$\vdash \forall x \cdot x \leq 0 \implies x \leq 5$$

$$\vdash \forall x, x' \cdot x \leq 5 \wedge x < 5 \wedge x' = x + 1 \implies x' \leq 5$$

$$\vdash \forall x \cdot x \leq 5 \wedge x \geq 10 \implies false$$

UNIVERSITY OF WATERLOO

Example CHC: is this SAT?

$$\forall x \cdot x \leq 0 \implies Q(x)$$
$$\forall x, x' \cdot Q(x) \land x < 5 \land x' = x + 1 \implies Q(x')$$
$$\forall x \cdot Q(x) \land x \geq 2 \implies \mathit{false}$$

No! This set of clauses is unsatisfiable

Justification is a refutation by **resolution** and **instantiation**

Example CHC: is this SAT?

$$\forall x \cdot x \leq 0 \implies Q(x)$$
$$\forall x, x' \cdot Q(x) \land x < 5 \land x' = x + 1 \implies Q(x')$$
$$\forall x \cdot Q(x) \land x \geq 2 \implies \mathit{false}$$

Refutation

$$\dfrac{\dfrac{\forall x \cdot x \leq 0 \implies Q(x)}{Q(0)}\,(x=0) \qquad \forall x \cdot Q(x) \land x < 5 \implies Q(x+1)}{Q(1)}$$

$$\dfrac{Q(1) \qquad \forall x \cdot Q(x) \land x < 5 \implies Q(x+1)}{Q(2)}$$

$$\dfrac{Q(2) \qquad \forall x \cdot Q(x) \land x \geq 2 \implies \mathit{false}}{\mathit{false}}$$

UNIVERSITY OF WATERLOO

A Brief History of Modern CHC in MC

PLDI 2012 S. Grebenshchikov, N. P. Lopes, C. Popeea, A. Rybalchenko , “**Synthesizing software verifiers from proof rules**”

- Constrained Horn Clauses as input format for Software Model Checkers

SAT 2012 K. Hoder, N. Bjørner , “**Generalized Property Directed Reachability**”

- IC3/PDR for SMT == Solving CHCs

SMT 2012 N. Bjørner, K. L. McMillan, A. Rybalchenko, “**Program Verification as Satisfiability Modulo Theories**”

- CHC format extension for SMT-LIB

CAV 2014 A. Komuravelli, G., S. Chaki, “**SMT-Based Model Checking of Recursive Programs**”

- First version of SPACER as an extension of GPDR in Z3

CAV 2015 G, T. Kahsai, A. Komuravelli, J. Navas , “**The SeaHorn Verification Framework**”

- First robust and efficient automated verification tool based on CHC solving

2018 1st CHC-COMP, SPACER merged into Z3 master

- https://chc-comp.github.io/2018/

Horn Clauses for Program Verification

Rybalchenko et al. Synthesizing Software Verifiers from Proof Rules. PLDI'12

$\ell_{out}(x_0, w', e_o)$, which is an entry point into successor edges. with the edges are formulated as follows:

$$p_{init}(x_0, \boldsymbol{w}, \bot) \leftarrow x = x_0 \quad \text{where } x \text{ occurs in } \boldsymbol{w}$$
$$p_{exit}(x_0, ret, \top) \leftarrow \ell(x_0, \boldsymbol{w}, \top) \quad \text{for each label } \ell\text{, and re}$$
$$p(x, ret, \bot, \bot) \leftarrow p_{exit}(x, ret, \bot)$$
$$p(x, ret, \bot, \top) \leftarrow p_{exit}(x, ret, \top)$$
$$\ell_{out}(x_0, \boldsymbol{w}', e_o) \leftarrow \ell_{in}(x_0, \boldsymbol{w}, e_i) \wedge \neg e_i \wedge \neg wlp(S, \neg(e_i =$$

Weakest Preconditions If we apply Boogie directly we obtain a translation from programs to Horn logic using a weakest liberal pre-condition calculus [26]:

$$\mathsf{ToHorn}(program) := wlp(Main(), \top) \wedge \bigwedge_{decl \in program} \mathsf{ToHorn}(decl)$$
$$\mathsf{ToHorn}(\mathbf{def}\ p(x)\ \{S\}) := wlp\left(\begin{array}{l}\mathbf{havoc}\ x_0; \mathbf{assume}\ x_0 = x; \\ \mathbf{assume}\ p_{pre}(x); S, \qquad p(x_0, ret)\end{array}\right)$$
$$wlp(x := E, Q) := \mathbf{let}\ x = E\ \mathbf{in}\ Q$$
$$wlp((\mathbf{if}\ E\ \mathbf{then}\ S_1\ \mathbf{else}\ S_2), Q) := wlp(((\mathbf{assume}\ E; S_1)\Box(\mathbf{assume}\ \neg E; S_2)), Q)$$
$$wlp((S_1 \Box S_2), Q) := wlp(S_1, Q) \wedge wlp(S_2, Q)$$
$$wlp(S_1; S_2, Q) := wlp(S_1, wlp(S_2, Q))$$
$$wlp(\mathbf{havoc}\ x, Q) := \forall x\ .\ Q$$
$$wlp(\mathbf{assert}\ \varphi, Q) := \varphi \wedge Q$$
$$wlp(\mathbf{assume}\ \varphi, Q) := \varphi \rightarrow Q$$
$$wlp((\mathbf{while}\ E\ \mathbf{do}\ S), Q) := inv(\boldsymbol{w}) \wedge \forall \boldsymbol{w}\ .\ \left(\begin{array}{l}((inv(\boldsymbol{w}) \wedge E) \rightarrow wlp(S, inv(\boldsymbol{w}))) \\ \wedge\ ((inv(\boldsymbol{w}) \wedge \neg E) \rightarrow Q)\end{array}\right)$$

```
5. incorrect :- Z=W+1, W≥0, W+1<
                read(A,W,U), read(A,Z
6. p(I1,N,B) :- 1≤I, I<N, D=I−1, I1=I+1, V=U+1,
                read(A,D,U), write(A
7. p(I,N,A) :- I=1, N>1.
```

De Angelis et al. Verifying Array Programs by Transforming Verification Conditions. VMCAI'14

To translate a procedure call $\ell : y := q(E); \ell'$ within a procedure p, create he clauses:

$$p(\boldsymbol{w}_0, \boldsymbol{w}_4) \leftarrow p(\boldsymbol{w}_0, \boldsymbol{w}_1), call(\boldsymbol{w}_1, \boldsymbol{w}_2), q(\boldsymbol{w}_2, \boldsymbol{w}_3), return(\boldsymbol{w}_1, \boldsymbol{w}_3, \boldsymbol{w}_4)$$
$$q(\boldsymbol{w}_2, \boldsymbol{w}_2) \leftarrow p(\boldsymbol{w}_0, \boldsymbol{w}_1), call(\boldsymbol{w}_1, \boldsymbol{w}_2)$$
$$call(\boldsymbol{w}, \boldsymbol{w}') \leftarrow \pi = \ell, x' = E, \pi' = \ell_{q_{init}}$$
$$return(\boldsymbol{w}, \boldsymbol{w}', \boldsymbol{w}'') \leftarrow \pi' = \ell_{q_{exit}}, \boldsymbol{w}'' = \boldsymbol{w}[ret'/y, \ell'/\pi]$$

Bjørner, Gurfinkel, McMillan, and Rybalchenko:

Horn Clause Solvers for Program Verification

UNIVERSITY OF WATERLOO

Horn Clauses for Concurrent / Distributed / Parameterized Systems

For assertions $R_1, \ldots, R_N$ over V and $E_1, \ldots, E_N$ over V, V',

$$
\begin{array}{lll}
\text{CM1}: & \mathit{init}(V) & \rightarrow R_i(V) \\
\text{CM2}: & R_i(V) \wedge \rho_i(V,V') & \rightarrow R_i(V') \\
\text{CM3}: & (\bigvee_{i \in 1..N \setminus \{j\}} R_i(V) \wedge \rho_i(V,V')) & \rightarrow E_j(V,V') \\
\text{CM4}: & R_i(V) \wedge E_i(V,V') \wedge \rho_i^{=}(V,V') & \rightarrow R_i(V') \\
\text{CM5}: & R_1(V) \wedge \cdots \wedge R_N(V) \wedge \mathit{error}(V) & \rightarrow \mathit{false}
\end{array}
$$

multi-threaded program P is safe

Rybalchenko et al. Synthesizing Software Verifiers from Proof Rules. PLDI'12

$$\left\{ R(\mathsf{g}, \mathsf{p}_{\sigma(1)}, \mathsf{l}_{\sigma(1)}, \ldots, \mathsf{p}_{\sigma(k)}, \mathsf{l}_{\sigma(k)}) \leftarrow \mathit{dist}(\mathsf{p}_1, \ldots, \mathsf{p}_k) \wedge R(\mathsf{g}, \mathsf{p}_1, \mathsf{l}_1, \ldots, \mathsf{p}_k, \mathsf{l}_k) \right\}_{\sigma \in S_k} \quad (6)$$

$$R(\mathsf{g}, \mathsf{p}_1, \mathsf{l}_1, \ldots, \mathsf{p}_k, \mathsf{l}_k) \leftarrow \mathit{dist}(\mathsf{p}_1, \ldots, \mathsf{p}_k) \wedge \mathit{Init}(\mathsf{g}, \mathsf{l}_1) \wedge \cdots \wedge \mathit{Init}(\mathsf{g}, \mathsf{l}_k) \quad (7)$$

$$R(\mathsf{g}', \mathsf{p}_1, \mathsf{l}_1', \ldots, \mathsf{p}_k, \mathsf{l}_k) \leftarrow \mathit{dist}(\mathsf{p}_1, \ldots, \mathsf{p}_k) \wedge ((\mathsf{g}, \mathsf{l}_1) \xrightarrow{\mathsf{p}_1} (\mathsf{g}', \mathsf{l}_1')) \wedge R(\mathsf{g}, \mathsf{p}_1, \mathsf{l}_1, \ldots, \mathsf{p}_k, \mathsf{l}_k) \quad (8)$$

$$R(\mathsf{g}', \mathsf{p}_1, \mathsf{l}_1, \ldots, \mathsf{p}_k, \mathsf{l}_k) \leftarrow \mathit{dist}(\mathsf{p}_0, \mathsf{p}_1, \ldots, \mathsf{p}_k) \wedge ((\mathsf{g}, \mathsf{l}_0) \xrightarrow{\mathsf{p}_0} (\mathsf{g}', \mathsf{l}_0')) \wedge \mathit{RConj}(0, \ldots, k) \quad (9)$$

$$\mathit{false} \leftarrow \mathit{dist}(\mathsf{p}_1, \ldots, \mathsf{p}_r) \wedge \Big(\bigwedge_{j=1,\ldots,m} (\mathsf{p}_j = p_j \wedge (\mathsf{g}, \mathsf{l}_j) \in E_j) \Big) \wedge \mathit{RConj}(1, \ldots, r) \quad (10)$$

Figure 4: Horn constraints encoding a homogeneous infinite system with the help of a k-indexed invariant. S_k is the symmetric group on $\{1, \ldots, k\}$, i.e., the group of all permutations of k numbers; as an optimisation, any generating subset of S_k, for instance transpositions, can be used instead of S_k. In (10), we define $r = \max\{m, k\}$.

Hojjat et al. Horn Clauses for Communicating Timed Systems. HCVS'14

$$\mathit{Init}(i,j,\overline{v}) \wedge \mathit{Init}(j,i,\overline{v}) \wedge \mathit{Init}(i,i,\overline{v}) \wedge \mathit{Init}(j,j,\overline{v}) \Rightarrow I_2(i,j,\overline{v})$$

$$I_2(i,j,\overline{v}) \wedge \mathit{Tr}(i,\overline{v},\overline{v}') \Rightarrow I_2(i,j,\overline{v}') \quad (3)$$

$$I_2(i,j,\overline{v}) \wedge \mathit{Tr}(j,\overline{v},\overline{v}') \Rightarrow I_2(i,j,\overline{v}') \quad (4)$$

$$I_2(i,j,\overline{v}) \wedge I_2(i,k,\overline{v}) \wedge I_2(j,k,\overline{v}) \wedge \mathit{Tr}(k,\overline{v},\overline{v}') \wedge k \neq i \wedge k \neq j \Rightarrow I_2(i,j,\overline{v}') \quad (5)$$

$$I_2(i,j,\overline{v}) \Rightarrow \neg \mathit{Bad}(i,j,\overline{v})$$

Figure 3: $VC_2(T)$ for two-quantifier invariants.

Gurfinkel et al. SMT-Based Verification of Parameterized Systems. FSE 2016

(initial) $\mathsf{init}(g, x_1) \wedge \cdots \wedge \mathsf{init}(g, x_n) \rightarrow \mathit{Inv}(g, \ell_{\mathsf{init}}, x_1, \ldots, \ell_{\mathsf{init}}, x_k)$

(inductive) $\mathit{Inv}(g, \ell_1, x_1, \ldots, \ell_i, x_i, \ldots, \ell_k, x_k) \wedge s(g, x_i, g', x_i') \rightarrow \mathit{Inv}(g', \ell_1, x_1, \ldots, \ell_i', x_i', \ldots, \ell_k,$

(non-interference) $\mathit{Inv}(g, \ell_1, x_1, \ldots, \ell_k, x_k) \wedge$
$\mathit{Inv}(g, \ell^\dagger, x^\dagger, \ell_2, x_2, \ldots, \ell_k, x_k) \wedge$
$\vdots$
$\mathit{Inv}(g, \ell_1, x_1, \ldots, \ell_{k-1}, x_{k-1}, \ell^\dagger, x^\dagger) \wedge s(g, x^\dagger, g', \cdot) \rightarrow \mathit{Inv}(g', \ell_1, x_1, \ldots, \ell_k, x_k)$

(safe) $\mathit{Inv}(g, \ell_1, x_1, \ldots, \ell_k, x_k) \wedge \mathsf{err}(g, \ell_1, x_1, \ldots, \ell_m, x_m) \rightarrow \mathit{false}$

Figure 6. Horn clause encoding for thread modularity at level k (where (ℓ_i, s, ℓ_i') and $(\ell^\dagger, s, \cdot)$ refer to statement s on a
from ℓ_i to ℓ_i' and, respectively, from $\ell^\dagger$ to some other location in the control flow graph)

Hoenicke et al. Thread Modularity at Many Levels. POPL'17

Program Verification with HORN(LIA)

```
z = x; i = 0;
assume (y > 0);
while (i < y) {
  z = z + 1;
  i = i + 1;
}
assert(z == x + y);
```

IS SAT?

```
z = x & i = 0 & y > 0                            → Inv(x, y, z, i)
Inv(x, y, z, i) & i < y & z1=z+1 & i1=i+1        → Inv(x, y, z1, i1)
Inv(x, y, z, i) & i >= y & z != x+y              → false
```

In SMT-LIB

```
(set-logic HORN)

;; Inv(x, y, z, i)
(declare-fun Inv ( Int Int Int Int) Bool)

(assert
 (forall ( (A Int) (B Int) (C Int) (D Int))
        (=> (and (> B 0) (= C A) (= D 0))
           (Inv A B C D)))
 )
(assert
 (forall ( (A Int) (B Int) (C Int) (D Int) (C1 Int) (D1 Int) )
        (=>
         (and (Inv A B C D) (< D B) (= C1 (+ C 1)) (= D1 (+ D
1)))
         (Inv A B C1 D1)
         )
        )
 )
(assert
 (forall ( (A Int) (B Int) (C Int) (D Int))
        (=> (and (Inv A B C D) (>= D B) (not (= C (+ A B))))
           false
           )
        )
 )

(check-sat)
(get-model)
```

```
$ z3 add-by-one.smt2
sat
(model
  (define-fun Inv ((x!0 Int) (x!1 Int) (x!2 Int) (x!3 Int)) Bool
    (and (<= (+ x!2 (* (- 1) x!0) (* (- 1) x!3)) 0)
         (<= (+ x!2 (* (- 1) x!0) (* (- 1) x!1)) 0)
         (<= (+ x!0 x!3 (* (- 1) x!2)) 0)))
)
```

```
Inv(x, y, z, i)

      z  = x + i

      z <= x + y
```

Program Verification with HORN(LIA)

```
int inc(int z) { return z + 1; }
assume(x <= 0);
while (x < 5) {
  x = inc(x);
}
assert(x < 10);
```

IS SAT?

```
r = z + 1                         → Inc(z, r)
x <= 0                            → Inv(x)
Inv(x) & x < 5 & Inc(x, y)        → Inv(y)
Inv(x) & x >= 5 & x >= 10         → false
```

In SMT-LIB

```
(set-logic HORN)
(set-option :fp.xform.inline_linear false)
(set-option :fp.xform.inline_eager false)
(declare-fun Inv ( Int ) Bool)
(declare-fun Inc ( Int Int ) Bool)

(assert (forall ((z Int)) (Inc z (+ z 1))))

(assert (forall ((x Int)) (=> (<= x 0) (Inv x))))

(assert (forall ((x Int) (y Int)) (=> (and (< x 5) (Inc x y))
(Inv y))))

(assert (forall ((x Int)) (=> (and (Inv x) (>= x 5) (>= x 10))
false)))

(check-sat)
(get-model)
```

```
$ z3 add-by-one-fn.smt2
sat
(
  (define-fun Inc ((x!0 Int) (x!1 Int)) Bool
    (not (>= (+ x!1 (* (- 1) x!0)) 2)))
  (define-fun Inv ((x!0 Int)) Bool
    (not (>= x!0 6)))
)
```

```
Inc(x0,x1) :=
      x1 <= x0 + 1
Inv(x0) :=
      x0 <= 5
```

Applications of CHCs

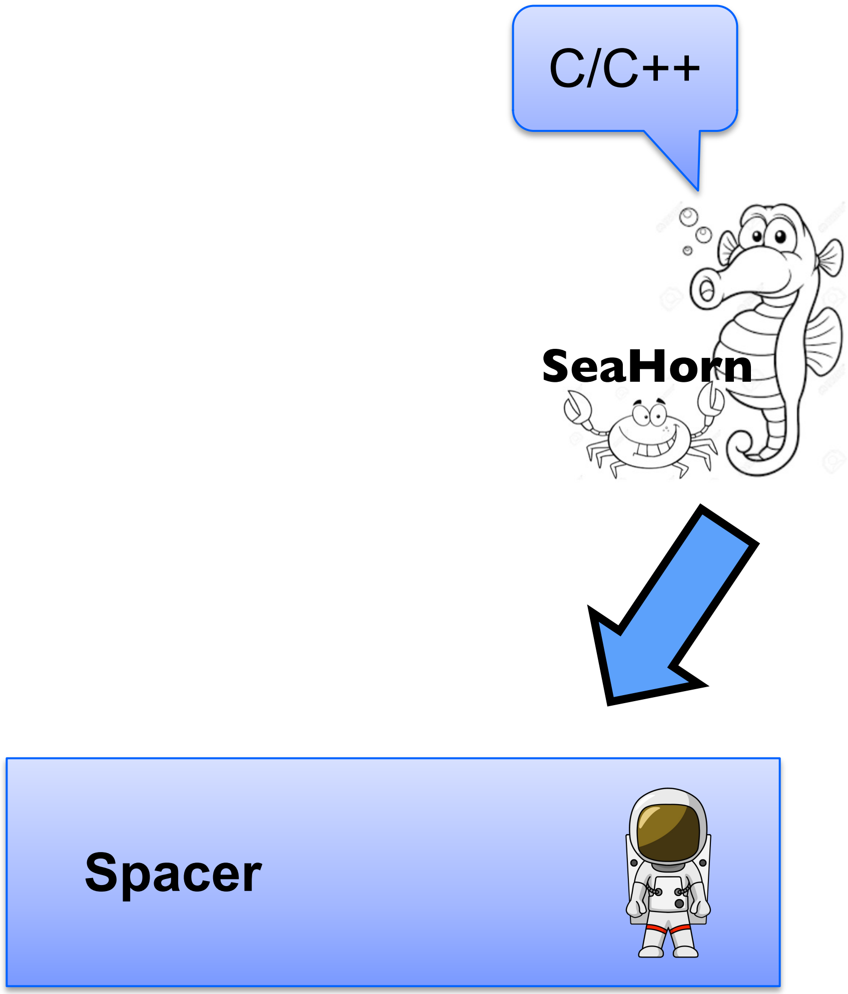
Prototyping different strategies and proof rules for verification

- verification by inductive invariants
- modular invariants
- predicate abstraction
- modular proof rules for concurrent systems
- verification of parameterized systems
- type inference for refinement type systems
- synthesis
- …
- create new verification tools by reducing to CHCs

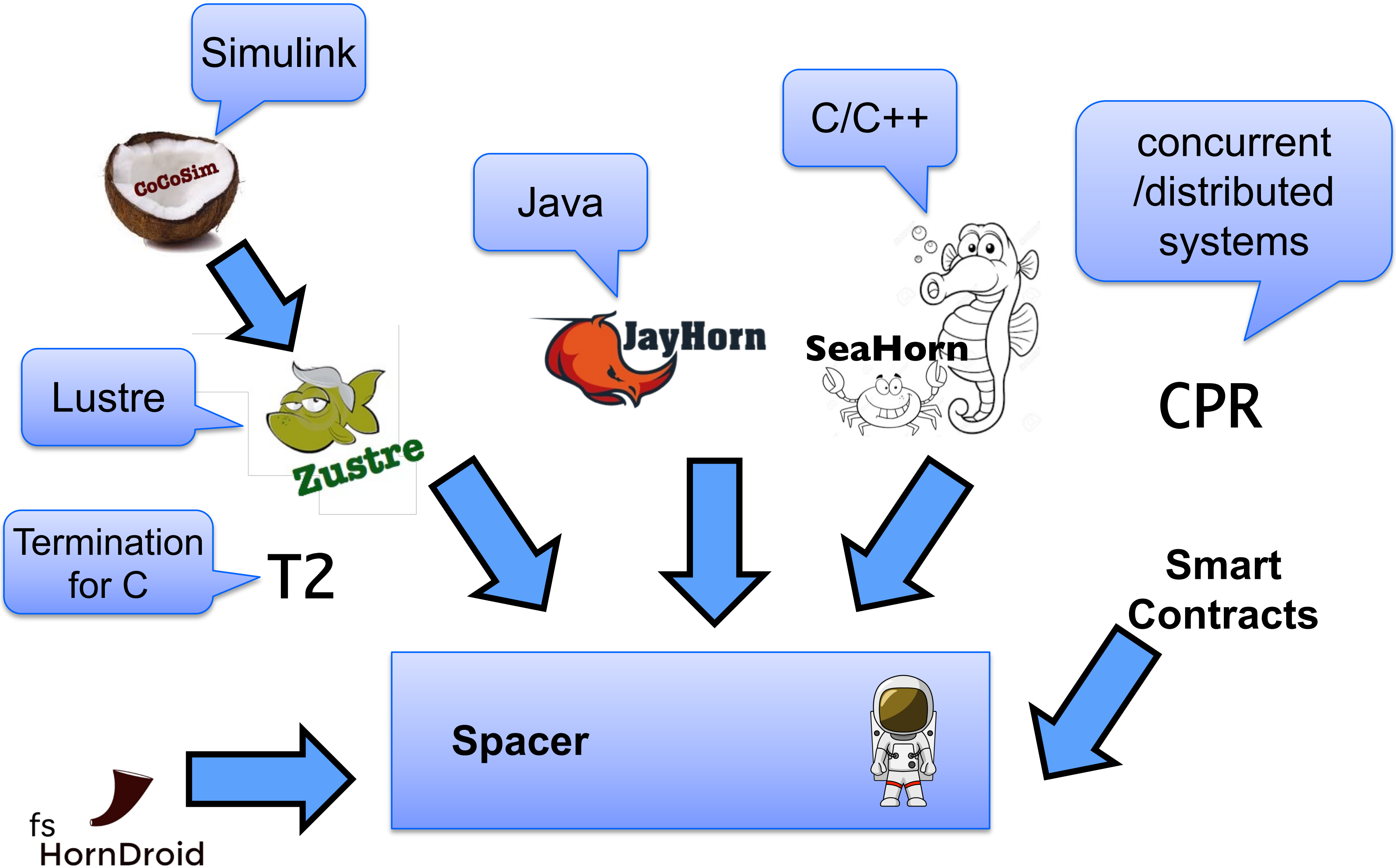
Building automated verification tools

- SeaHorn, JayHorn, RustHorn, …
- SmartACE, SolCMC, …

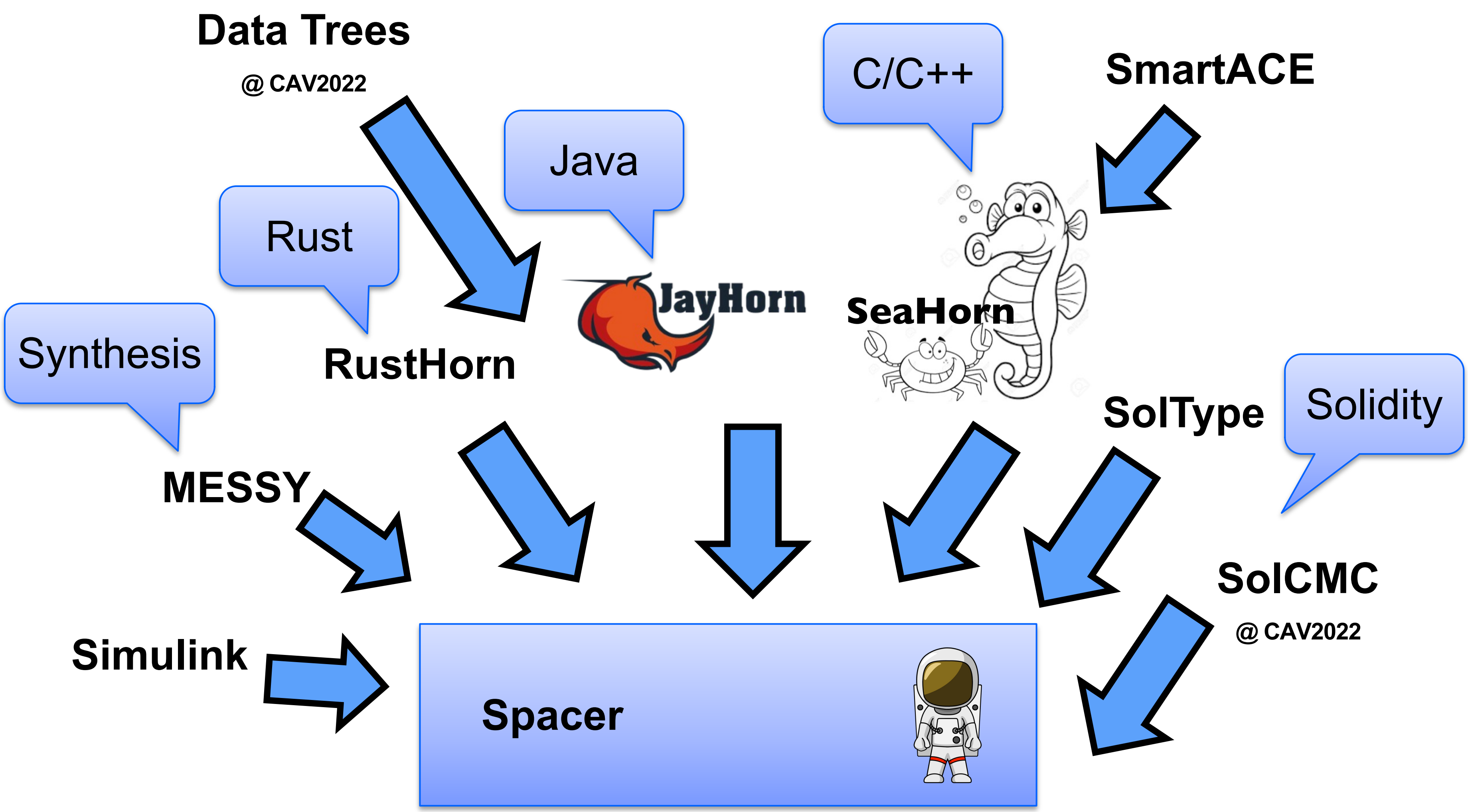
Logic-based Algorithmic Verification



Logic-based Algorithmic Verification



Logic-based Algorithmic Verification (in 2022)



Current State of CHC Solving

Multiple mature solvers using competing techniques and algorithms

- Spacer (in Z3), Eldarica, FreqHorn, Golem, …

Annual competition

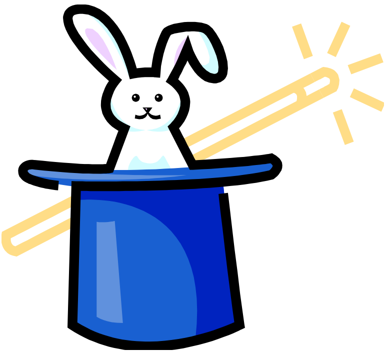
- CHC-COMP: https://chc-comp.github.io/
- in 2022, 7 tracks with 5+1 solvers

Growing collection of benchmarks

- maintained by CHC-COMP
- established (simplified) format
- organized in separate repos under https://github.com/chc-comp

Growing number of academic and industrial users

- SeaHorn, JayHorn, RustHorn, MESSY, SolType, SolC SMTChecker, …



SOLVING CONSTRAINED HORN CLAUSES

UNIVERSITY OF WATERLOO

A little bit of complexity

Satisfiability of CHC over most interesting theories is **undecidable**

- e.g., CHC(Linear Real Arithmetic), CHC(Linear Integer Arithmetic)
- proof: many easy reductions, for example, counter automata

Satisfiability of Linear CHC over Propositional logic is **decidable**

- **Finite state model checking** of transition systems
- Complexity: linear in the size of the graph induced by the transition system

Satisfiability of Non-Linear CHC over Propositional logic is **decidable**

- **Finite state** model checking of **pushdown systems**
- Complexity: cubic in the size of the pushdown system

Decidability of some classes of CHC: Difference arithmetic (= timed automata)

Procedures for Solving CHC(T)

Predicate abstraction by lifting Model Checking to HORN

- QARMC, Eldarica, …

Maximal Inductive Subset from a finite Candidate space (Houdini)

- TACAS'18: hoice, FreqHorn

Machine Learning

- PLDI'18: sample, ML to guess predicates, DT to guess combinations

Abstract Interpretation (Poly, intervals, boxes, arrays…)

- Approximate least model by an abstract domain (SeaHorn, …)

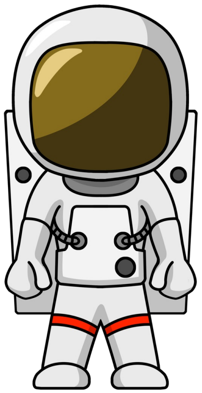
Interpolation-based Model Checking

- Duality, QARMC, …

SMT-based Unbounded Model Checking (building on IC3/PDR)

- **SPACER**, Implicit Predicate Abstraction

Spacer: Solving SMT-constrained CHC



Spacer: SAT procedure for SMT-constrained Horn Clauses

- now the default 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
- Good support for many other SMT-theories
  - bit-vectors, ADT, recursive functions, …

Supports Non-Linear CHC

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

A Magician's Guide to Solving Undecidable Problems

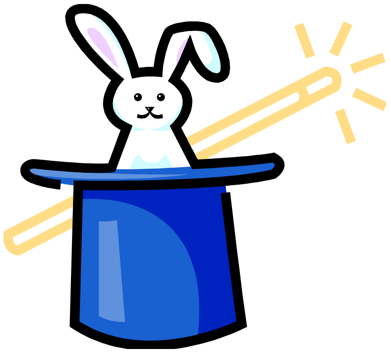
Develop a procedure ***P*** for a decidable problem

Show that ***P*** is a decision procedure for the problem

- e.g., model checking of finite-state systems

Choose one of

- Always terminate with some answer (over-approximation)
- Always make useful progress (under-approximation)



Extend procedure ***P*** to procedure ***Q*** that "solves" the undecidable problem

- Ensure that ***Q*** is still a decision procedure whenever ***P*** is
- Ensure that ***Q*** either always terminates or makes progress

SPACER's guiding principles for solving CHCs

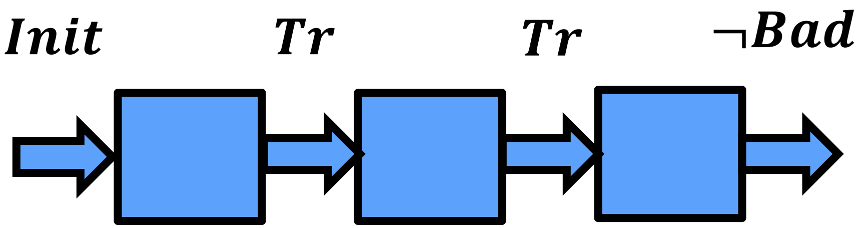
Make Progress

- always make progress
- if input CHC is unsatisfiable, after enough time, the solving procedure must terminate with UNSAT
- e.g., examine longer and longer resolution proofs (i.e., unfoldings)

Keep Decidability

- decision procedure for decidable fragments
- usually, we ensure that solving procedures are decision procedures for CHC over Propositional logic (i.e., finite state model checking)
- ”sharpen” decidability result based on specific domain (i.e., LIA, ADT, etc.)
- many open decidability questions remain
  - e.g., is Spacer a decision procedure for (encoding) of timed automata?

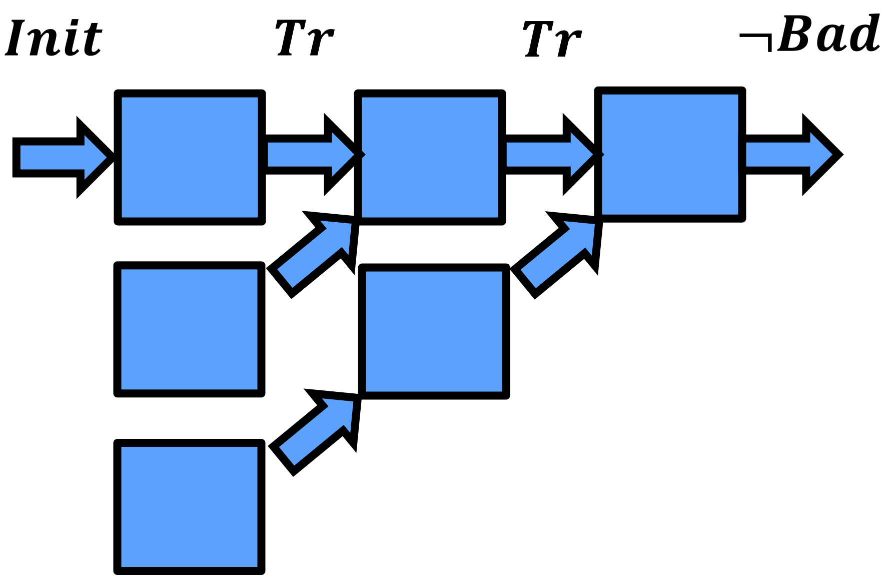
IC3, PDR, and friends



Finite State Machines

(HW model checking)

[Bradley, VMCAI 2011]

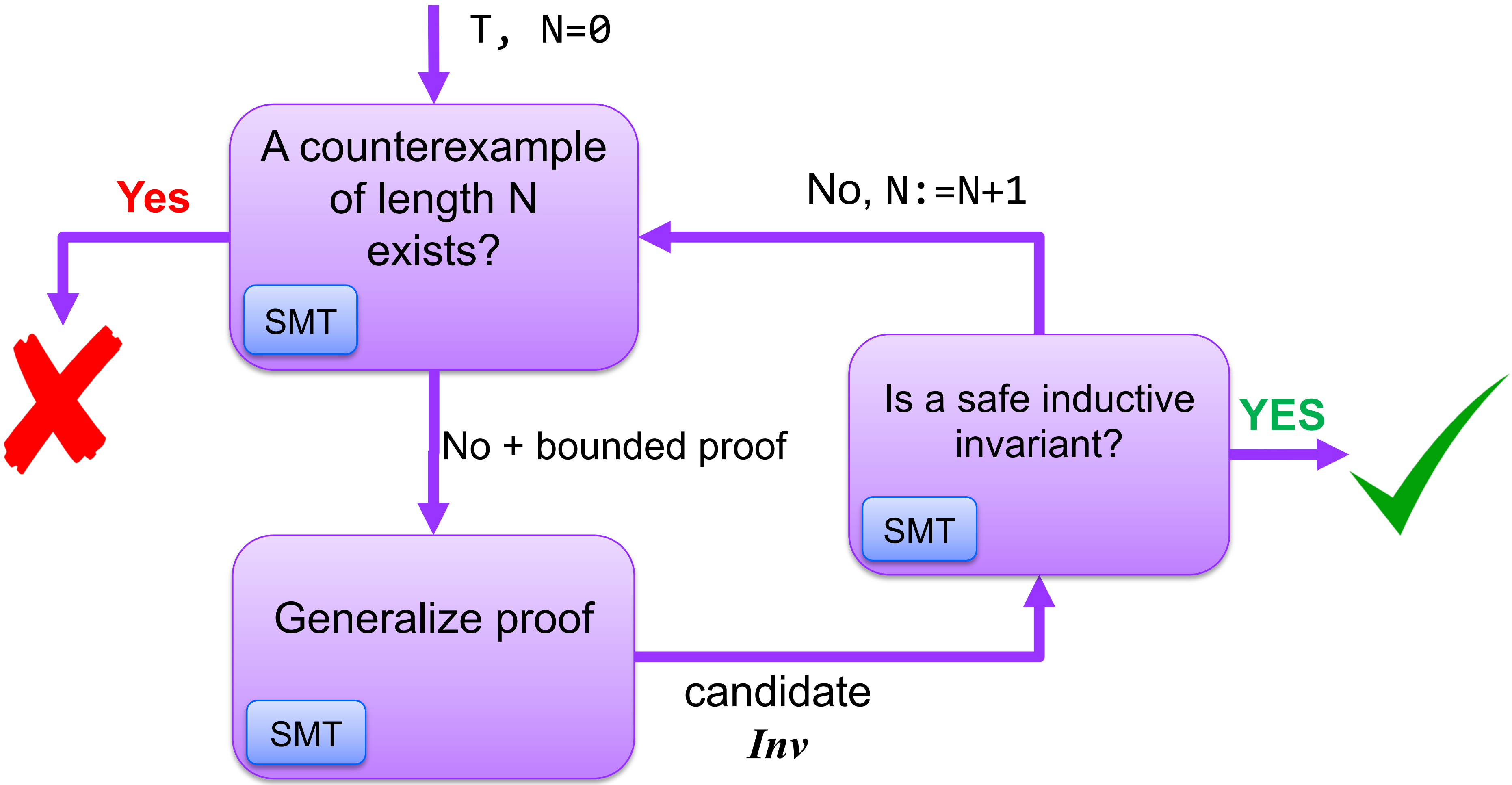


Push Down Machines

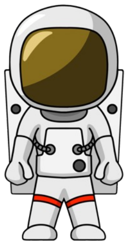
(SW model checking)

[Hoder&Bjørner, SAT 2012]

Verification by Incremental Generalization



SPACER



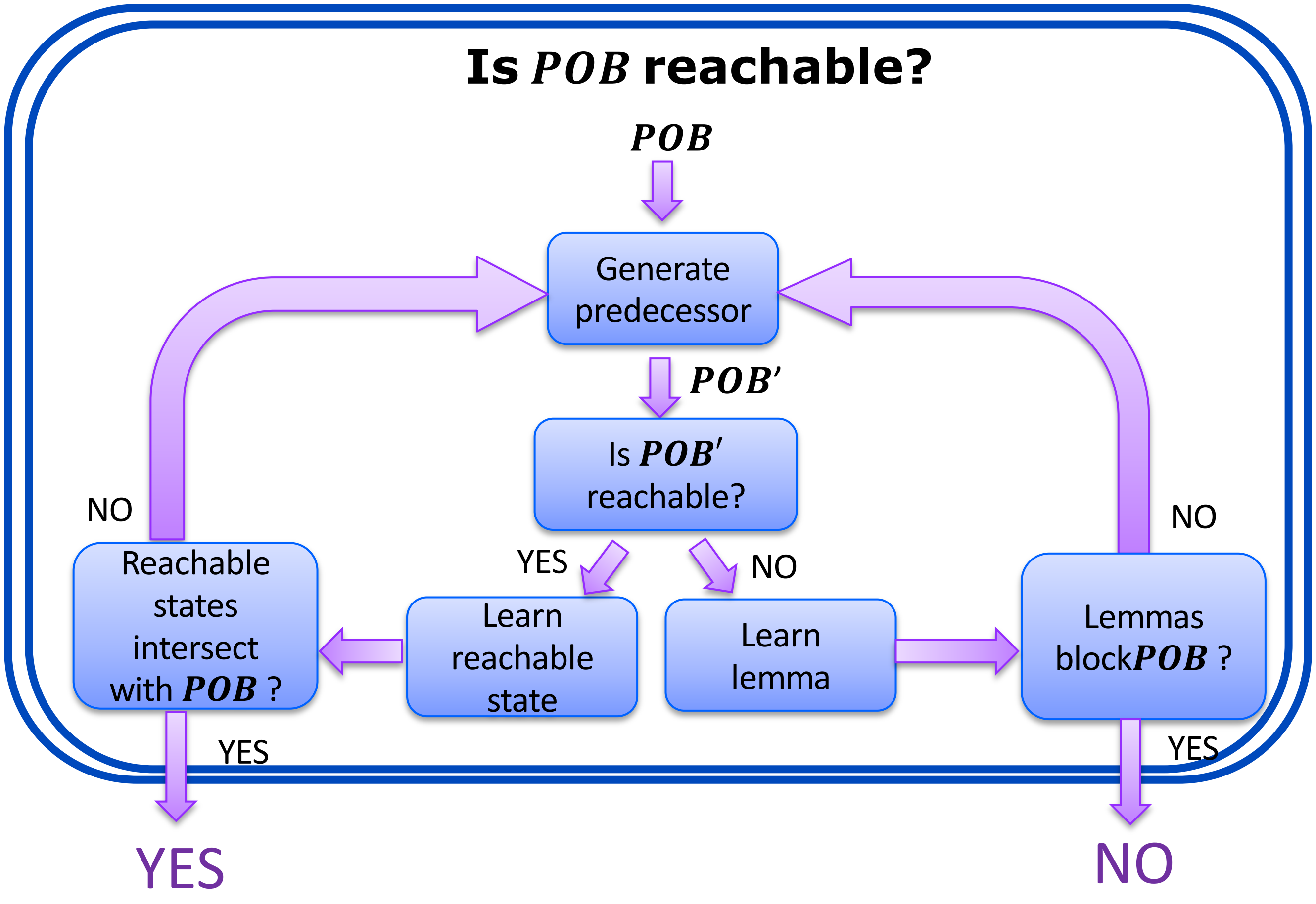
IC3-style search for solutions to CHCs

Works by recursively *blocking* ***proof obligations*** (POB)

POB

- BAD states
- Predecessors to BAD states

Generate predecessors using quantifier elimination (Model Based Projection)



Linear CHC Satisfiability

Satisfiability of a set of linear CHCs is reducible to satisfiability of **THREE** clauses of the form

$$Init(X) \rightarrow P(X)$$
$$P(X) \land Tr(X, X') \rightarrow P(X')$$
$$P(X) \rightarrow \neg Bad(X)$$

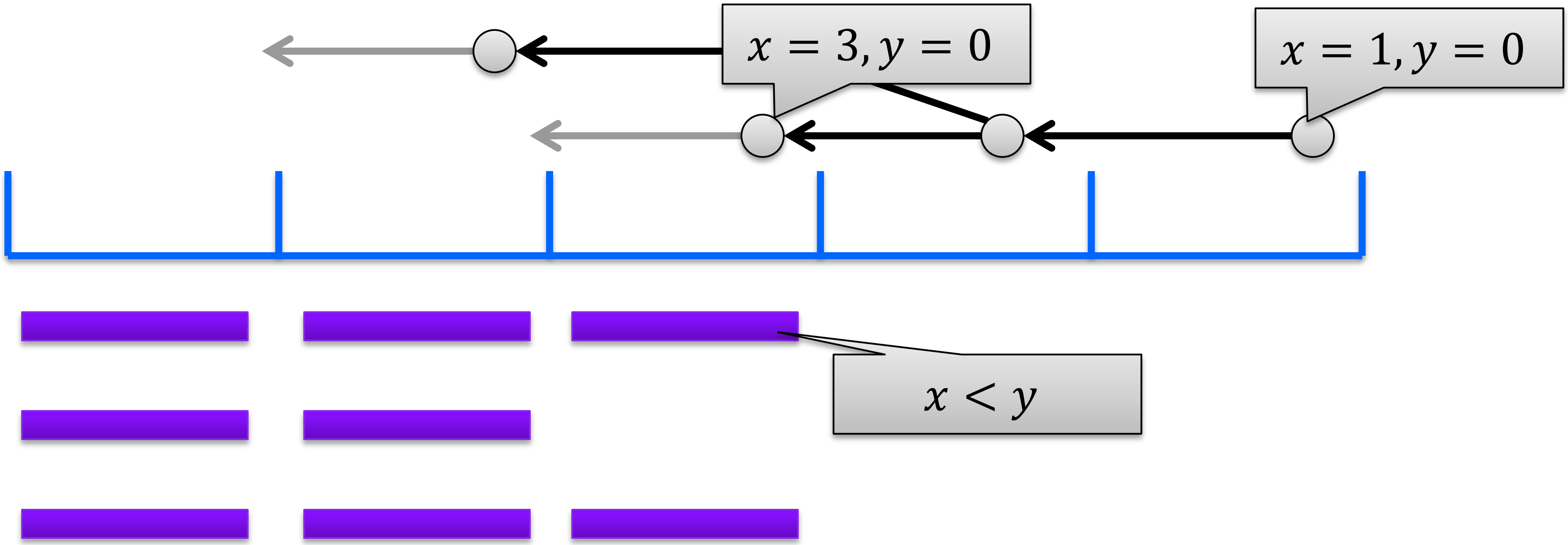
where, X' = {x' | x in X}, P a fresh predicate, and *Init*, *Bad*, and *Tr* are constraints

Proof:

add extra arguments to distinguish between predicates

$$\frac{Q(y) \land tau \rightarrow W(y, z)}{P(id=\text{'Q'}, y) \land tau \rightarrow P(id=\text{'W'}, y, z)}$$

IC3/PDR In Pictures: Search for Finite Cexs



Predecessor

$$\text{find } M \text{ s.t. } M \models F_i \wedge Tr \wedge m'$$

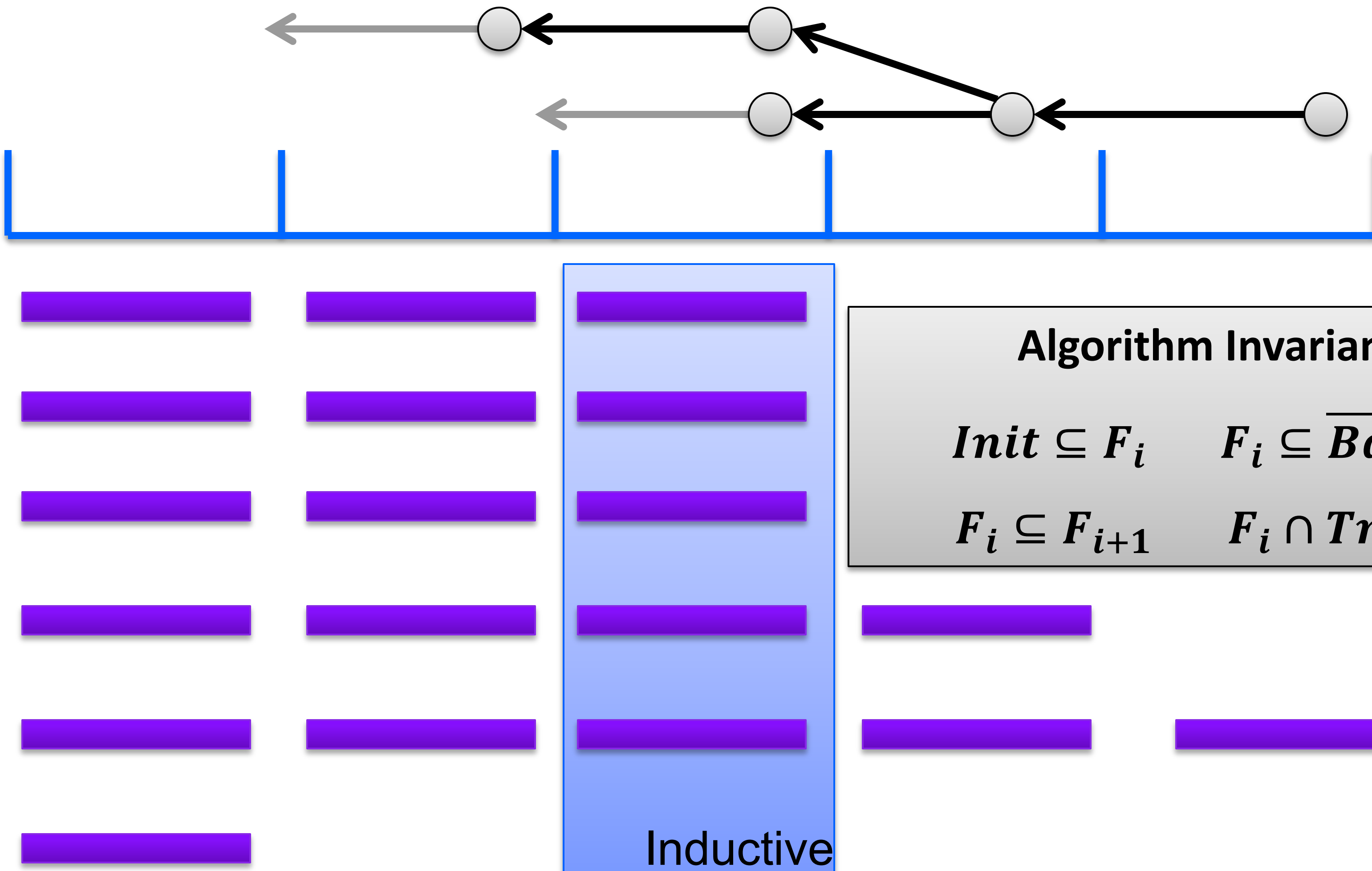
$$\text{find } m \text{ s.t. } (M \models m) \wedge (m \implies \exists V' \cdot Tr \wedge m')$$

NewLemma

$$\text{find } \ell \text{ s.t. } (F_i \wedge Tr \implies \ell') \wedge (\ell \implies \neg m)$$

UNIVERSITY OF WATERLOO

IC3/PDR in Pictures: Is Inductive



Algorithm Invariants

$Init \subseteq F_i$ $\quad F_i \subseteq \overline{Bad}$

$F_i \subseteq F_{i+1}$ $\quad F_i \cap Tr_i \subseteq F'_{i+1}$

SMT-query: $\vdash \ell \wedge F_i \wedge Tr \implies \ell'$

IC3/PDR: Solving Linear (Propositional) CHC

Unreachable and Reachable

- terminate the algorithm when a solution is found

Unfold

- increase search bound by 1

Candidate

- choose a bad state in the last frame

Predecessor

- extend a pob (backward) consistent with the current frame
- choose an assignment ***s*** s.t. $(s \wedge Fi \wedge Tr \wedge pob')$ is SAT

NewLemma

- construct a lemma to explain why pob cannot be extended
- Find a clause ***L*** s.t. $L \Rightarrow \neg pob$, $Init \Rightarrow L$, and $F_i \wedge Tr \Rightarrow L'$

Induction

- propagate a lemma as far into the future as possible
- (optionally) strengthen by dropping literals

UNIVERSITY OF WATERLOO

Extending IC3/PDR to CHC Solving

Theories with infinitely many models

- infinitely many satisfying assignments
- **can't** simply **enumerate** (when computing predecessor)
- **can't** block **one** assignment **at a time** (when blocking)

Non-Linear Horn Clauses

- multiple interdependent predecessors
- when a CHC clause depends on multiple predicates

CHC solving is undecidable in general

- want an algorithm that makes **progress**
- **doesn't** get **stuck** in a decidable sub-problem
- **guaranteed** to find a **counterexample** (if it exists)

IC3/PDR: Solving Linear (Propositional) CHC

Unreachable and Reachable

- terminate the algorithm when a solution is found

Unfold

- increase search bound by 1

Candidate

- choose a bad state in the last frame

Theory dependent

Predecessor

- extend a pob (backward) consistent with the current frame
- choose an assignment ***s*** s.t. $(s \wedge Fi \wedge Tr \wedge pob')$ is SAT

NewLemma

- construct a lemma to explain why pob cannot be extended
- Find a clause ***L*** s.t. $L \Rightarrow \neg pob$, $Init \Rightarrow L$, and $F_i \wedge Tr \Rightarrow L'$

Induction

- propagate a lemma as far into the future as possible
- (optionally) strengthen by dropping literals

UNIVERSITY OF WATERLOO

$$\left((\boldsymbol{F_i} \wedge \boldsymbol{Tr}) \vee \boldsymbol{Init'}\right) \Rightarrow \boldsymbol{\varphi'}, \qquad \boldsymbol{\varphi'} \Rightarrow \neg \boldsymbol{pob'}$$

Looking for φ’

NEW LEMMA (SPACER)

UNIVERSITY OF WATERLOO

Spacer: NewLemma Rule

Notation: $\mathcal{F}(A) = (A(X) \wedge \mathit{Tr}) \vee \mathit{Init}(X')$.

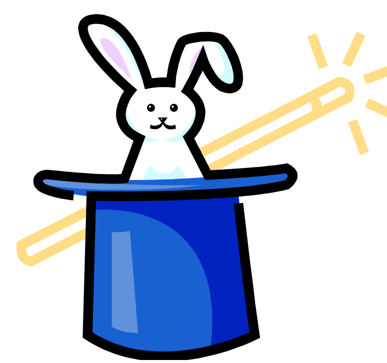
NewLemma For $0 \le i < N$, given a proof obligation $\langle P, i+1 \rangle \in Q$ s.t. $\mathcal{F}(F_i) \wedge P'$ is unsatisfiable, add $P^{\uparrow} = \textsc{Itp}(\mathcal{F}(F_i), P')$ to F_j for $j \le i+1$.

Proof obligation (pob) is blocked using Craig Interpolation

- summarizes the reason why the pob cannot be extended

Generalization is not inductive

- weaker than IC3/PDR
- *inductive generalization for arithmetic is still an open problem*



UNIVERSITY OF WATERLOO

Interpolation in Spacer

Much simpler than general interpolation problem for $A \wedge B$

- B is always a conjunction of literals (B is the pob)
- A is dynamically split into DNF by the SMT solver (A is the constraint)
- the signature of B is shared with the signature of A

Interpolation algorithm is reduced to analyzing all theory lemmas in a proof produced by the SMT solver

- every theory-lemma that mixes B-pure literals with other literals is interpolated to produce a single literal in the final solution
- interpolation is restricted to clauses of the form $(\bigwedge B_i \Rightarrow \bigvee A_j)$

Interpolating (UNSAT) Cores

- improve interpolation algorithms and definitions to the specific case of Spacer
- classical interpolation focuses on eliminating non-shared symbols
- in Spacer, the focus is on finding good generalizations

$$\boldsymbol{s \subseteq pre(pob)}$$

$$\equiv$$

$$\boldsymbol{s \Rightarrow \exists X'.\, Tr(X, X') \wedge pob(X')}$$

Computing a predecessor ***s*** of a proof obligation ***pob***

PREDECESSOR

Model Based Projection

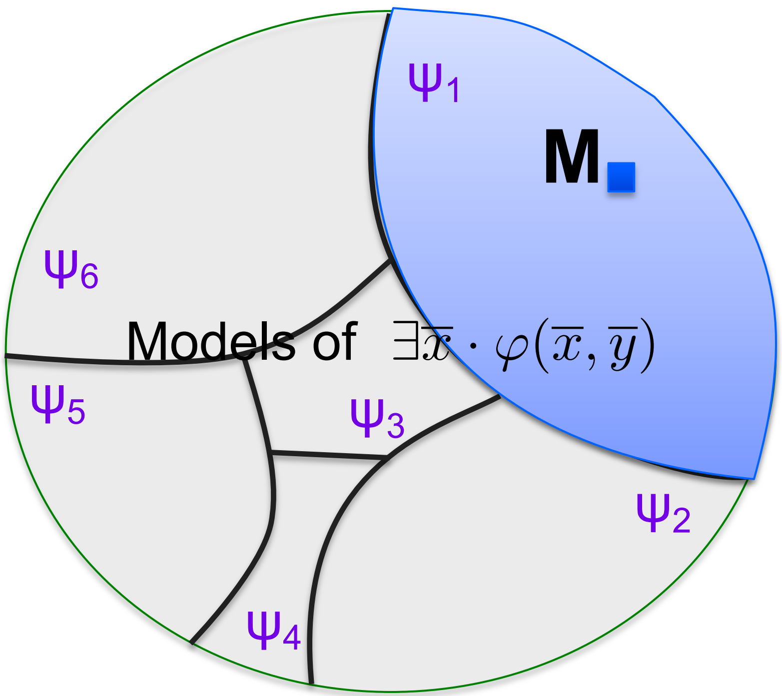
Definition: Let φ be a formula, X a set of variables, and M a model of φ. Then $\psi = MBP\,(X, M, \varphi)$ is a Model Based Projection of X, M, φ iff

1. ψ *is a conjunction of literals*
2. $Vars(\psi) \subseteq Vars(\varphi) \setminus X$
3. $M \vDash \psi$
4. $\psi \Rightarrow \exists X\,.\,\varphi$

Model Based Projection **under-approximates** existential quantifier elimination relative to a given model (i.e., satisfying assignment)

Model Based Projection

Expensive to find a quantifier-free $\psi(\overline{y}) \equiv \exists \overline{x} \cdot \varphi(\overline{x}, \overline{y})$



1. Find model M of φ (x,y)
2. Compute a disjunct of ∃x.φ containing M

$$\exists x \cdot \varphi \equiv \psi_1 \vee \psi_2 \vee \psi_3 \vee \psi_4 \vee \psi_5 \vee \psi_6$$

UNIVERSITY OF WATERLOO

Fourier–Motzkin Quantifier Elimination for LRA

$$\exists x \cdot \bigwedge_i s_i < x \wedge \bigwedge_j x < t_j$$

$$= \quad \bigwedge_i \bigwedge_j resolve(s_i < x, x < t_j, x)$$

$$= \quad \bigwedge_i \bigwedge_j s_i < t_j$$

Quadratic increase in the formula size per each eliminated variable

Fourier-Motzkin by Example

$$\exists x \cdot \quad s_0 < x \wedge s_1 < x \wedge s_2 < x \wedge$$
$$x < t_0 \wedge x < t_1 \wedge x < t_2$$

$$\equiv$$

$$s_0 < t_0 \wedge s_1 < t_0 \wedge s_2 < t_0$$
$$s_0 < t_1 \wedge s_1 < t_1 \wedge s_2 < t_1$$
$$s_0 < t_2 \wedge s_1 < t_2 \wedge s_2 < t_2$$

Quantifier Elimination with Assumptions

$$
\begin{aligned}
& \left(\bigwedge_{j \neq 0} t_0 \leq t_j\right) \wedge \exists x \cdot \bigwedge_i s_i < x \wedge \bigwedge_j x < t_j \\
= \quad & \left(\bigwedge_{j \neq 0} t_0 \leq t_j\right) \wedge \bigwedge_i \mathit{resolve}(s_i < x, x < t_0, x) \\
= \quad & \left(\bigwedge_{j \neq 0} t_0 \leq t_j\right) \wedge \bigwedge_i s_i < t_0
\end{aligned}
$$

Quantifier elimination is simplified by a choice of a minimal upper bound

- For each choice of minimal upper bound, no increase in term size
- Dually, can use largest lower bound

How to chose the assumptions?!

- MBP == use the order chosen by the model

MBP Example

$$\exists x \cdot \quad s_0 < x \wedge s_1 < x \wedge s_2 < x \wedge$$
$$x < t_0 \wedge x < t_1 \wedge x < t_2$$

$$\Leftarrow$$

an assumption

$$(t_0 < t_1 \wedge t_0 < t_2)$$
$$\wedge$$
$$(s_0 < t_0 \wedge s_1 < t_0 \wedge s_2 < t_0)$$

qelim under the assumption

MBP for Linear Rational Arithmetic

Input: a formula F, variable x, a model M of F

Use the model M to pick the right assumption to eliminate x

$$Mbp_x(M, x = s \wedge L) = L[x \leftarrow s]$$

$$Mbp_x(M, x \neq s \wedge L) = Mbp_x(M, s < x \wedge L) \text{ if } M(x) > M(s)$$

$$Mbp_x(M, x \neq s \wedge L) = Mbp_x(M, -s < -x \wedge L) \text{ if } M(x) < M(s)$$

$$Mbp_x(M, \bigwedge_i s_i < x \wedge \bigwedge_j x < t_j) = \bigwedge_i s_i < t_0 \wedge \bigwedge_j t_0 \leq t_j \text{ where } M(t_0) \leq M(t_i), \forall i$$

N. Bjørner, M. Janota: Playing with Quantified Satisfaction. LPAR (short papers) 2015

Spacer: Predecessor Rule

Notation: $\mathcal{F}(A) = (A(X) \wedge Tr) \vee Init(X')$.

Predecessor If $\langle P, i+1 \rangle \in Q$ and there is a model $m(X, X')$ s.t.
$m \models \mathcal{F}(F_i) \wedge P'$, add $\langle P_{\downarrow}, i \rangle$ to Q, where $P_{\downarrow} = MBP(X', m, \mathcal{F}(F_i) \wedge P')$.

Compute a predecessor pob using Model Based Projection

To ensure progress, **Predecessor** must be finite

- finitely many possible pob predecessors when all other arguments are fixed

Spacer: Solving CHC(LRA)

Unreachable and Reachable

- terminate the algorithm when a solution is found

Unfold

- increase search bound by 1

Candidate

- choose a bad state in the last frame

Predecessor

- extend a pob (backward) consistent with the current frame
- find a model **M** of ***s*** s.t. ($F_i \land Tr \land pob'$), and let **s** = MBP(X', $F_i \land Tr \land pob'$)

NewLemma

- construct a lemma to explain why pob cannot be extended
- Find an interpolant ***L*** s.t. $L \Rightarrow \neg pob$, $Init \Rightarrow L$, and $F_i \land Tr \Rightarrow L'$

Induction

- propagate a lemma as far into the future as possible
- (optionally) strengthen by dropping literals

Non-Linear CHC Satisfiability

Satisfiability of a set of arbitrary (i.e., linear or non-linear) CHCs is reducible to satisfiability of THREE (3) clauses of the form

$$Init(X) \to P(X)$$

$$P(X) \wedge P(X^o) \wedge Tr(X, X^o, X') \to P(X')$$

$$P(X) \to \neg Bad(X)$$

where, $X' = \{x' \mid x \text{ in } X\}$, $X^o = \{x^o \mid x \text{ in } X\}$, P a fresh predicate, and Init, Bad, and Tr are constraints

Multiple Predecessor POBs

$$P(x) \land P(y) \land x > y \land z = x + y \implies P(z)$$

How to compute a predecessor for a proof obligation z > 0

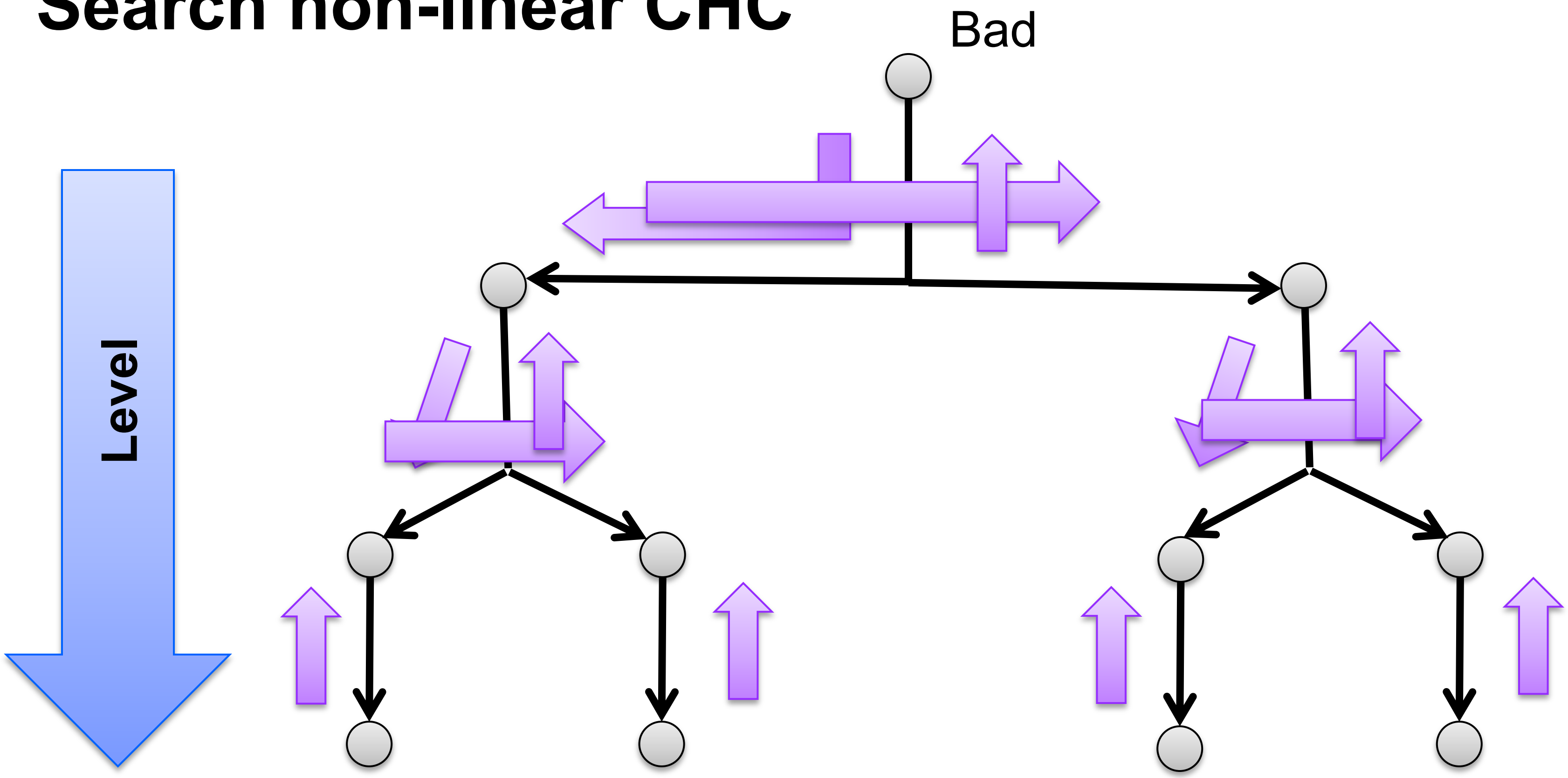
Predecessor over the constraint is:

$$\begin{aligned} & \exists z \cdot x > y \land z = x + y \land z > 0 \\ = \quad & x > y \land x + y > 0 \end{aligned}$$

Need to create **two** different proof obligations

- one for P(x) and one for P(y)
- project y using MBP for P(x)
- project x using MBP for P(y)

Search non-linear CHC



In **Predecessor**, unfold the derivation tree in a fixed depth-first order

- use MBP to create new pobs

Successor: Learn new facts (reachable states) on the way up

- use MBP to propagate facts bottom up

Spacer

Cache Reachable states

Pob queue as in IC3/PDR

Successor and two Predecessor rules use MBP

NewLemma rule uses ITP

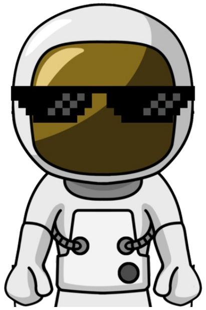
Input: A safety problem $\langle Init(X), Tr(X, X^o, X'), Bad(X)\rangle$.
Output: *Unreachable* or *Reachable*
Data: A cex queue Q, where a cex $c \in Q$ is a pair $\langle m, i\rangle$, m is a cube over state variables, and $i \in \mathbb{N}$. A level N. A set of reachable states REACH. A trace $F_0, F_1, \ldots$
Notation: $\mathcal{F}(A, B) = Init(X') \vee (A(X) \wedge B(X^o) \wedge Tr)$, and $\mathcal{F}(A) = \mathcal{F}(A, A)$
Initially: $Q = \emptyset$, $N = 0$, $F_0 = Init$, $\forall i > 0 \cdot F_i = \emptyset$, REACH $= Init$
Require: $Init \to \neg Bad$
repeat

- **Unreachable** If there is an $i < N$ s.t. $F_i \subseteq F_{i+1}$ **return** *Unreachable*.
- **Reachable** If REACH $\wedge$ *Bad* is satisfiable, **return** *Reachable*.
- **Unfold** If $F_N \to \neg Bad$, then set $N \leftarrow N + 1$ and $Q \leftarrow \emptyset$.
- **Candidate** If for some m, $m \to F_N \wedge Bad$, then add $\langle m, N\rangle$ to Q.
- **Successor** If there is $\langle m, i+1\rangle \in Q$ and a model M s.t. $M \models \psi$, where $\psi = \mathcal{F}(\vee \text{REACH}) \wedge m'$. Then, add s to REACH, where $s' \in \text{MBP}(\{X, X^o\}, \psi)$.
- **MustPredecessor** If there is $\langle m, i+1\rangle \in Q$, and a model M s.t. $M \models \psi$, where $\psi = \mathcal{F}(F_i, \vee \text{REACH}) \wedge m'$. Then, add s to Q, where $s \in \text{MBP}(\{X^o, X'\}, \psi)$.
- **MayPredecessor** If there is $\langle m, i+1\rangle \in Q$ and a model M s.t. $M \models \psi$, where $\psi = \mathcal{F}(F_i) \wedge m'$. Then, add s to Q, where $s^o \in \text{MBP}(\{X, X'\}, \psi)$.
- **NewLemma** If there is an $\langle m, i+1\rangle \in Q$, s.t. $\mathcal{F}(F_i) \wedge m'$ is unsatisfiable. Then, add $\varphi = \text{ITP}(\mathcal{F}(F_i), m')$ to F_j, for all $0 \le j \le i+1$.
- **ReQueue** If $\langle m, i\rangle \in Q$, $0 < i < N$ and $\mathcal{F}(F_{i-1}) \wedge m'$ is unsatisfiable, then add $\langle m, i+1\rangle$ to Q.
- **Push** For $0 \le i < N$ and a clause $(\varphi \vee \psi) \in F_i$, if $\varphi \notin F_{i+1}$, $\mathcal{F}(\varphi \wedge F_i) \to \varphi'$, then add φ to F_j, for all $j \le i+1$.

until ∞;

A. Komuravelli, G., S. Chaki: SMT-Based Model Checking for Recursive Programs. CAV 2014

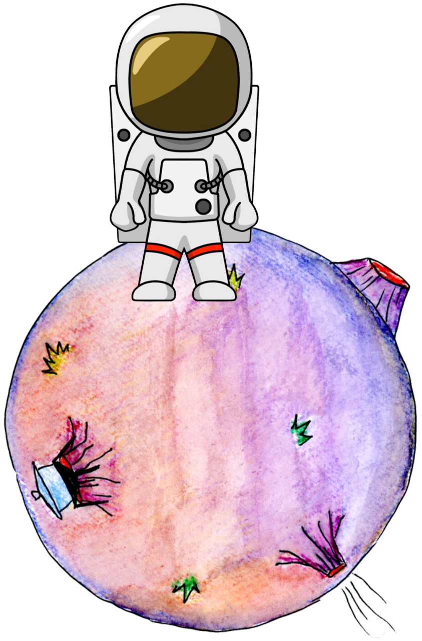
The Curse of Interpolantion



Hari Govind V. K., J. Chen, S. Shoham, G.; Global Guidance for Local Generalization in Model Checking. CAV 2020

Spacer Tom ONLY knows how to do

Local **reasoning**



Generalizing from single predecessors results in **limited** exploration **horizon**

Generalization typically relies on **interpolation**

Interpolation can work wonders!

- e.g., generate breakthrough terms like equality: a = b

Ground Control to Spacer Tom:

We've got a PROBLEM!

Not aware of the structure of the inductive proof so far

Interpolant is very much dependent on heuristics in the underlying SMT engine

$a + b < 4$ is just as likely as $a = b$

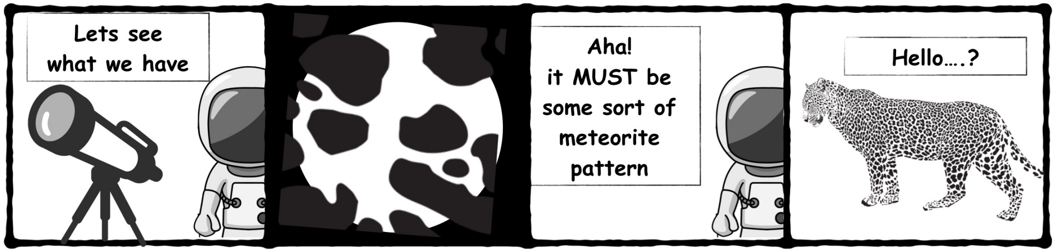
Crucial in infinite-state systems than in finite-state systems

there are usually infinitely many generalizations to choose from

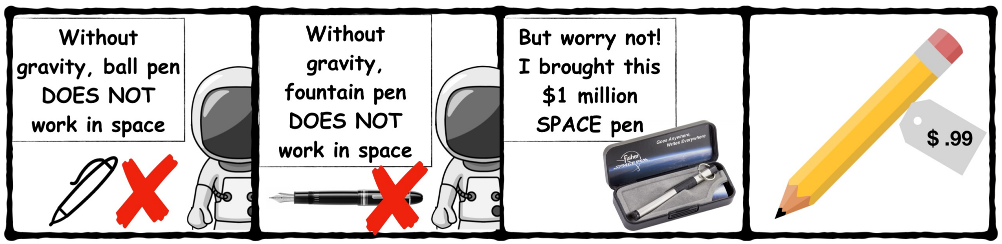
Spacer Tom can be MISSGUIDED!

As illustrated by

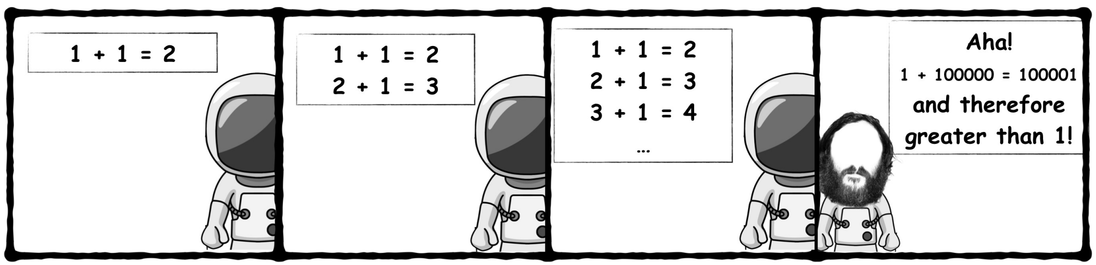
Myopic generalization



Excessive generalization



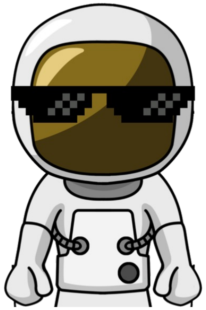
Getting stuck in a rut



Spacer Tom can be MISSGUIDED!

Myopic Generalization

```
a, c = 0;
b, d = 0;
while (nd()) {
inv: (a - c = b - d)
 if (nd()) {a++; b++;}
 else      {c++; d++;}
}
assert (a < c ⇒ b < d);
```



nd() returns a non-deterministic Boolean value.

```
a, c = 0;
b, d = 0;
while (nd()) {
inv: (a - c = b - d)
 if (nd()) {a++; b++;}
 else      {c++; d++;}
}
assert (a < c ⇒ b < d);
```

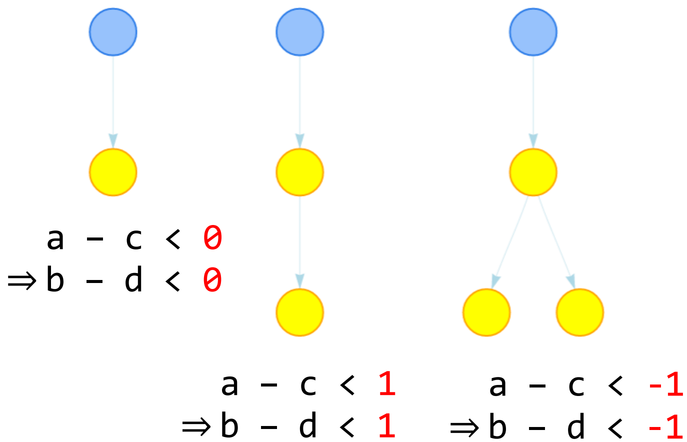
$$a - c < 0 \Rightarrow b - d < 0$$

```
a, c = 0;
b, d = 0;
while (nd()) {
inv: (a - c = b - d)
 if (nd()) {a++; b++;}
 else      {c++; d++;}
}
assert (a < c ⇒ b < d);
```

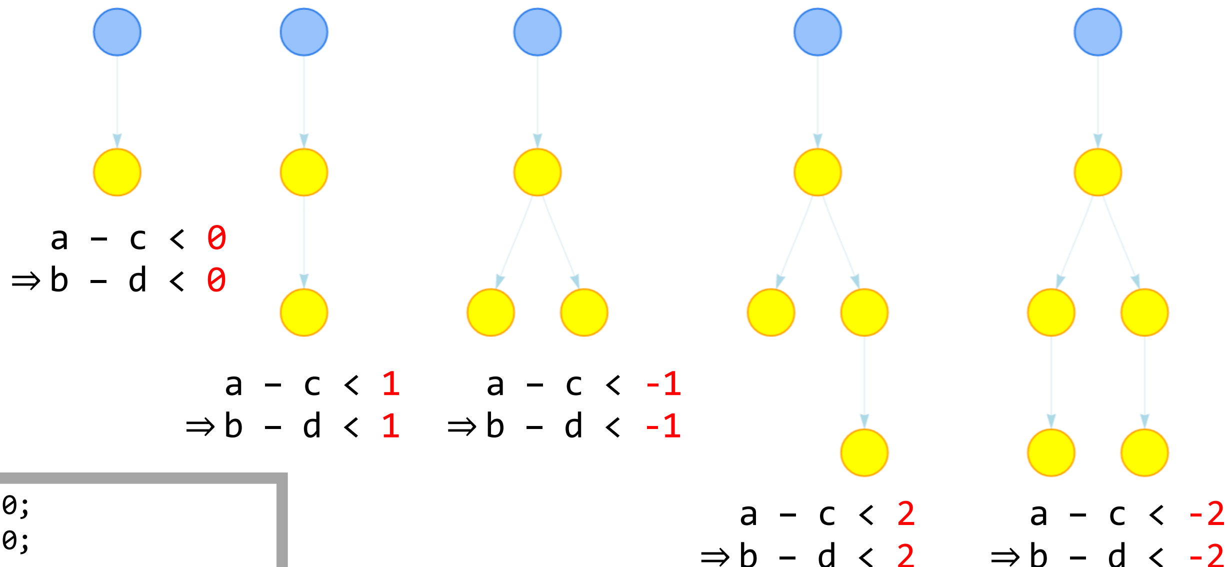
$$a - c < 0 \Rightarrow b - d < 0$$

$$a < c \Rightarrow b < d$$

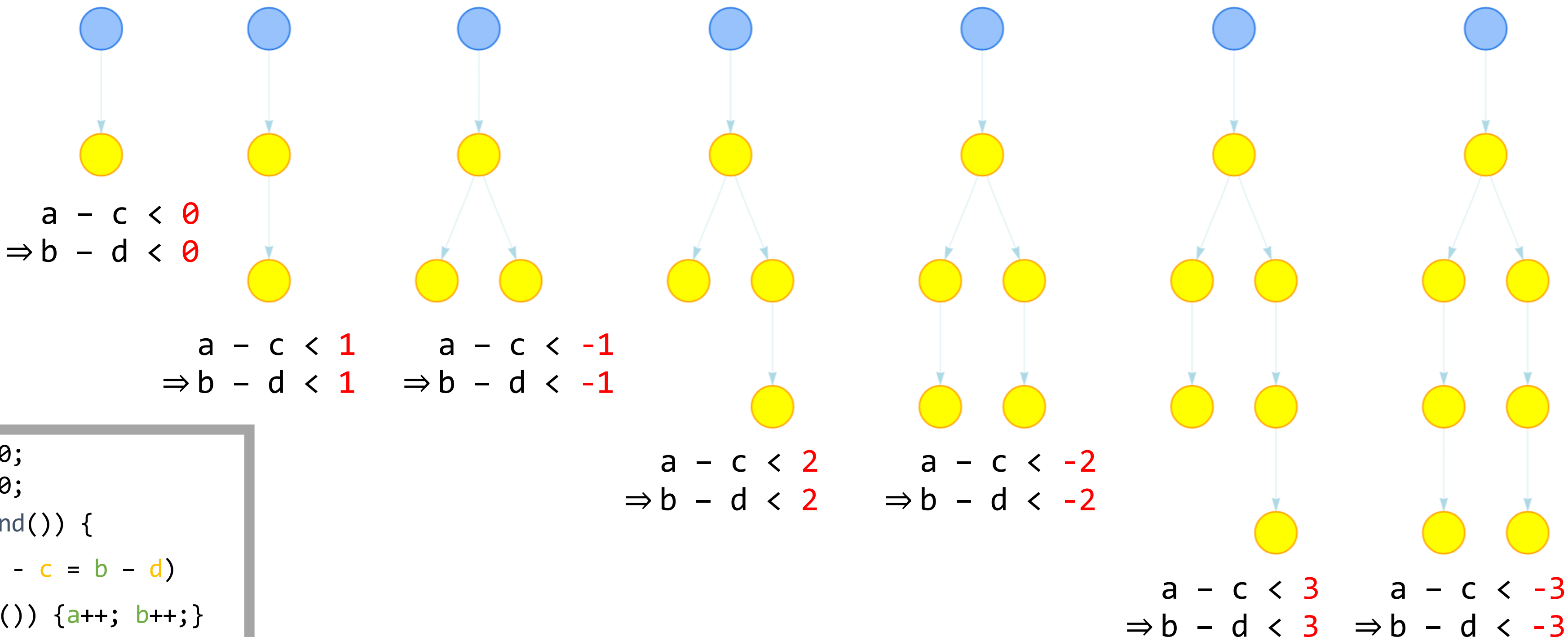
```
a, c = 0;
b, d = 0;
while (nd()) {
inv: (a - c = b - d)
 if (nd()) {a++; b++;}
 else      {c++; d++;}
}
assert (a < c ⇒ b < d);
```




```
a, c = 0;
b, d = 0;
while (nd()) {
inv: (a - c = b − d)
 if (nd()) {a++; b++;}
 else      {c++; d++;}
}
assert (a < c ⇒ b < d);
```




```
a, c = 0;
b, d = 0;
while (nd()) {
inv: (a - c = b − d)
 if (nd()) {a++; b++;}
 else      {c++; d++;}
}
assert (a < c ⇒ b < d);
```




```
a, c = 0;
b, d = 0;
while (nd()) {
inv: (a - c = b – d)
 if (nd()) {a++; b++;}
 else      {c++; d++;}
}
assert (a < c ⇒ b < d);
```

Data Driven Generalization & Lemma Discovery

Global view of the current solver state

- **group** lemmas (and pobs) based on syntactic/semantic similarity
  - we currently use anti-unification on interpreted constants
- detect whenever global proof is diverging and mitigate

One lemma to rule them all

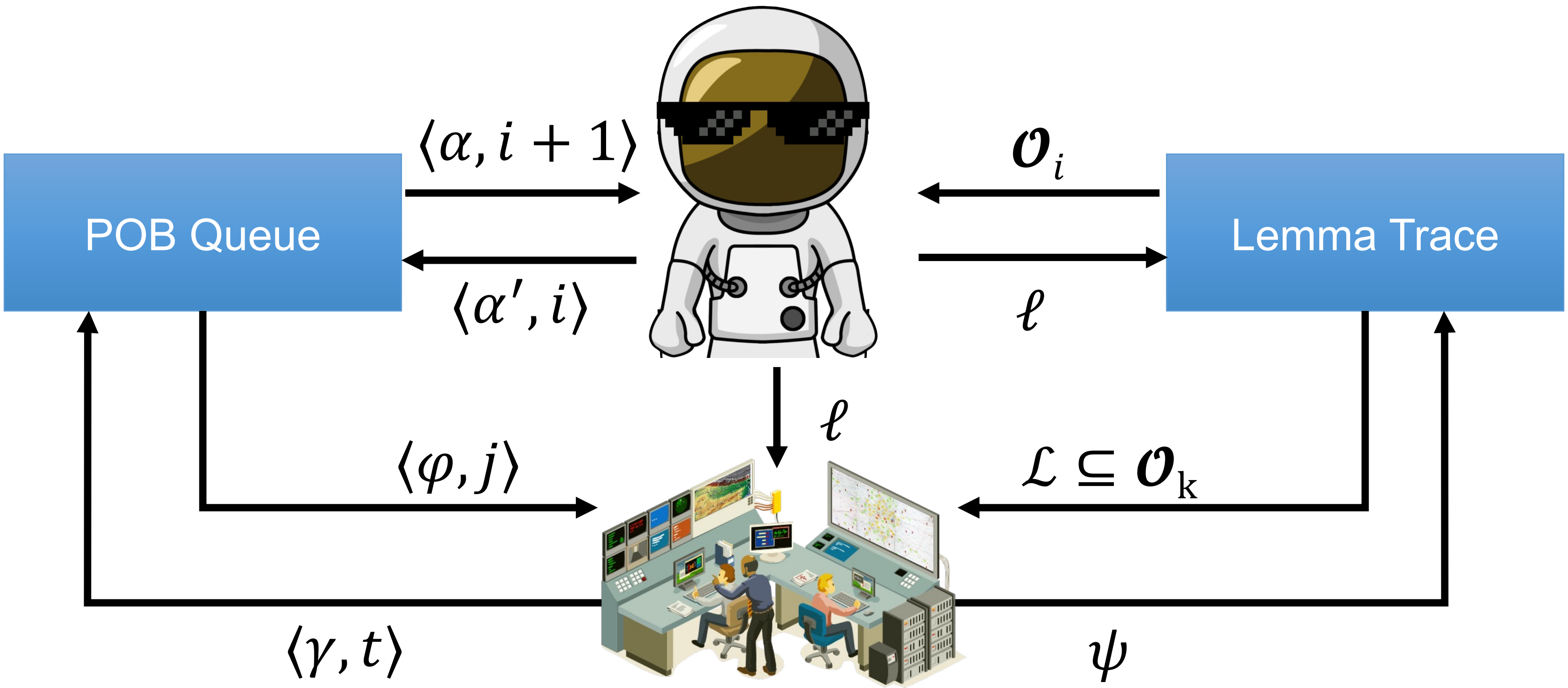
- **merge** lemmas in group to form a single *universal* lemma
- interpolation and inductive generalization can be applied to generalize further
- new lemma reduces the global proof by blocking all POBs in its group

Reduce, reuse, recycle

- under-approximate groups that cannot be merged in current theory
- learn multiple (simple) lemmas to block a (complex) proof obligation

Ground Control to Spacer Tom:

Global Guidance



Ground Control to Spacer Tom:

Global Guidance trinity

Subsume

combine multiple lemmas into one

Concretize

simplify terms by concrete values

Conjecture

drop literals that are in the way

1st Global Guidance to GSpacer Tom:

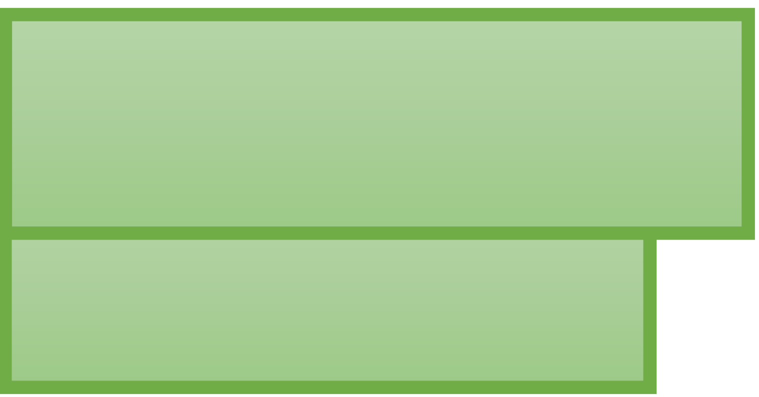
Subsume Rule

```
if (∃ψ · ∀ℓ ∈ ℒ · ψ ⇒ ℓ) then
   add ψ to trace
```

1st Global Guidance to GSpacer Tom:

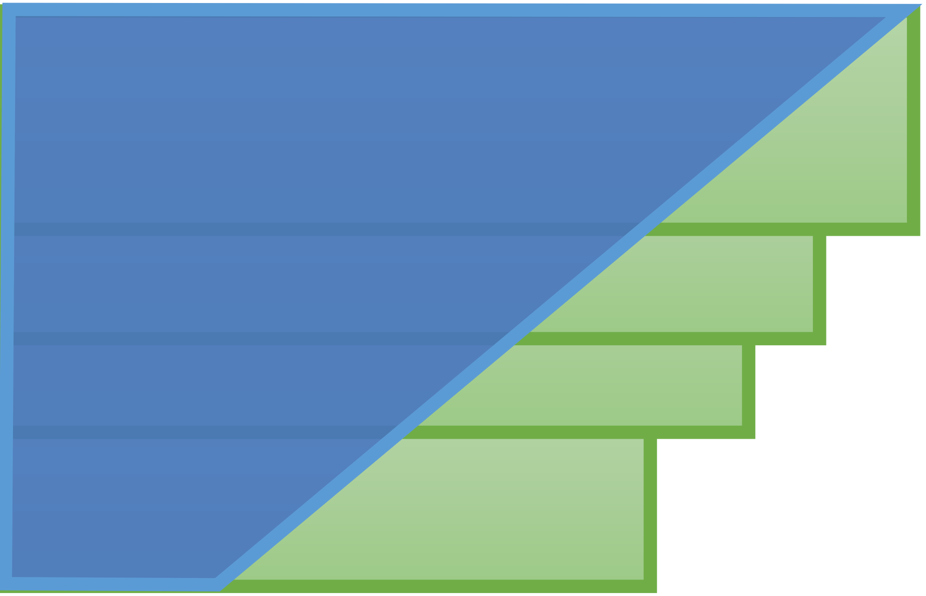
Subsume Rule

if $(\exists \psi \cdot \forall \ell \in \mathcal{L} \cdot \psi \Rightarrow \ell)$ then
  add ψ to trace



1st Global Guidance to GSpacer Tom:

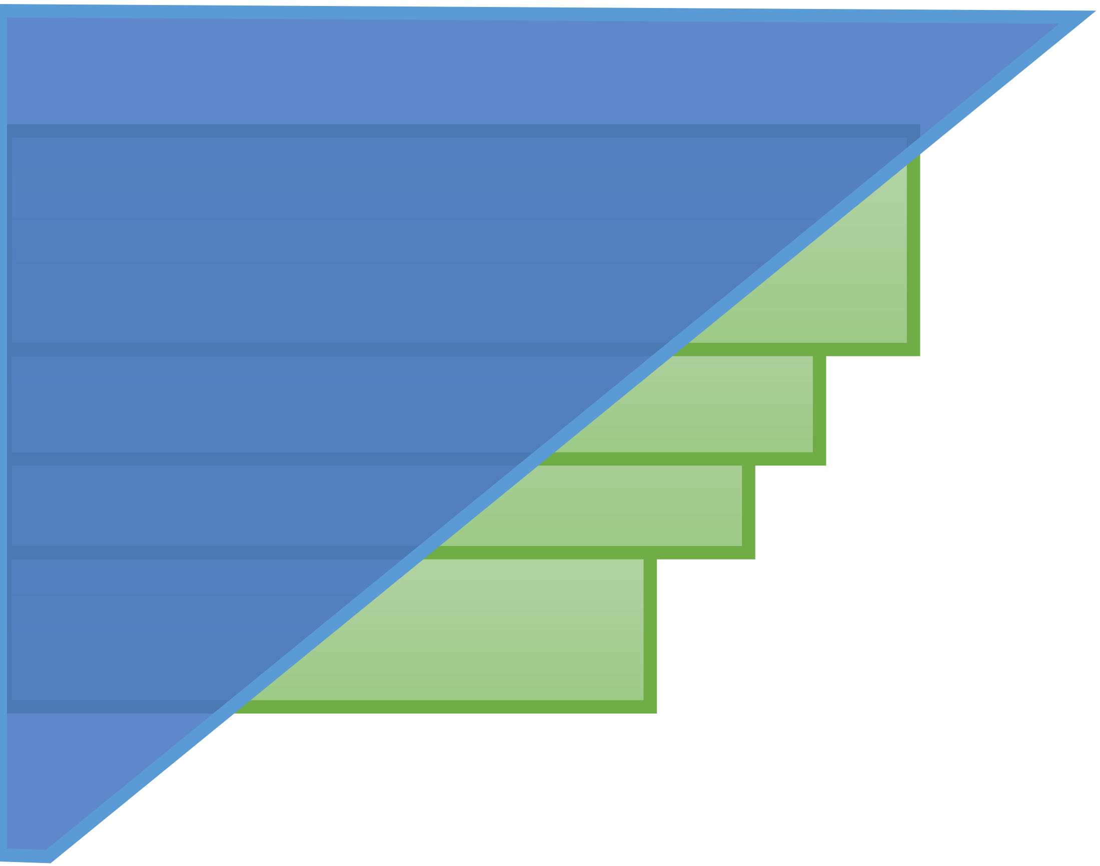
Subsume Rule



```
if (∃ψ · ∀ℓ ∈ ℒ · ψ ⇒ ℓ) then
   add ψ to trace
```

1st Global Guidance to GSpacer Tom:

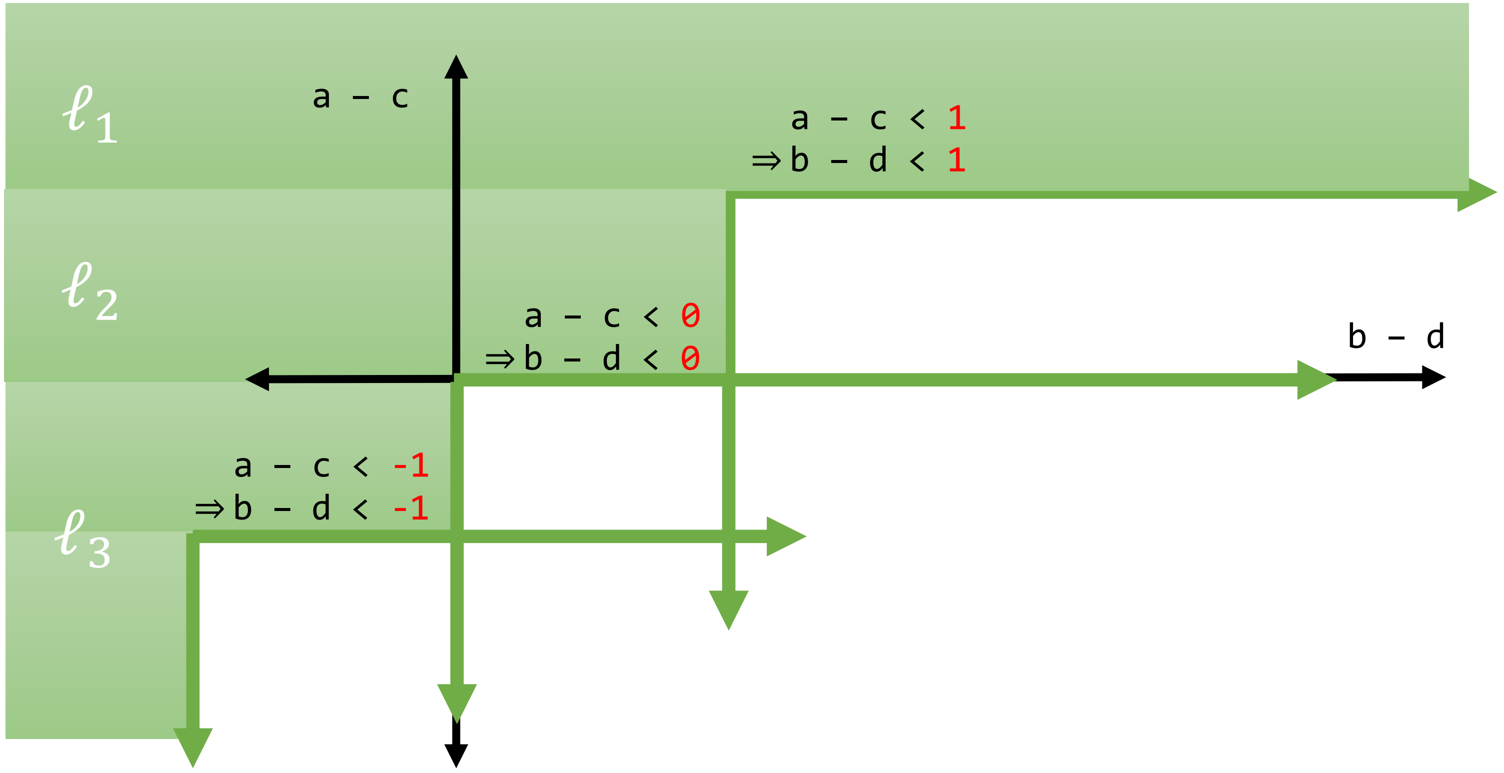
Subsume Rule



if $(\exists \psi \cdot \forall \ell \in \mathcal{L} \cdot \psi \Rightarrow \ell)$ then
    add ψ to trace

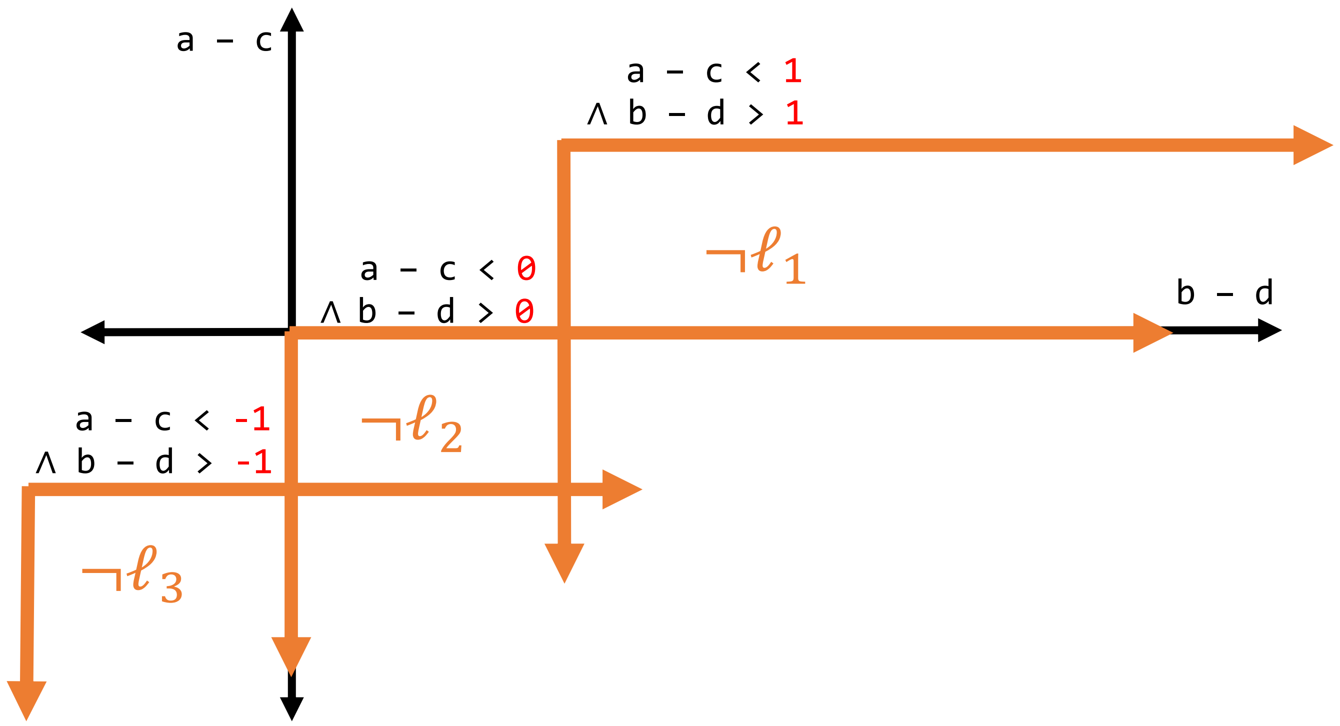
Subsume Rule in Action:

Subsume Rule on LIA



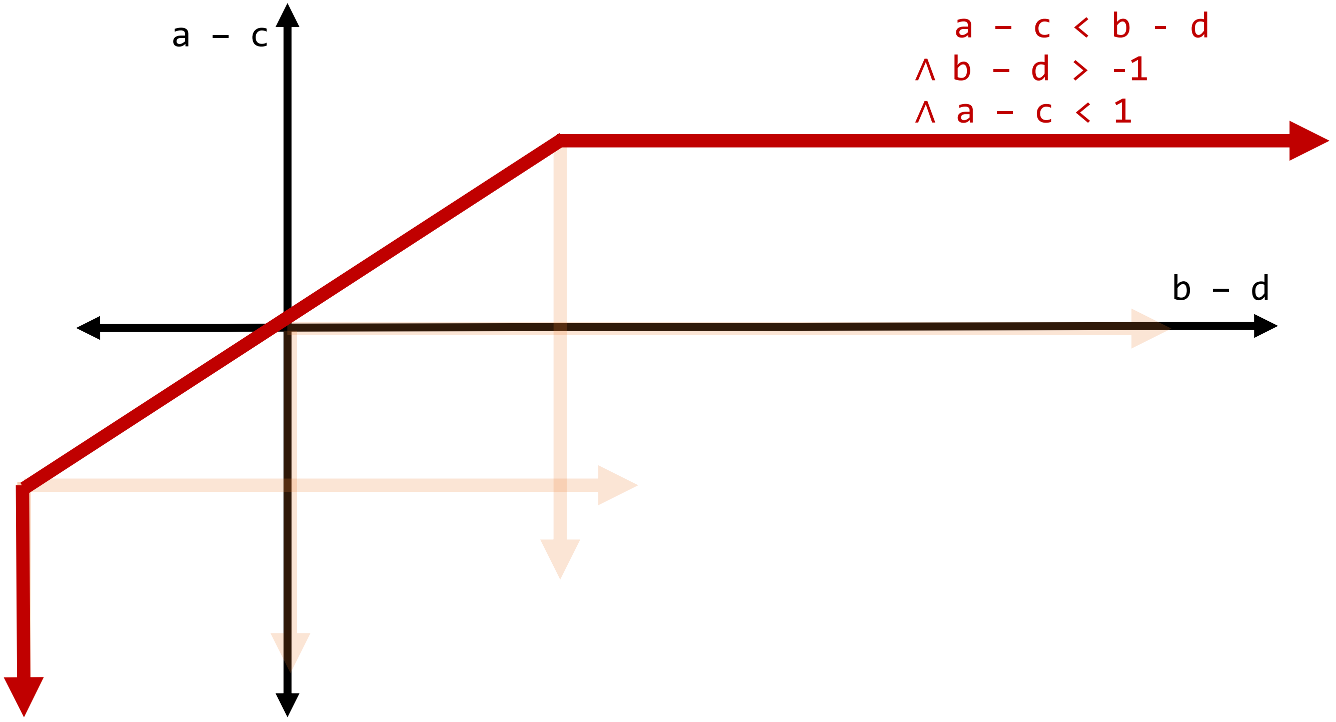
Subsume Rule in Action:

Subsume Rule on LIA



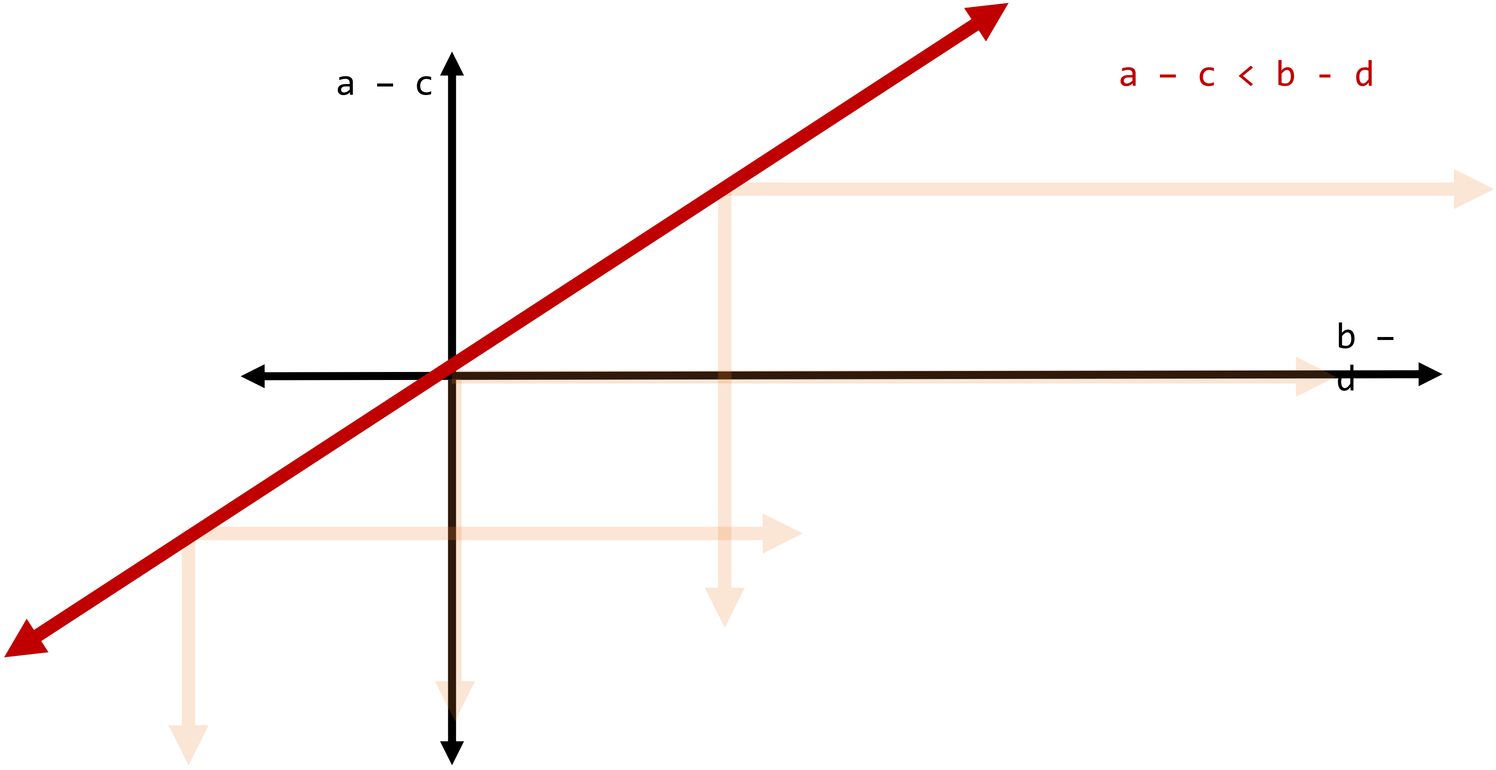
Subsume Rule in Action:

Subsume Rule on LIA



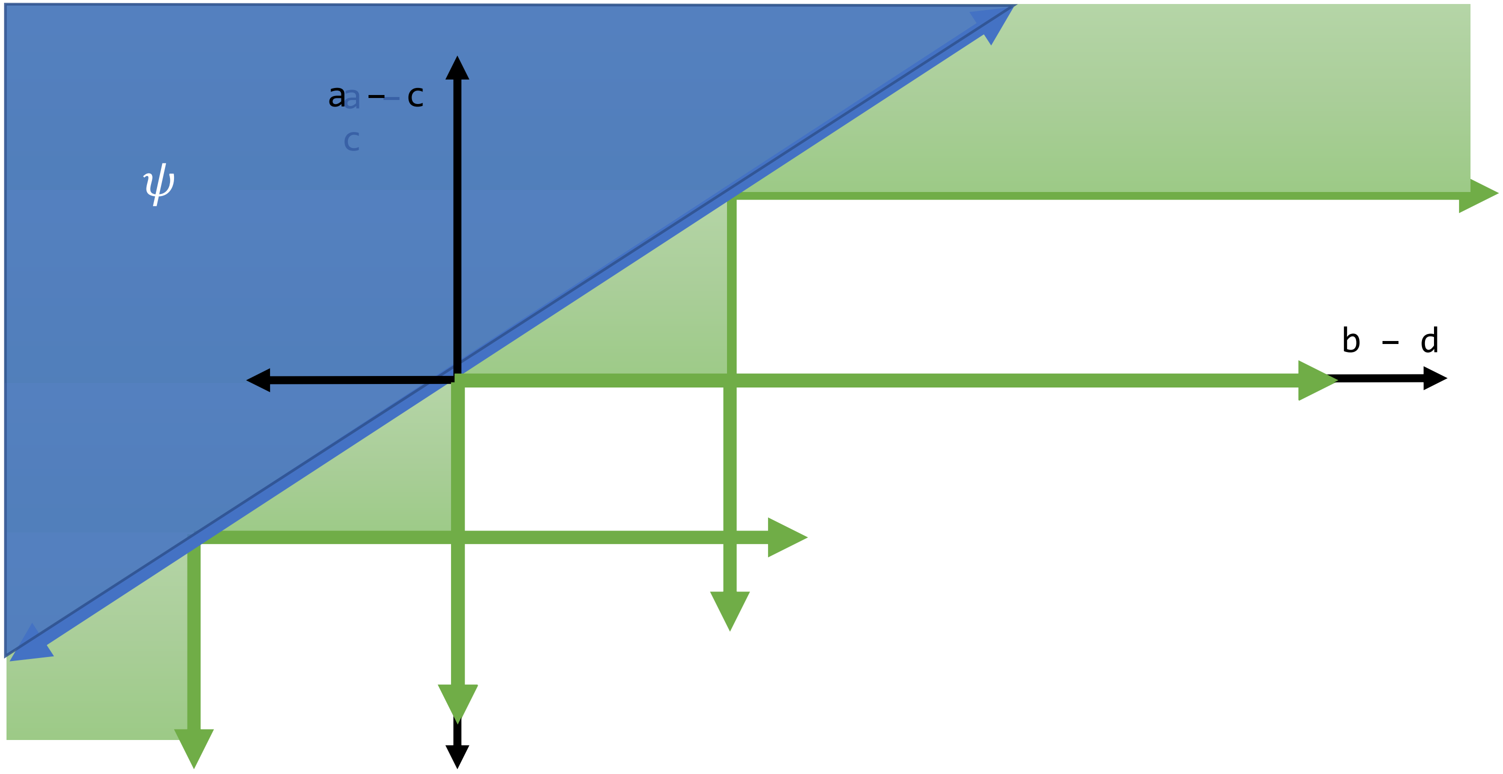
Subsume Rule in Action:

Subsume Rule on LIA



Subsume Rule in Action:

Subsume Rule on LIA



Summary of Subsume Rule in Spacer

In general, subsume rule requires

- a method to **cluster** lemmas/pobs together to discover common pattern
- i.e., multiple lemmas are *different* instances of some more general pattern
- a method to **merge** lemmas into a single lemma that strengthens all lemmas in a cluster

For LIA, we implement subsume as follows

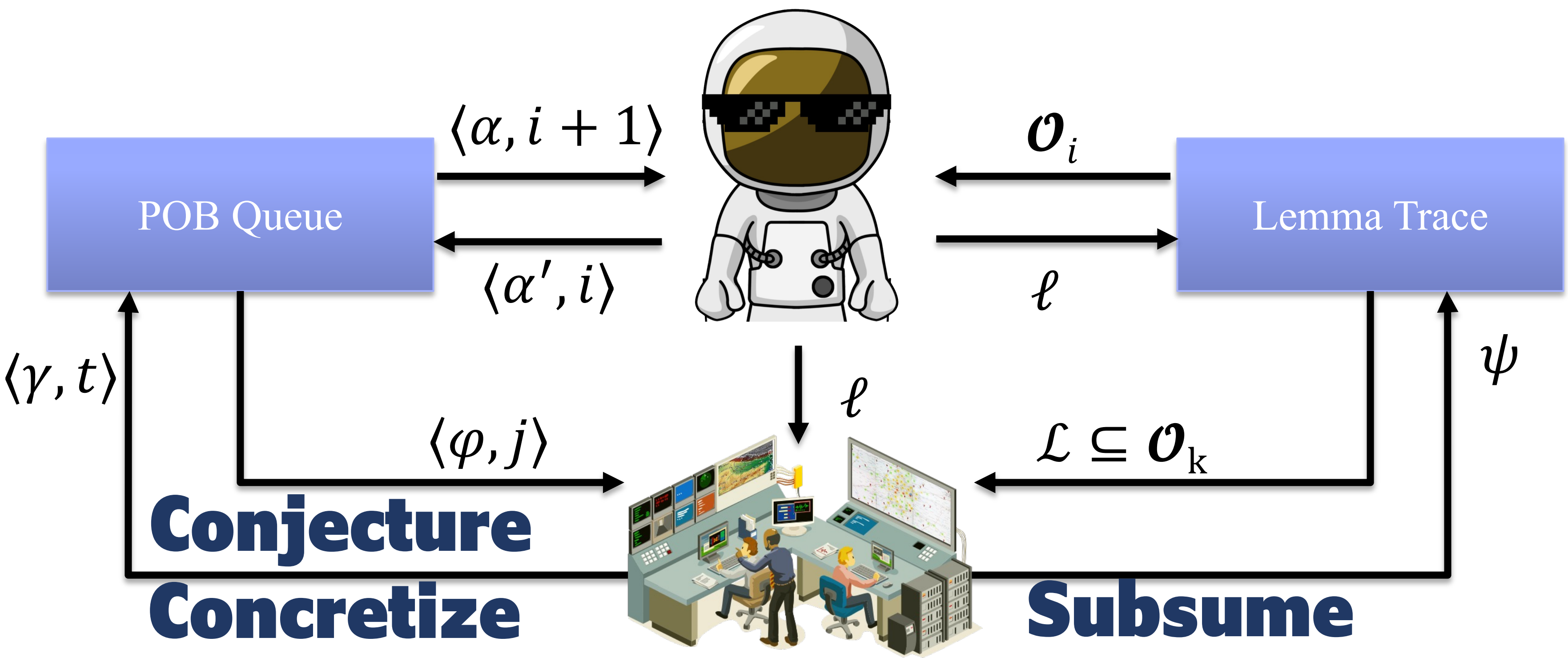
- **clustering**: anti-unification on constants
- two lemmas are cluster if they only differ in some numeric constants
- **merge**: convex closure (CC)
- implement CC symbolically using quantifier elimination, approximate by MBP

Integrated in Spacer via the strategy

  – cluster lemmas as they are learned
  – when the cluster is large enough, merge and create a conjecture
  – add conjecture as may pob

Ground Control to Spacer Tom:

Global Guidance



Implementation and Evaluation

Implemented in Spacer (still PR in Z3)

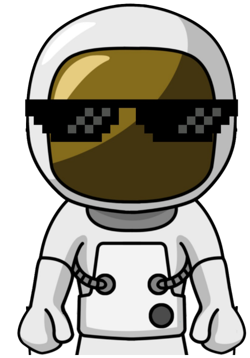
- https://github.com/Z3Prover/z3/pull/6026

Supports

- Linear Integer Arithmetic, Linear Real Arithmetic
- Linear and Non-linear CHCs
- (in progress) Quantified Arrays and Fixed-Size Bit-Vectors

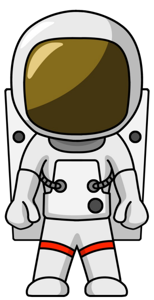
Evaluated on LIA instances from CHC-COMP

Evaluation on CHC-COMP



No interpolation!

Bench	SPACER						GSPACER						VBS	
	fw		bw		sc		fw		bw		sc			
	safe	unsafe	safe	unsafe	safe	unsafe	safe	unsafe	safe	unsafe	safe	unsafe	safe	unsafe
CHC-18	159	66	163	**69**	123	68	**214**	67	**214**	63	**214**	**69**	229	74
CHC-19	193	84	186	84	125	84	**202**	84	196	**85**	200	84	207	85

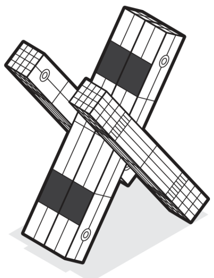


fw and ***bw*** are different interpolation strategies
sc configuration disables interpolation

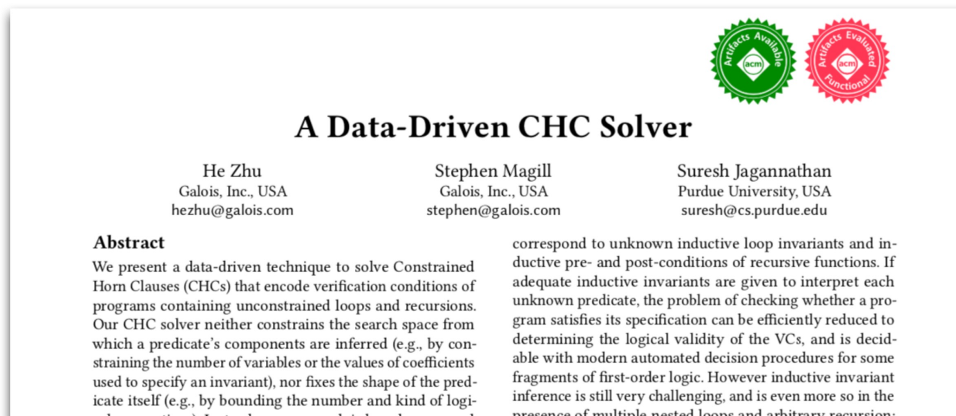
GSpacer won 3 of the 4 tracks at CHC-COMP 2020

UNIVERSITY OF WATERLOO

Linear Arbitrary (LArb) from PLDI 18



Data-driven, machine learning based invariant inference algorithm



Artifacts Available
Artifacts Evaluated Functional

A Data-Driven CHC Solver

He Zhu
Galois, Inc., USA
hezhu@galois.com

Stephen Magill
Galois, Inc., USA
stephen@galois.com

Suresh Jagannathan
Purdue University, USA
suresh@cs.purdue.edu

Abstract

We present a data-driven technique to solve Constrained Horn Clauses (CHCs) that encode verification conditions of programs containing unconstrained loops and recursions. Our CHC solver neither constrains the search space from which a predicate's components are inferred (e.g., by constraining the number of variables or the values of coefficients used to specify an invariant), nor fixes the shape of the predicate itself (e.g., by bounding the number and kind of logi-

correspond to unknown inductive loop invariants and inductive pre- and post-conditions of recursive functions. If adequate inductive invariants are given to interpret each unknown predicate, the problem of checking whether a program satisfies its specification can be efficiently reduced to determining the logical validity of the VCs, and is decidable with modern automated decision procedures for some fragments of first-order logic. However inductive invariant inference is still very challenging, and is even more so in the presence of multiple nested loops and arbitrary recursion;

Evaluation showed promise on
a subset of SV-COMP benchmarks

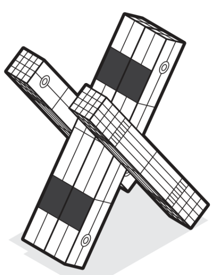
H. Zhu, S. Magill, S. Jagannathan: A data-driven CHC solver. PLDI 2018

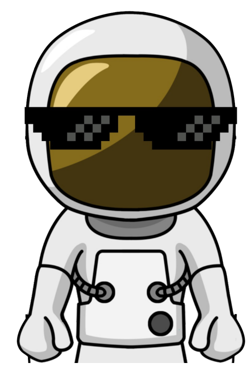
GSpacer versus LArb

On CHC-COMP instnaces, LArb is not competitive even against Spacer w/o global guidance

Instead, we compare GSpacer and LArb on benchmarks from PLDI'18 paper

Bench	SPACER		LARB		GSPACER		VB	
	safe	unsafe	safe	unsafe	safe	unsafe	safe	unsafe
PLDI18	216	**68**	270	65	**279**	**68**	284	68

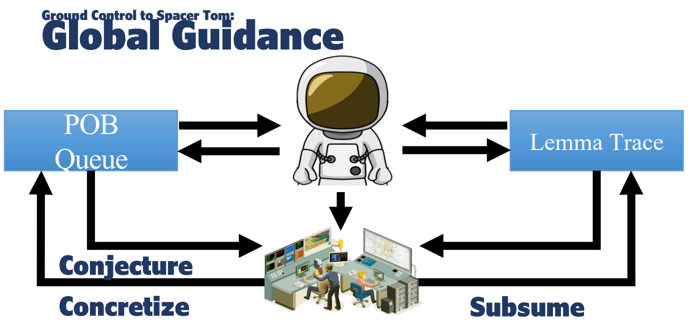


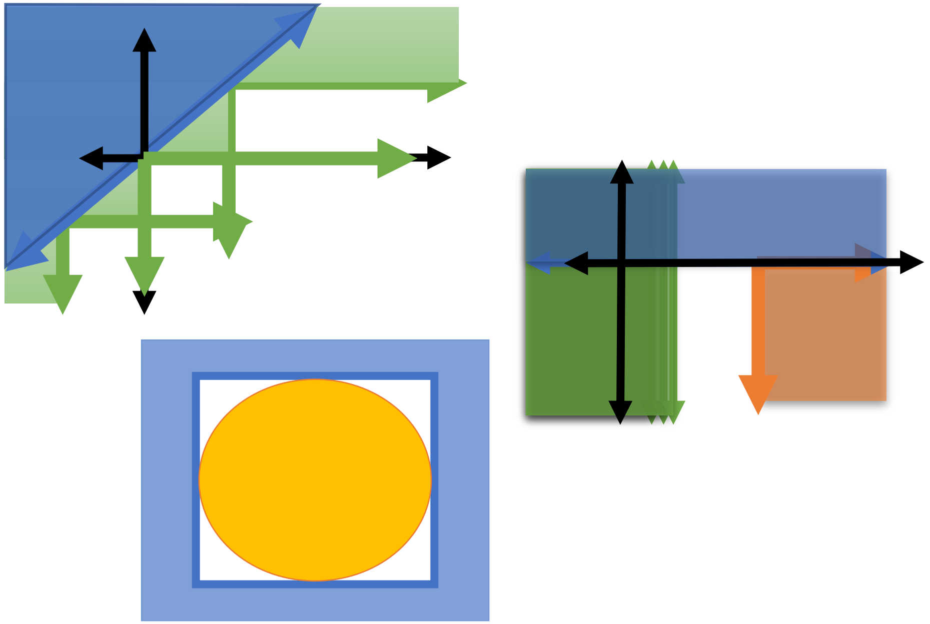


UNIVERSITY OF WATERLOO

VB stands for virtual best

Conclusion





Global guidance technique to mitigate limitations of local reasoning

Stable under different interpolation strategies

Data driven guidance for MC is better than both invariant inference and local reasoning

CHC MODULO BIT-VECTORS

Hari Govind V. K., G. Fedyukovich, G: Word Level Property Directed Reachability. ICCAD 2020

Motivating example

```
uint32_t x = 1, y = 1;
while (1)
invariant: (x + y) & 1 == 0
{
  x = x + 2 * nd();
  y = y + 2 * nd();
  assert(x + y != 1);
}
```

`nd()` returns a non-deterministic `uint32_t` value.

Predecessors to Bad states:

`(x = -1, y = -2),`
`(x = 1, y = 0),`
} Very specific

`(x + y = 1),`
`(x + y = -3)` …

How to generalize from

`(x + y != 1),`
`(x + y != -3)` …

to

`((x + y) & 1 == 0)` ?

Computing predecessors

PROBLEM:

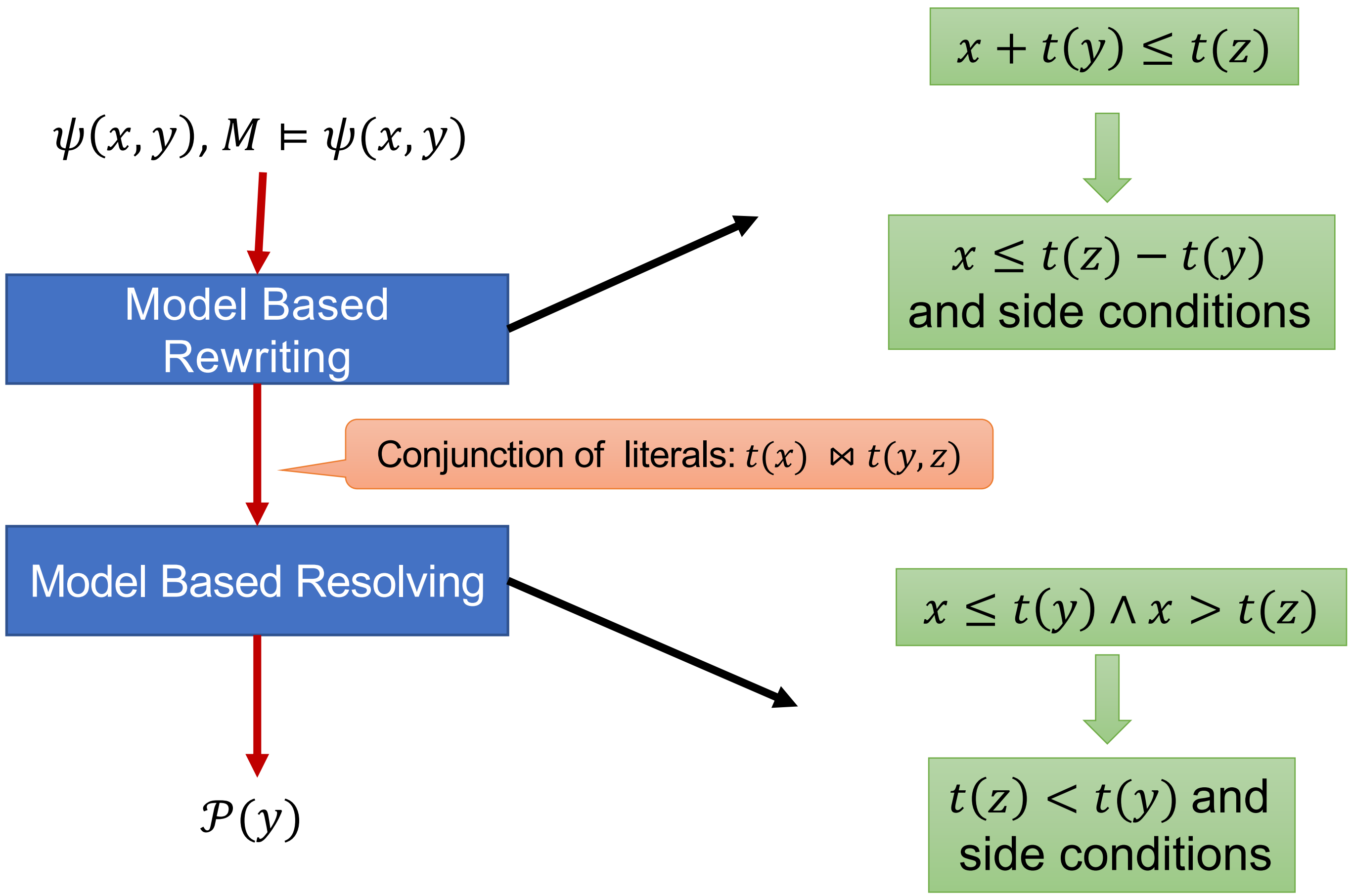
Given $\varphi(x')$, find $\alpha(x)$ such that
$\alpha(x) \wedge Tr(x, x') \wedge \varphi(x')$ is SAT

Eliminate x' from $\exists x' \cdot Tr(x, x') \wedge \varphi(x')$

Model Based Projection (MBP):

Compute: $\mathcal{P}(y) \Rightarrow \exists x \cdot \psi(x, y)$
Given: $M \vDash \psi(x, y)$

MBP for arithmetic operators in BV



MBP for arithmetic operators in BV

$$\frac{t(x) \geq z - y \quad t(x) \leq -y-1 \quad y \neq 0}{t(x) + y \geq z}\,[add_5] \qquad \frac{y = 0 \quad z \leq t(x)}{t(x) + y \geq z}\,[add_6]$$

$\psi(x, y), M \models \psi(x, y)$

Model Based Rewriting

$$\frac{}{t(x)+y \leq z}\,[add_1] \quad \frac{}{t(x)+y \leq z}\,[add_2] \quad \frac{}{t(x)+y \leq z}\,[add_3] \quad \frac{}{t(x)+y \geq z}\,[add_4]$$

$$\frac{t(x) \geq z - y \quad t(x) \leq -y-1 \quad y \neq 0}{t(x)+y \geq z}\,[add_5] \quad \frac{y = 0 \quad z \leq t(x)}{t(x)+y \geq z}\,[add_6] \quad \frac{y \neq 0 \quad z \leq y-1 \quad x \leq -y-1}{t(x)+y \geq z}\,[add_7]$$

$$\frac{y \leq t_2(x) - t_1(x) \quad t_1(x) \leq t_2(x)}{t_1(x)+y \leq t_2(x)}\,[bothx_1] \quad \frac{y \leq t_2(x) - t_1(x) \quad -t_1(x) \leq y}{t_1(x)+y \leq t_2(x)}\,[bothx_2] \quad \frac{-t_1(x) \leq y \quad t_1(x) \leq t_2(x) \quad t_1(x) \neq 0}{t_1(x)+y \leq t_2(x)}\,[bothx_3]$$

$$\frac{a \leq b \; b \leq a}{a = b}\,[eq] \quad \frac{a < b}{a \neq b} \quad \frac{a > b}{a \neq b}\,[neq] \quad \frac{b \leq a-1 \quad 1 \leq a}{\neg(a \leq b)}\,[nule] \quad \frac{-y \leq t(x)}{-t(x) \leq y} \quad \frac{t(x) \leq -y}{y \leq -t(x)}\,[inv] \quad \frac{x \leq \frac{2^n k_2}{k_1}}{k_1 x \leq k_2 x}\,[bothx_4]$$

Figure 2: Rewrite rules for BV arithmetic. Terms $t_1(x)$, $t_2(x)$, and $t(x)$ contain constant x. Terms y and z do not contain x. Terms a and b may or may not contain x. Rules add_1 to add_7 rewrite unsigned inequalities so that $t(x)$ is the sole term on one side of the inequality. Rules $bothx_1$ to $bothx_4$ rewrite inequalities that contain x on both sides. Rules inv remove the negation of the x term.

Conjunction of literals: $t(x) \bowtie t(y, z)$

Model B

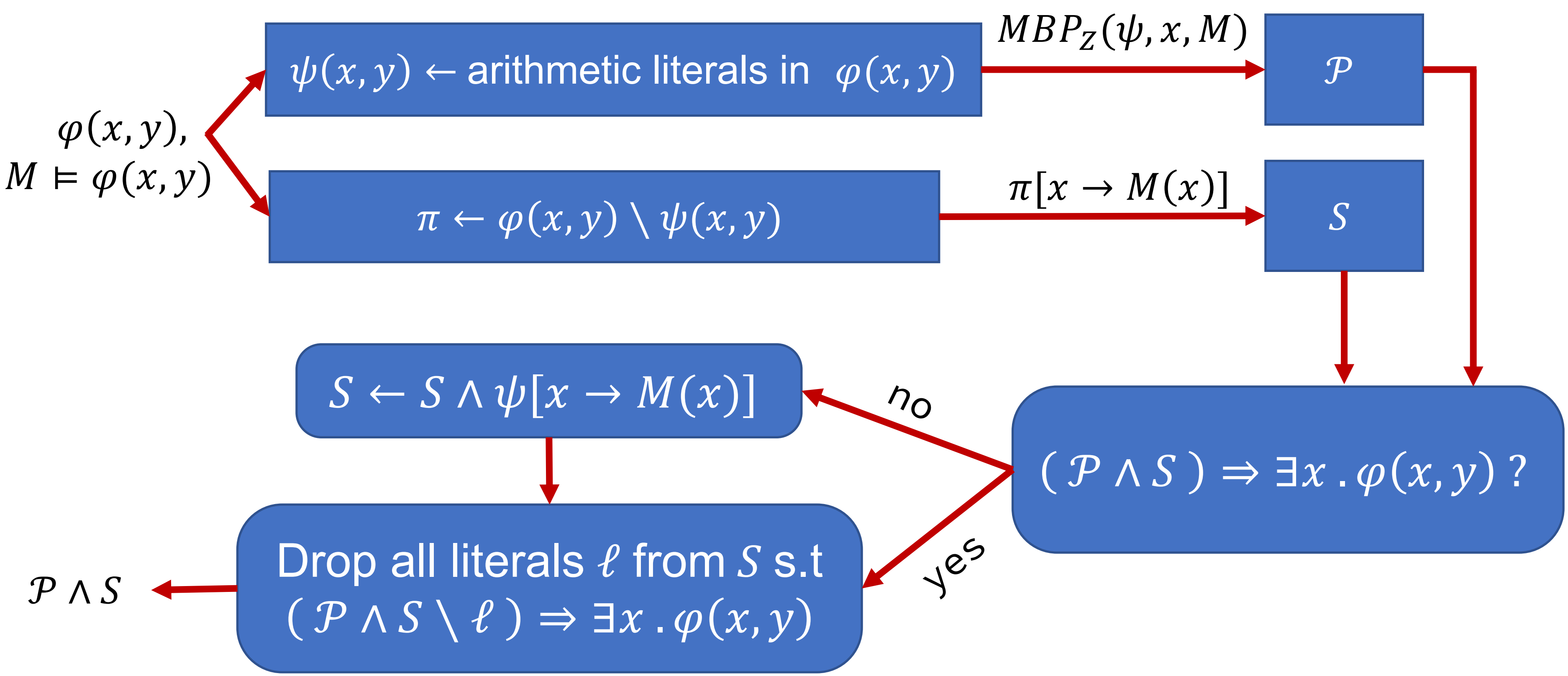
$$\mathrm{MBP}_{\mathbb{Z}}\left(M, \psi \wedge \left(\bigwedge_i a_i < \alpha_i \times x\right) \wedge \left(\bigwedge_j \beta_j \times x \leq b_j\right)\right) \stackrel{\text{def}}{=} \psi \wedge$$

$$(a_L \times (\textsc{LCM}\ \mathtt{div}\ \alpha_L)\ \mathtt{div}\ \textsc{LCM}) < (b_U \times (\textsc{LCM}\ \mathtt{div}\ \beta_U)\ \mathtt{div}\ \textsc{LCM}) \wedge$$

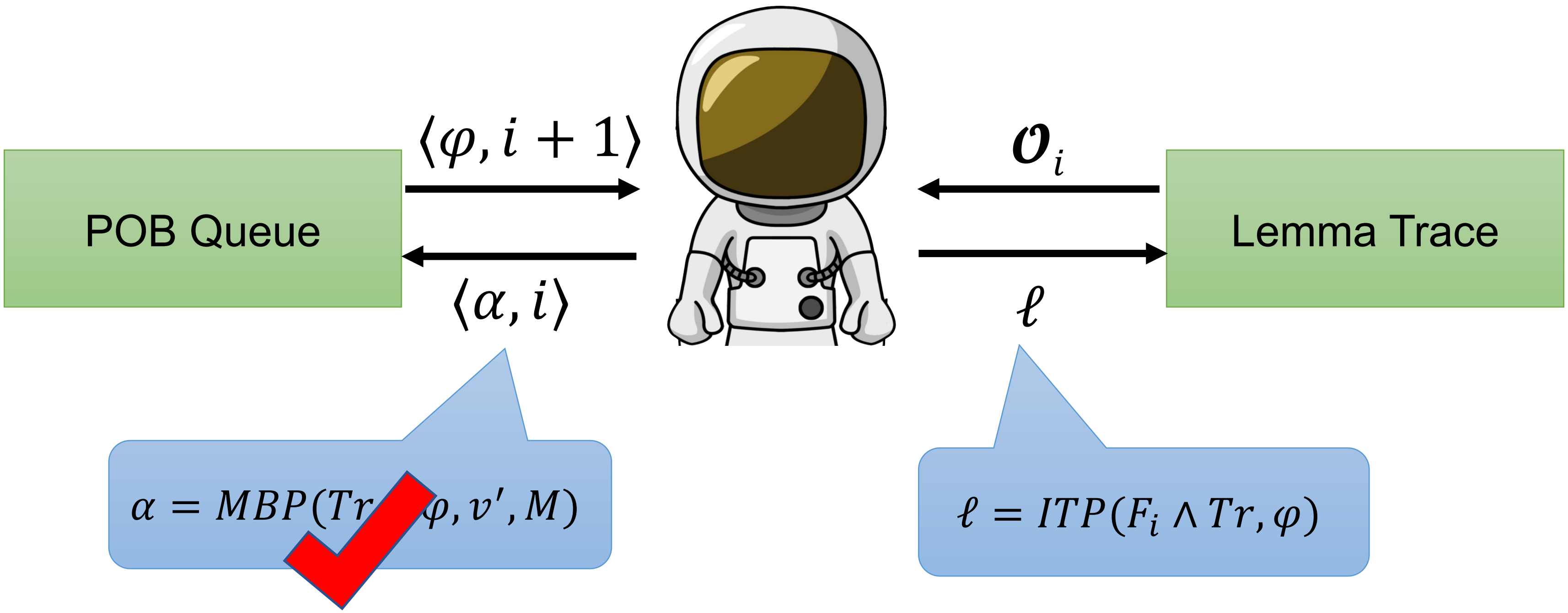
$$\bigwedge_i a_i \leq (2^n - 1)\ \mathtt{div}\ (\textsc{LCM}\ \mathtt{div}\ \alpha_i) \wedge$$

- $M(x) \times \textsc{LCM} \in \mathbb{Z}_{2^n-1}$, where $M(x)$ is the value of x in M,
- for each i, $M \models a_i \leq (2^n - 1)\ \mathtt{div}\ (\textsc{LCM}\ \mathtt{div}\ \alpha_i)$, and
- for each j: $M \models b_j \leq (2^n - 1)\ \mathtt{div}\ (\textsc{LCM}\ \mathtt{div}\ \beta_j)$.

MBP for full BV



Spacer



But there are no good interpolation strategies for BV !!!

Instantiated guidance rules for BV

Subsume

```
if (∃ψ · ∀ℓ ∈ ℒ · ψ ⇒ ℓ) then
    add ψ to trace
```

Conjecture

```
if (φ ≡ α ∧ β) ∧
    (∀ℓ ∈ ℒ · ℓ blocks β but does not block α) then
        add α to POB queue
```

beyond Spacer

CHC SOLVERS

CHC COMP

Report on the 2022 edition

https://chc-comp.github.io/

Emanuele De Angelis, Inst. for Systems Analysis and Computer Science - National Research Council, Italy

Hari Govind V K, University of Waterloo, Canada

https://chc-comp.github.io/CHC-COMP2022_presentation.pdf

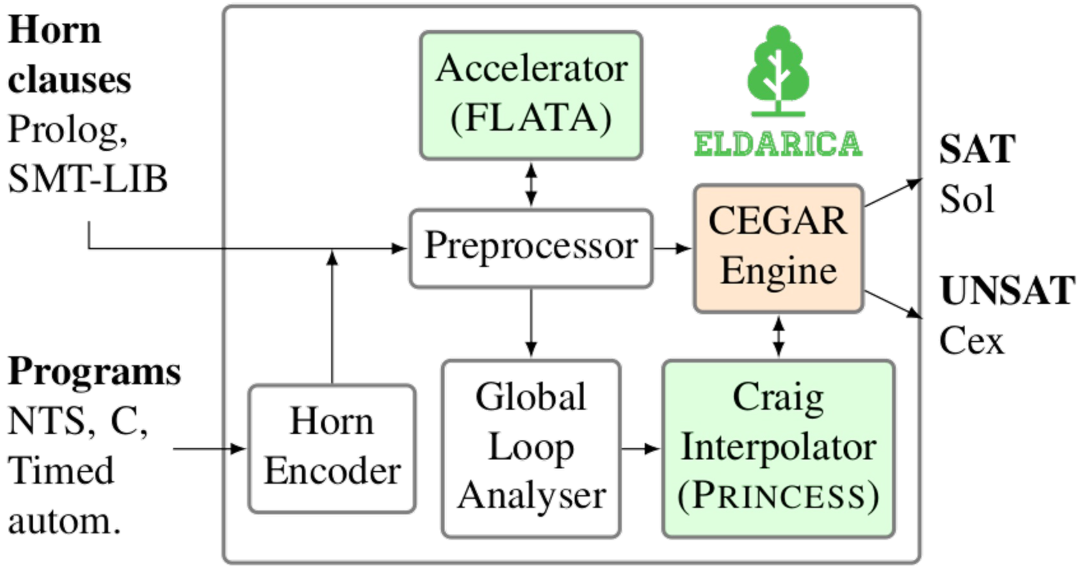
The Eldarica Horn Solver

- A Horn solver tailored to verification of software and infinite-state systems
  - **Algorithms:** Predicate abstraction, CEGAR, Craig interpolation
  - **Theories:** LIA, NIA, BV, ADTs, arrays, heap
  - **Input formats:** SMT-LIB, Prolog (+ built-in C front-end)
  - **Output:** Full solution + counterexample output

- Open-source, entirely implemented in Scala
- Started in 2011, since then developed continuously
  - E.g., integration of new theories, hand-in-hand with development of new decision/interpolation procedures
  - Upcoming: LRA, improved heap support

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

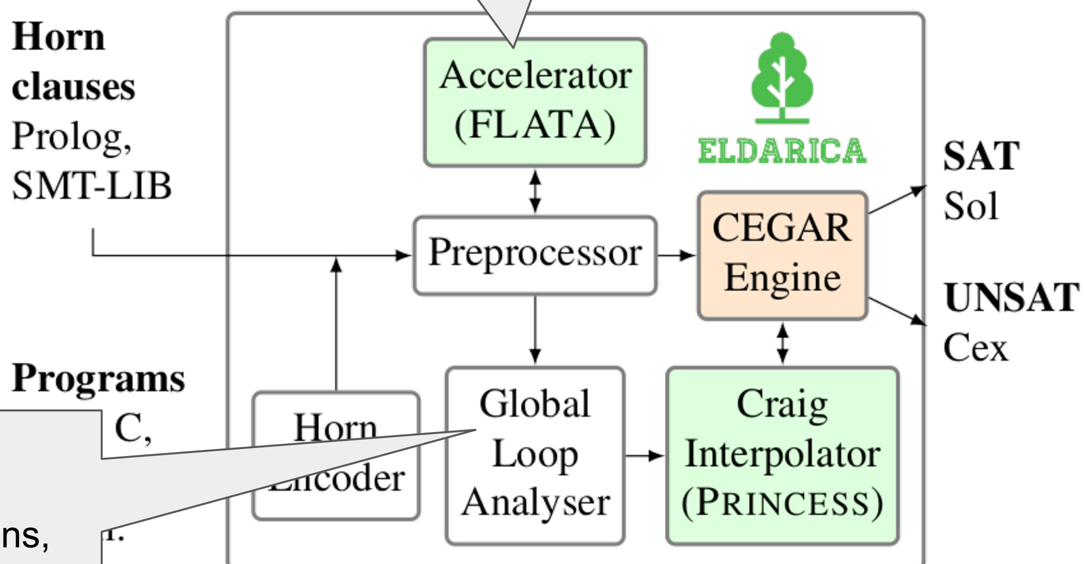
Architecture



Hossein Hojjat, Philipp Rümmer: **“The ELDARICA Horn Solver”** (FMCAD 2018)

Key feature: global invari… …tion methods

Replace cycles with closed-form representations



Identify arithmetic progressions, variable relationships, etc.

FreqHorn: CHC solving by enumerative search

G. Fedyukovich, S. Kaufman, R. Bodik, FMCAD'17

High-level view:

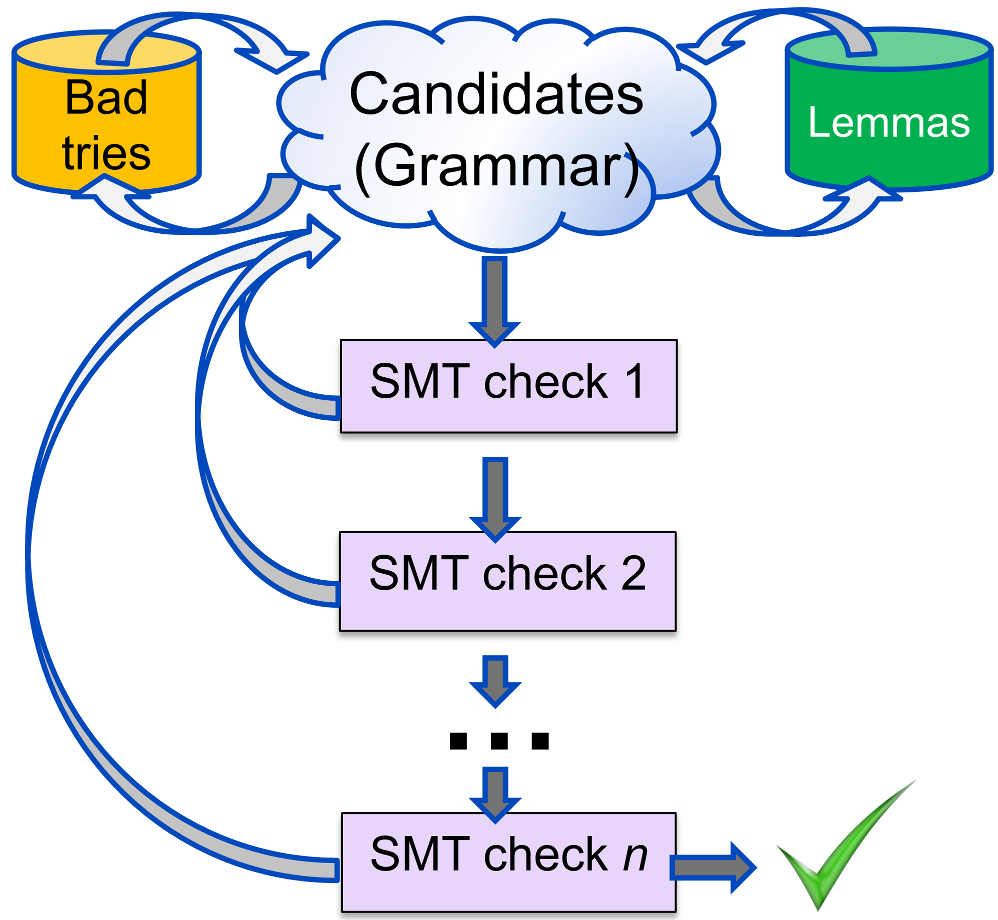
- Loop between a candidate generator and SMT-solver
- Synthesizes lemmas separately

Candidate generator

- Syntax-Guided Synthesis (SyGuS)
- Non-recursive grammars obtained from ASTs of verification conditions
- Learnes from positive / negative candidates

SMT-based decision maker

- Off-the-shelf SMT solver
- Does not need interpolation or quantifier elimination
- Easy to maintain



FreqHorn: CHC applications/extensions

Safety of numerical programs:

- Accelerated using interpolation and Houdini — Fedyukovich and R. Bodík, TACAS'18
- Accelerated using data learning and quantifier elimination — Fedyukovich, Prabhu, Madhukar, Gupta, FMCAD'18
- Extended to arrays and quantified invariants — -||-, CAV'19
- Extended to disjunctive invariants — Riley and Fedyukovich, FSE'22

(Non)-termination of programs

- Ranking functions, recurrence sets — Fedyukovich, Zhang, Gupta, CAV'18

Modular analysis

- Generation of function summaries — Pick, Fedyukovich, Gupta, VMCAI'21
- Proving hyperproperties — Pick, Fedyukovich, Gupta, FMCAD'20

Specification synthesis

- From invariants to spec and back — Prabhu, Fedyukovich, Madhukar, D'Souza, PLDI'21

Test case generation

- Invariants block unreachable branches — Zlatkin and Fedyukovich, TACAS'22

UNIVERSITY OF WATERLOO

Golem: Overview

Solver for Constrained Horn Clauses

Developed at USI Formal Verification and Security Lab (Lugano, Switzerland) by Martin Blicha et al.

Craig interpolation-based algorithm for CHC solving

Tight integration with interpolating SMT solver OpenSMT

Supports linear real and integer arithmetic as the background theory

Available at https://github.com/usi-verification-and-security/golem

Golem: Brief History

Summer 2020

- first commits

Winter 2020

- Impact engine [McMillan ’06] (Lazy Abstraction with Interpolants)

March 2021

- 3 medals at CHC-COMP ’21

May 2021

- Spacer engine [Komuravelli et al. ’16]

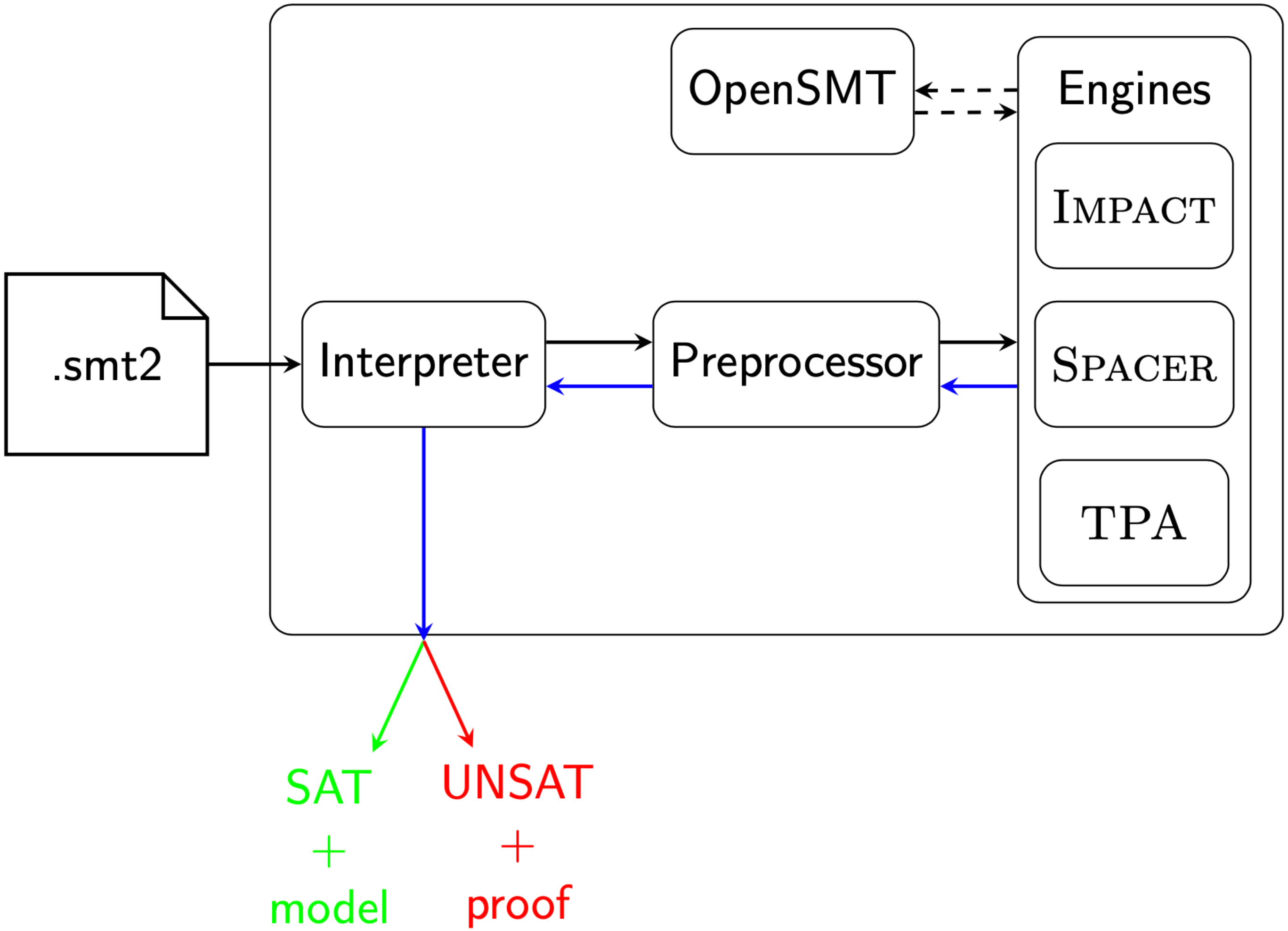
Summer 2021

- TPA engine [Blicha et al. ’22] (Transition Power Abstraction)

April 2022

- 4 medals at CHC-COMP ’22

Golem: Architecture



Golem: Future

Extend supported background theories (arrays, ADTs)

Extend TPA engine to support nonlinear CHC systems

Golem as backend for Korn

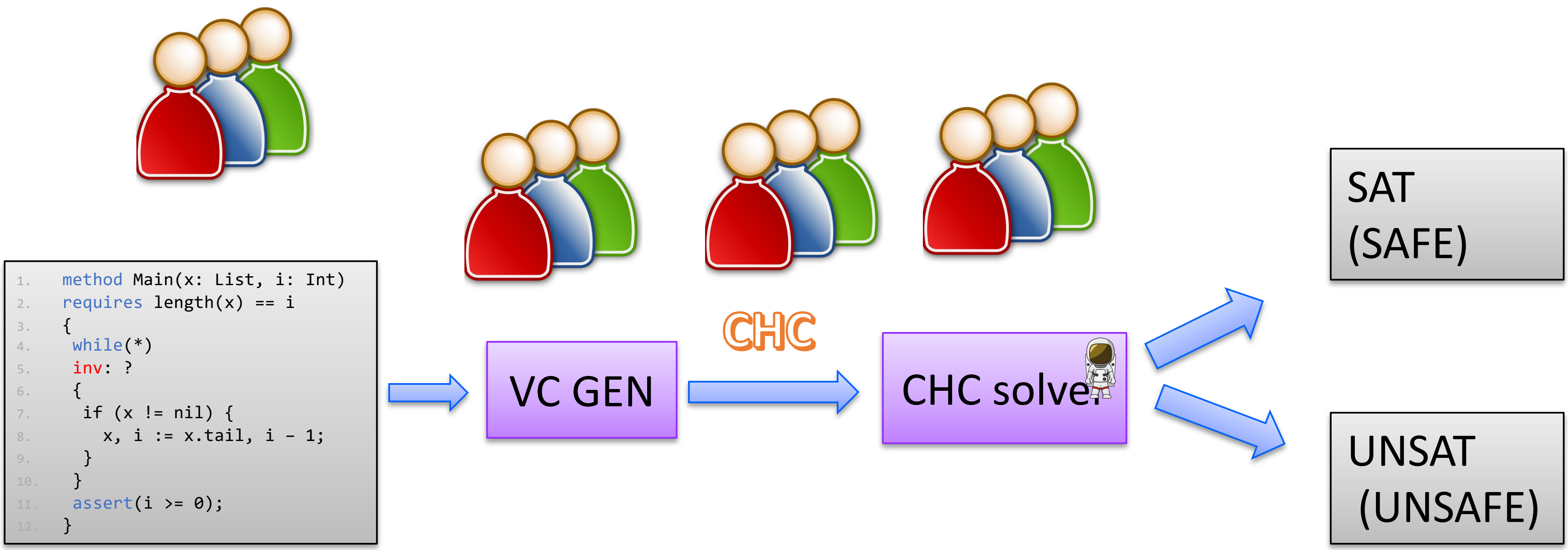
Golem as backend for SolCMC (Alt et al. ’22)

Develop proper API for Golem as library

Support Datalog input format

UNIVERSITY OF WATERLOO

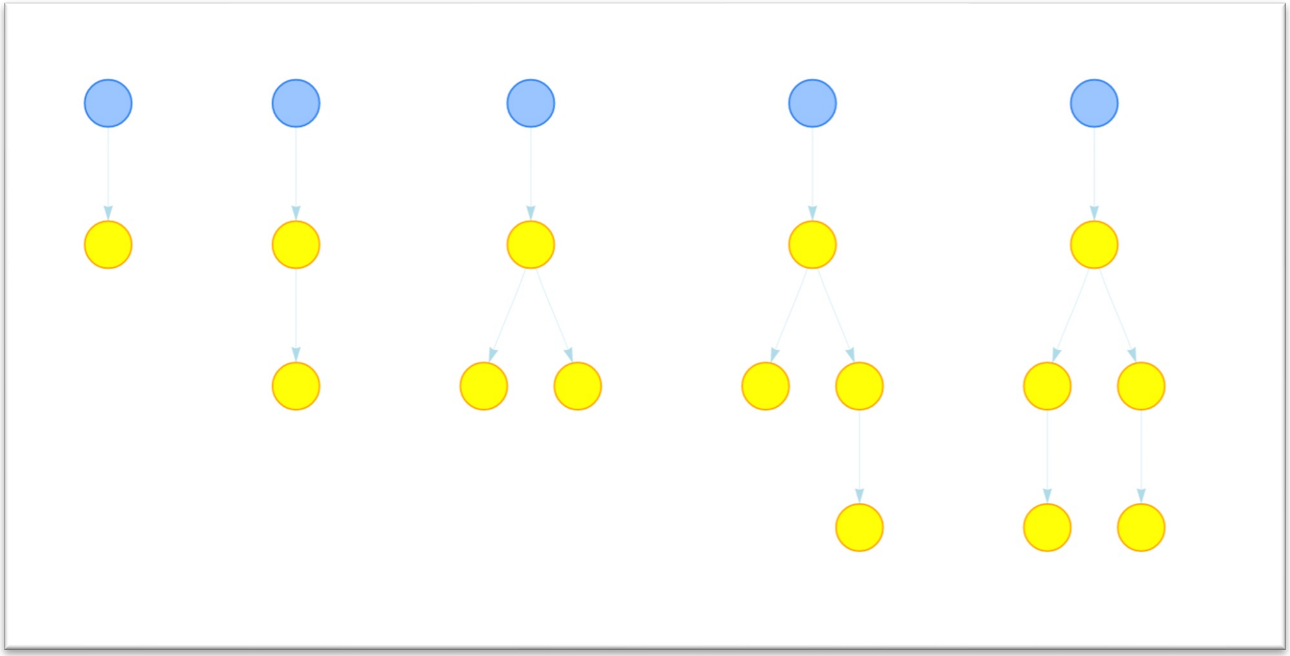
Helping Users of CHC Solvers



SeaHorn JayHorn rust-horn SOLIDITY

UNIVERSITY OF WATERLOO

HST: Spacer Visualizer (ver. 1)



Created by Matteo Marescotti as a side-project to understand Spacer behavior

Extremely useful in understanding what Spacer is doing

Intendent for internal use only

HST: Spacer Visualizer (ver. 2)

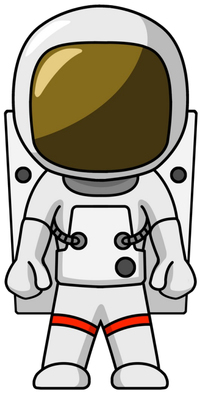
with Aishwarya Ramanathan, Nham Le, and Richard Trefler

- based on Vampire Visualizer by Bernhard Gleiss

https://github.com/nhamlv-55/spacer-visualization/

UNIVERSITY OF WATERLOO

Art, Science, and Magic of CHCs



Model Checking of Safety Properties is CHC satisfiability

- Logic: Constrained Horn Clauses (CHC)
- "Decision" procedure: Spacer
- Constraints: arithmetic, bv, **arrays**, **quantifiers**, **adt + recfn**, …

Art: finding the right encoding from the problem domain to logic

- the difference between easy to impossible
- encodings can "simulate" specialized algorithms

Science: Progress, termination (when decidable)

- while the underlying problem is undecidable, many fragment or sub-problems are decidable

Magic: actually solving useful problems

- interpolation, heuristics, generalizations, …
- the list is endless

END

ADT AND RECURSIVE FUNCTIONS

Automatic program verification

```
method Main(x: List, i: Int)
requires length(x) == i
{
 while(*)
 inv: length(x) == i
 inv: i >= 0
 {
  if (x != nil) {
    x, i := x.tail, i - 1;
  }
 }
 assert(i >= 0);
}
```

Algebraic Data Type

```
List: nil |
      cons(h, t)
```

Recursive Function

```
length(l):
   match l
     case nil => 0
     case cons(h, t) => 1 +
length(t)
```

How to automatically come up with inv in the presence of ADTs and RFs?

UNIVERSITY OF WATERLOO

Constrained Horn Clauses

```
method Main(x: List, i: Int)
requires length(x) == i
{
 while(*)
 inv: ?
 {
  if (x != nil) {
    x, i := x.tail, i - 1;
  }
 }
 assert(i >= 0);
}
```

Constraints on uninterpreted predicates

All constraints are horn clauses

$$length(x) = i \Rightarrow inv(x, i)$$

$$inv(x, i) \wedge x \neq nil \wedge x' = tail(x) \wedge i' = i - 1 \Rightarrow inv(x', i')$$

$$inv(x, i) \wedge i < 0 \Rightarrow \bot$$

CHC_f

modulo ADT,RF

AD
T

```
List: nil |
      cons(h, t)
```

RF

```
length(l):
   match l
     case nil => 0
     case cons(h, t) => 1 + length(t)
```

UNIVERSITY OF WATERLOO

Solutions to CHCs (inductive invariants of programs)

Any interpretation that satisfies all constraints

- Interpretations are expressed in some language

$$length(x) = i \Rightarrow inv(x, i)$$
$$inv(x, i) \land x \neq nil \land x' = tail(x) \land i' = i - 1 \Rightarrow inv(x', i')$$
$$inv(x, i) \land i < 0 \Rightarrow \bot$$

Solution

$$inv(x, i) \triangleq length(x) = i$$

Synthesize RFs

Construct solutions with those RFs

Apply RFs to arguments

UNIVERSITY OF WATERLOO

Encoding RF applications in CHCs

$$length(x) = i \Rightarrow inv(x, i)$$
$$inv(x, i) \wedge x \neq nil \wedge x' = tail(x) \wedge i' = i - 1 \Rightarrow inv(x', i')$$
$$inv(x, i) \wedge i < 0 \Rightarrow \bot$$

Solution

$$inv(x, i) \triangleq length(x) = i$$

Use ghost variables to capture RF applications (term abstraction)

$$length(x) = i \Rightarrow inv(x, i, \underline{i})$$
$$inv(x, i, \underline{j}) \wedge \underline{length(x) = j} \wedge x \neq nil$$
$$x' = tail(x) \wedge i' = i - 1 \wedge \underline{length(x') = j'} \Rightarrow inv(x', i', \underline{j'})$$
$$inv(x, i, \underline{j}) \wedge \underline{length(x) = j} \wedge i < 0 \Rightarrow \bot$$

Solutio
n

$$inv(x, i, j) \triangleq \underline{j} = i$$

Assume: RF applications are given

Search for solutions without RFs

UNIVERSITY OF WATERLOO

Challenge: RFs are hard!!!!

```
length(l):
   match l
     case nil => 0
     case cons(h, t) => 1 +
length(t)
```

☹ Need inductive reasoning

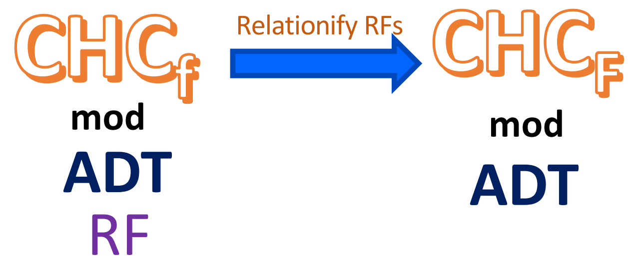
Given a list **`l`**, show **`length(l) >= 0`**

☹ 2 sources of undecidability

RFs without any Uninterpreted Predicates

CHCs without any RFs

Typical approach: Relationification



Encode RFs as CHCs

$$length(x) = j \longrightarrow Length(x, j)$$

$$length(x) = i \Rightarrow inv(x, i, i)$$

$$inv(x, i, j) \land length(x) = j \land x \neq nil \land$$
$$x' = tail(x) \land i' = i - 1 \land length(x') = j' \Rightarrow inv(x', i', j')$$

$$inv(x, i, j) \land length(x) = j \land i < 0 \Rightarrow \bot$$

Typical approach: Relationification

CHC$_f$ mod **ADT** RF → (Relationify RFs) → CHC$_F$ mod **ADT**

Encode RFs as CHCs

$$length(x) = j \longrightarrow Length(x, j)$$

$$x = nil \land i = 0 \Rightarrow Length(x, i)$$

$$Length(x, i) \land x' \neq nil \land$$
$$x = tail(x') \land i' = 1 + i \Rightarrow Length(x', i')$$

$$Length(x, i) \Rightarrow inv(x, i, i)$$

$$inv(x, i, j) \land Length(x, j) \land x \neq nil \land$$
$$x' = tail(x) \land i' = i - 1 \land Length(x', j') \Rightarrow inv(x', i', j')$$
$$inv(x, i, j) \land Length(x, j) \land i < 0 \Rightarrow \bot$$

Solution

$$inv(x, i, j) \triangleq i = j$$

$$Length(x, j) \triangleq$$

$$\left\{ \begin{array}{c} \langle nil, 0\rangle, \\ \langle cons(0, \ldots \\ \langle cons \ldots), 1\rangle, \\ \ldots, \\ \ldots 0, cons(0, nil)), 2\rangle \\ \ldots, \end{array} \right\}$$

Not expressible

UNIVERSITY OF WATERLOO

Typical approach: Relationification

CHC_f mod ADT RF —Relationify RFs→ CHC_F mod ADT

Encode RFs as CHCs

$$length(x) = j \longrightarrow Length(x, j)$$

$$x = nil \land i = 0 \Rightarrow Length(x, i)$$

$$Length(x, i) \land x' \neq nil \land$$
$$x = tail(x') \land i' = 1 + i \Rightarrow Length(x', i')$$

$$Length(x, i) \Rightarrow inv(x, i, i)$$

$$inv(x, i, j) \land Length(x, j) \land x \neq nil \land$$
$$x' = tail(x) \land i' = i - 1 \land Length(x', j') \Rightarrow inv(x', i', j')$$

$$inv(x, i, j) \land Length(x, j) \land i < 0 \Rightarrow \bot$$

Preserves sat

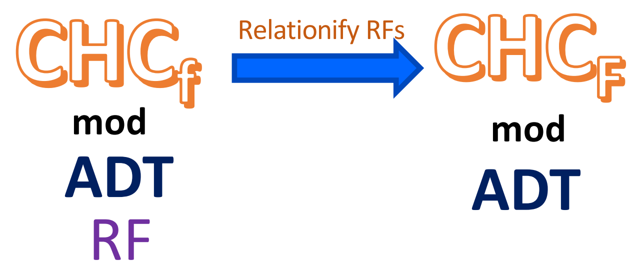
☺ Inductive summaries of RFs

☹ Sometimes no satisfying summary is expressible

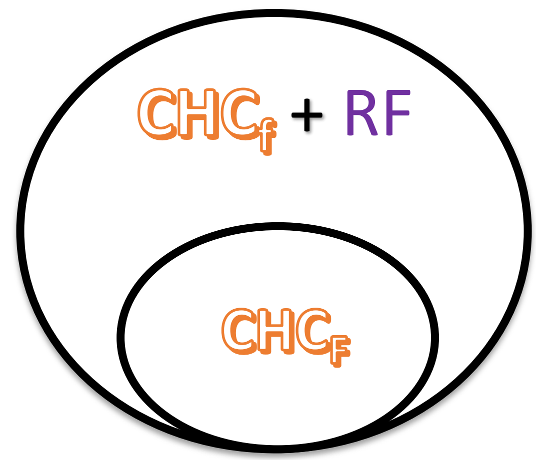
UNIVERSITY OF WATERLOO

124

Typical approach: Relationification



CONTRIBUTION 1:
Relationification preserves satisfiability but not solutions



WE NEED RFs !!!

Solving CHCs modulo ADTs and RFs

CHC_f mod **ADT** RF

Relationify and Add back RF →

CHC_{fF} mod **ADT** RF

Relationified Predicates to infer inductive summaries of RFs

\+

Original RFs to preserve solutions

→ RACER

Spacer style algorithm

Unroll and abstract RF to overcome undecidability

Always find a counter example, if one exists.

CHC$_{fF}$

Contains RFs *and* Uninterpreted Predicates

$$x = nil \land i = 0 \Rightarrow Length(x, i)$$

$$Length(x, i) \land x' \neq nil \land$$
$$x = tail(x') \land i' = 1 + i \Rightarrow Length(x', i')$$

$$Length(x, i) \Rightarrow inv(x, i, i)$$

$$inv(x, i, j) \land Length(x, j) \land x \neq nil \land$$
$$x' = tail(x) \land i' = i - 1 \land Length(x', j') \Rightarrow inv(x', i', j')$$

$$inv(x, i, j) \land Length(x, j) \land i < 0 \Rightarrow \bot$$

CHC$_{fF}$

Contains RFs *and* Uninterpreted Predicates

$$x = nil \land i = 0 \Rightarrow Length(x, i)$$

$$Length(x, i) \land x' \neq nil \land$$
$$x = tail(x') \land i' = 1 + i \Rightarrow Length(x', i')$$

$$\underline{length(x) = i} \land Length(x, i) \Rightarrow inv(x, i, i)$$

$$inv(x, i, j) \land \underline{length(x) = j} \land Length(x, j) \land x \neq nil \land$$
$$x' = tail(x) \land i' = i - 1 \land \underline{length(x') = j'} \land Length(x', j') \Rightarrow inv(x', i', j')$$

$$inv(x, i, j) \land \underline{length(x) = j} \land Length(x, j) \land i < 0 \Rightarrow \bot$$

☺ Inductive summaries of RFs

☺ All solutions are preserved

☹ 2 sources of Undecidability

RF abstraction

Unroll and replace with an *uninterpreted* function

```
length(l):
  match l
    case nil => 0
    case cons(h, t) => 1 + match t
                             case nil => 0
                             case cons(hh, tt) => 1 + length_uf(tt)
```

No Definition

Reasoning with *uninterpreted* functions is decidable

Over-approximates RF

UNIVERSITY OF WATERLOO

129

CHC_{fF} with RF abstraction

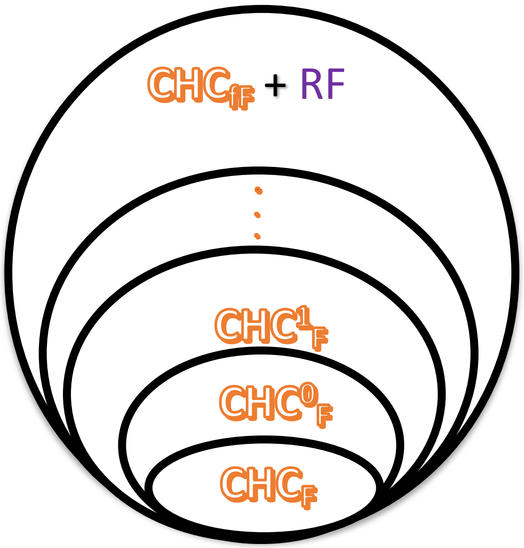
The more you unroll, the more solutions you get

No RFs after abstraction!!!!

☺ Inductive summaries of RFs

☺ More solutions than just Relationification

☺ 1 source of Undecidability



130

UF abstraction

- Remove all literals with Uninterpreted Functions
- Over-approximates RF
- Used to remove UF before quantifier elimination

Solving CHCs modulo ADTs and RFs

CHC_f
mod
ADT
RF

Relationify and
Add back RF

CHC_{fF}
mod
ADT
RF

Relationified Predicates to infer inductive summaries of RFs

+

Original RFs to preserve solutions

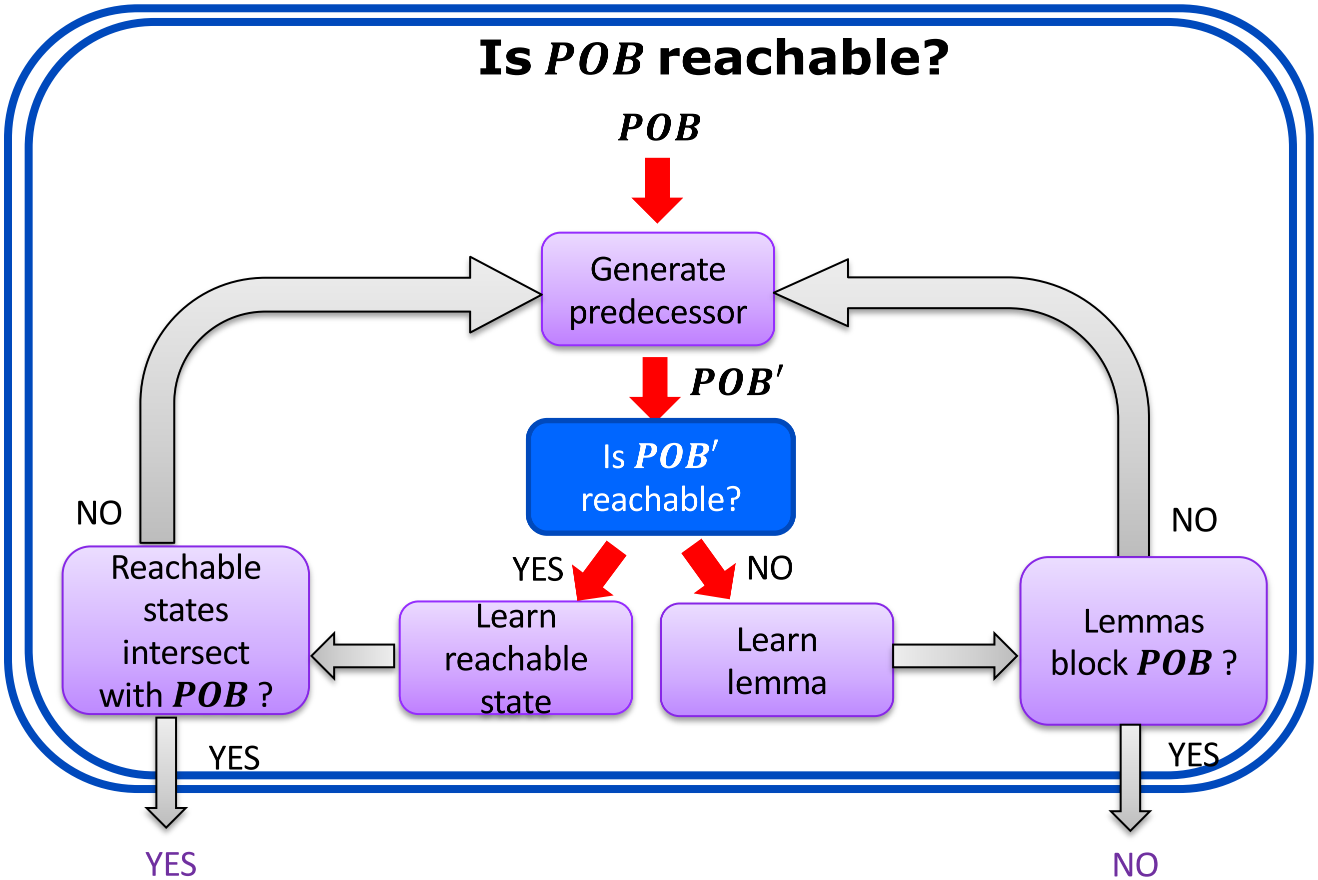
RACER

Spacer style algorithm

Unroll and abstract RF to overcome undecidability

Always find a counter example, if one exists.

RACER



UF abstraction

RF abstraction

RACER

Uses (abstractions of) RF when searching for and verifying solutions

- Periodically increases depth of unrolling

Uses Relationified predicate to produce

1. Inductive summaries of RFs
2. Counter-examples

☺ Makes no undecidable queries to SMT solver

☺ Always find a counter-example, if one exists