The Science, Art and Magic of Constrained Horn Clauses

Arie Gurfinkel and Nikolaj Bjørner

SYNASC 2019
21st International Symposium on Symbolic and Numeric Algorithms for Scientific Computing

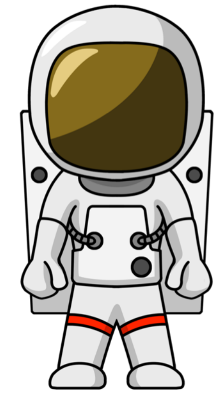
Microsoft
UNIVERSITY OF WATERLOO

https://notebooks.azure.com/arie-gurfinkel/projects/spacerexamples

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

https://notebooks.azure.com/arie-gurfinkel/projects/spacerexamples

Constrained Horn Clauses (CHC)

Alfred Horn

Is it short for Horner?

Is it related to hornets?

Is it Santa Clause blowing a Horn?

https://notebooks.azure.com/arie-gurfinkel/projects/spacerexamples

Example CHC: Is this SAT?

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

This set of clauses is satisfiable

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

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

https://notebooks.azure.com/arie-gurfinkel/projects/spacerexamples

Constrained Horn Clauses (CHC)

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

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

where

- 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

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

Example CHC: Is this SAT?

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

This set of clauses is satisfiable

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

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

https://notebooks.azure.com/arie-gurfinkel/projects/spacerexamples

Validating the solution

Original CHC

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

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

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

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

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

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

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

https://notebooks.azure.com/arie-gurfinkel/projects/spacerexamples

Example CHC: is this SAT?

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

This set of clauses is unsatisfiable

Justification is a refutation by resolution and instantiation

https://notebooks.azure.com/arie-gurfinkel/projects/spacerexamples

Example CHC: is this SAT?

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

Refutation

$$\cfrac{(x=0)\ \cfrac{\forall x \cdot x \leq 0 \implies Q(x)}{Q(0)} \qquad \forall x \cdot Q(x) \wedge x < 5 \implies Q(x+1)}{\cfrac{Q(1) \quad \forall x \cdot Q(x) \wedge x < 5 \implies Q(x+1)}{\cfrac{Q(2) \quad \forall x \cdot Q(x) \wedge x \geq 2 \implies false}{false}}}$$

https://notebooks.azure.com/arie-gurfinkel/projects/spacerexamples

Horn Clauses for Program Verification

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

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

```
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

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

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

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

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

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

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

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

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

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

$$Init(i, j, \overline{v}) \wedge Init(j, i, \overline{v}) \wedge$$
$$Init(i, i, \overline{v}) \wedge Init(j, j, \overline{v}) \Rightarrow I_2(i, j, \overline{v})$$
$$I_2(i, j, \overline{v}) \wedge Tr(i, \overline{v}, \overline{v}') \Rightarrow I_2(i, j, \overline{v}') \quad (3)$$
$$I_2(i, j, \overline{v}) \wedge Tr(j, \overline{v}, \overline{v}') \Rightarrow I_2(i, j, \overline{v}') \quad (4)$$
$$I_2(i, j, \overline{v}) \wedge I_2(i, k, \overline{v}) \wedge I_2(j, k, \overline{v}) \wedge$$
$$Tr(k, \overline{v}, \overline{v}') \wedge k \neq i \wedge k \neq j \Rightarrow I_2(i, j, \overline{v}') \quad (5)$$
$$I_2(i, j, \overline{v}) \Rightarrow \neg Bad(i, j, \overline{v})$$

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

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

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

(inductive) $\quad Inv(g, \ell_1, x_1, \ldots, \ell_i, x_i, \ldots, \ell_k, x_k) \wedge s(g, x_i, g', x'_i) \rightarrow Inv(g', \ell_1, x_1, \ldots, \ell'_i, x'_i, \ldots, \ell_k,$

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

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

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

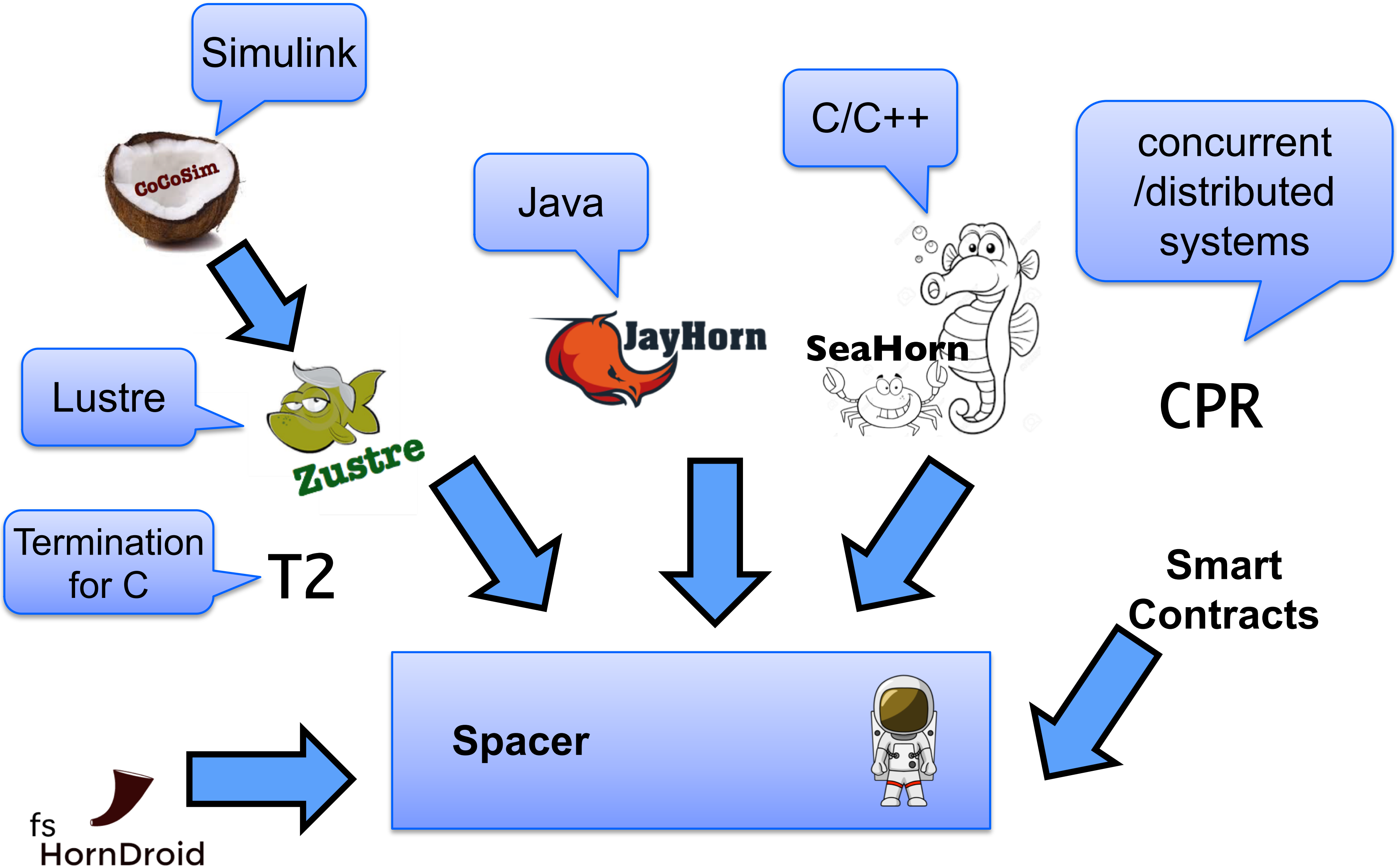
```
(set-logic HORN)

;; 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
```

UNIVERSITY OF WATERLOO

Logic-based Algorithmic Verification



https://notebooks.azure.com/arie-gurfinkel/projects/spacerexamples

INTERACTIVE TUTORIAL

https://notebooks.azure.com/arie-gurfinkel/projects/spacerexamples

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, …)

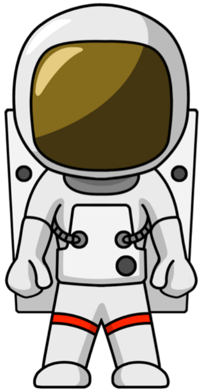
Interpolation-based Model Checking

- Duality, QARMC, …

SMT-based Unbounded Model Checking (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
- 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.

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

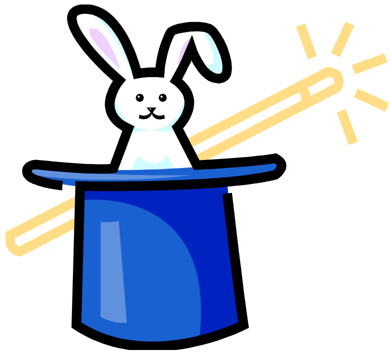
Satisfiability of Linear CHC over Propositional logic is decidable

- 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)



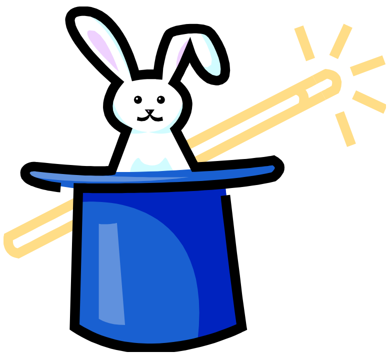
SOLVING CONSTRAINED HORN CLAUSES

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

Linear CHC Satisfiability

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

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

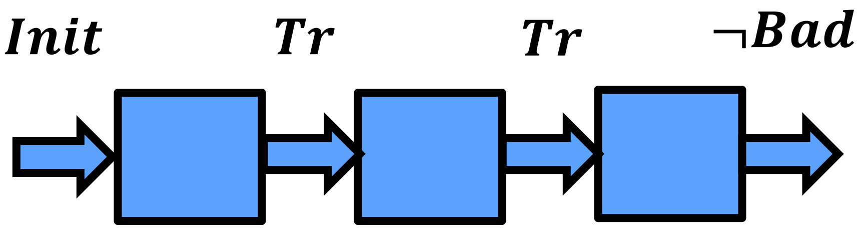
where, X' = {x' | x $\in$ X}, P a fresh predicate, and *Init*, *Bad*, and *Tr* are constraints

Proof:

add extra arguments to distinguish between predicates

$$\frac{Q(y) \wedge \phi \rightarrow W(y, z)}{P(id='Q', y) \wedge \phi \rightarrow P(id='W', y, z)}$$

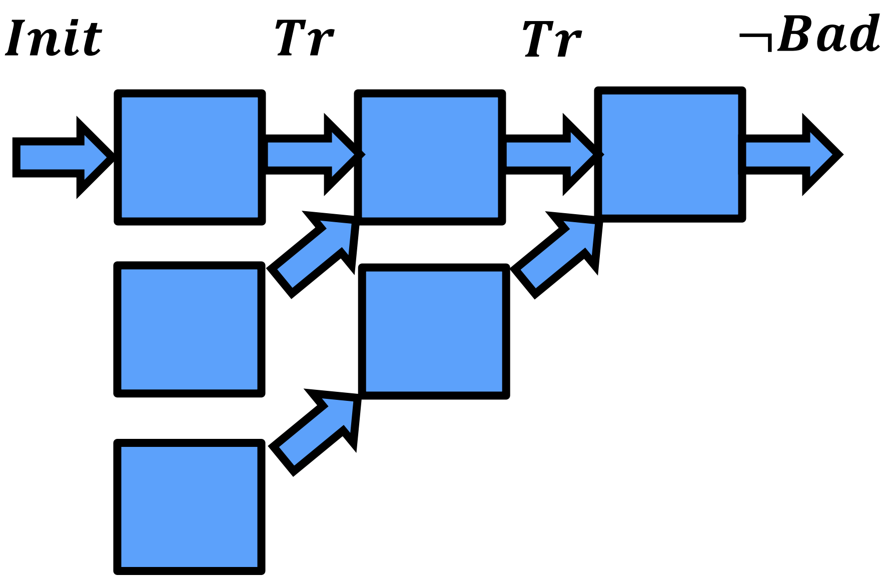
IC3, PDR and friends



Finite State Machines

(HW model checking)

[Bradley, VMCAI 2011]

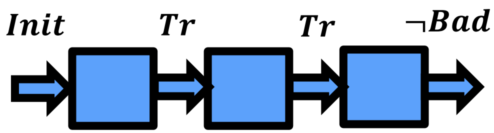


Push Down Machines

(SW model checking)

[Hoder&B, SAT 2012]

IC3, PDR and friends



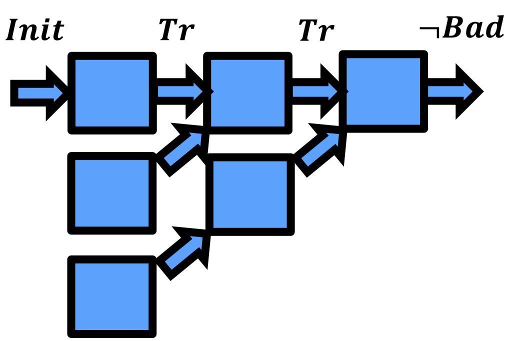
Finite State Machines

(HW model checking)

[Bradley, VMCAI 2011]

Finite State

- Incremental SAT solving [Bradley, VMCAI 11]
- Fast prime implicants [Een& FMCAD 11]
- Basis for predicate abstraction [Cimatti& TACAS 14, Birgmeier& CAV 14]



Push Down Machines

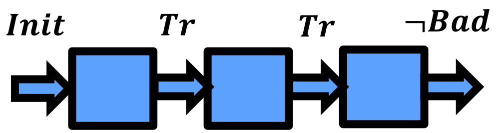
(SW model checking)

[Hoder, B, SAT 2012]

Infinite State

- Arithmetic + Farkas [H&B, SAT 12]
- Arithmetic + Model Based Projection [K&, CAV 14]
- Polyhedra + Convex Closure [B&G, VMCAI 15]
- Arithmetic + Arrays [K&, FMCAD 15]
- ∃∀ - EPR fragment [K'&, CAV 15]
- ∃∀ + Arithmetic/Arrays [G&, ATVA 18]

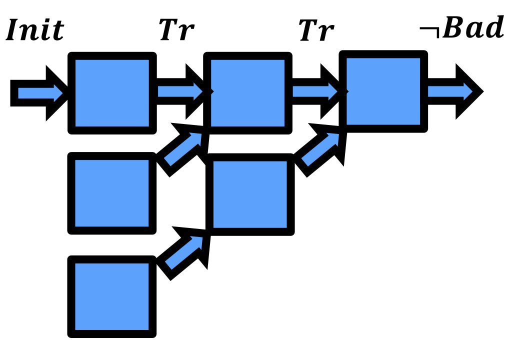
IC3, PDR and friends



Finite State Machines

(HW model checking)

[Bradley, VMCAI 2011]



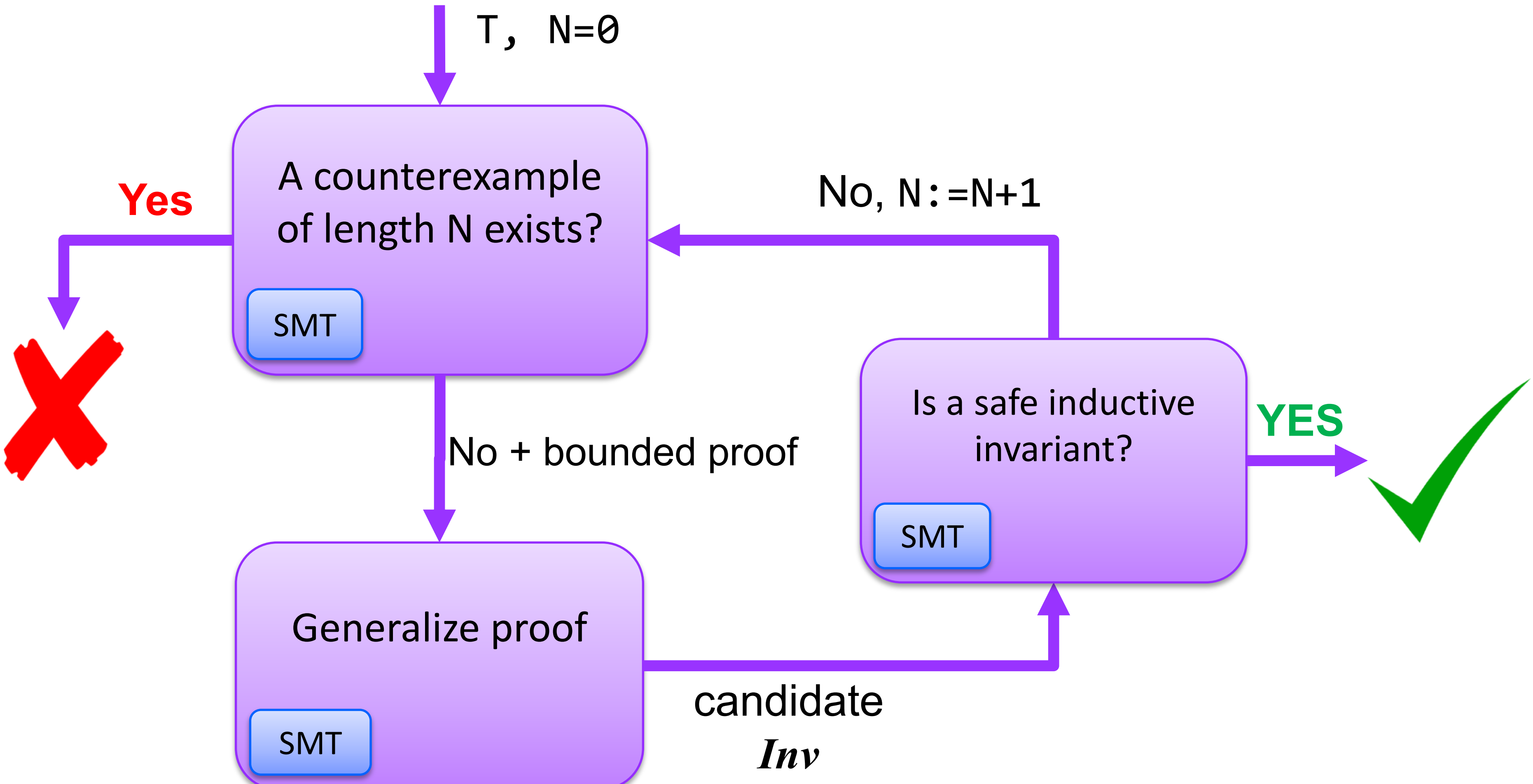
Push Down Machines

(SW model checking)

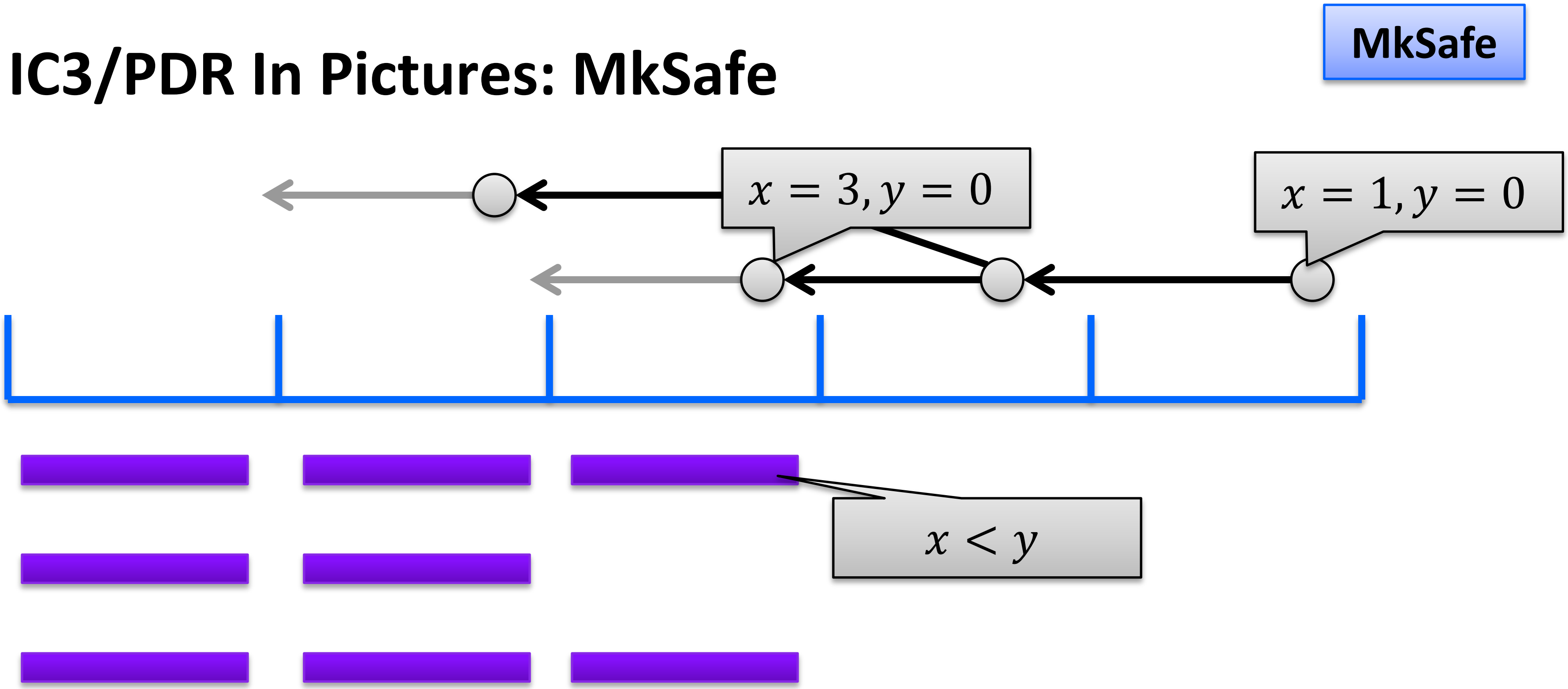
[Hoder, B, SAT 2012]

Finite State	Search Strategies
SAT	[Bradley, VMCAI 11] CTI – Counter Examples To Induction
Infinite State	[G&Ivrii, FMCAD 15] Under and over-approximations
SMT Arithmetic Arrays Quantifiers	[Vizel&G, CAV 14] Use SAT for blocking IC3 for pushing

Verification by Incremental Generalization



IC3/PDR In Pictures: MkSafe



Predecessor

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

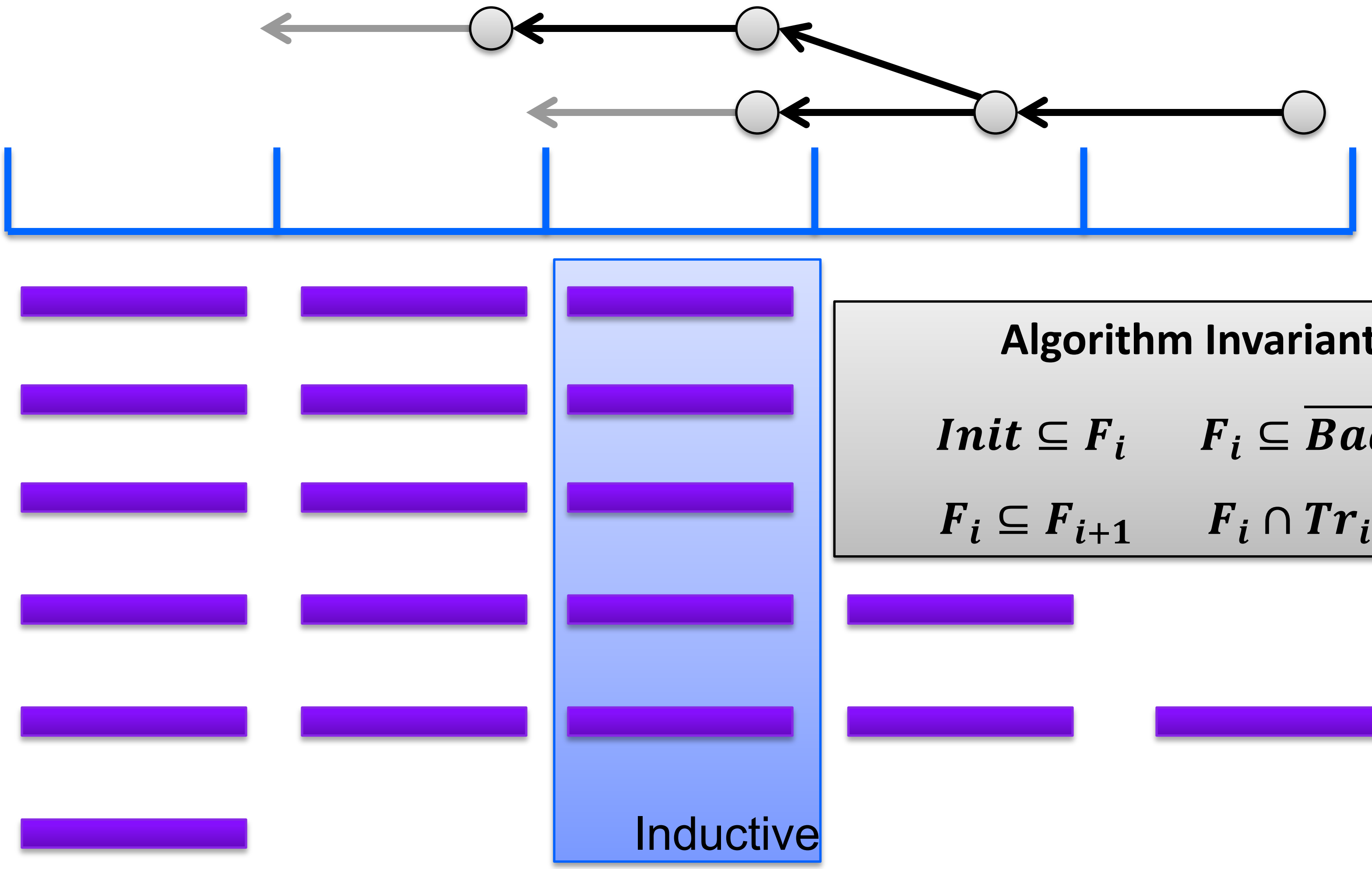
$$\text{find } m \text{ s.t. } (M \models m) \wedge (m \implies \exists V' \cdot Tr \wedge m')$$

NewLemma

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

IC3/PDR in Pictures: Push

Push



Algorithm Invariants

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

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

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

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

Decide

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

Conflict

- construct a lemma to explain why cex cannot be extended
- Find a clause ***L*** s.t. $L \Rightarrow \neg cex$, $Init \Rightarrow L$, 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

From Propositional PDR to Solving CHC

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 predecessors (when computing predecessors)

The problem is undecidable in general, but we 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

Decide

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

Conflict

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

Theory dependent

Induction

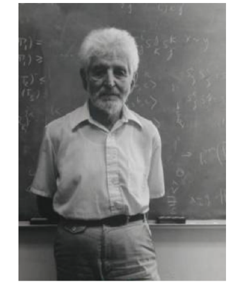
- propagate a lemma as far into the future as possible
- (optionally) strengthen by dropping literals

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

Looking for φ’

CONFLICT (ARITHMETIC)

Craig Interpolation Theorem



Theorem (Craig 1957)
Let A and B be two First Order (FO) formulae such that $A \Rightarrow \neg B$, then there exists a FO formula I, denoted ITP(A, B), such that

$$A \Rightarrow I \qquad I \Rightarrow \neg B \qquad \Sigma(I) \in \Sigma(A) \cap \Sigma(B)$$

A Craig interpolant ITP(A, B) can be effectively constructed from a resolution proof of unsatisfiability of $A \wedge B$

In Model Checking, Craig Interpolation Theorem is used to safely over-approximate the set of (finitely) reachable states

UNIVERSITY OF WATERLOO

Examples of Craig Interpolation for Theories

Boolean logic

$$A = (\neg b \wedge (\neg a \vee b \vee c) \wedge a) \qquad B = (\neg a \vee \neg c)$$

$$ITP(A, B) = a \wedge c$$

Equality with Uniterpreted Functions (EUF)

$$A = (f(a) = b \wedge p(f(a))) \qquad B = (b = c \wedge \neg p(c))$$

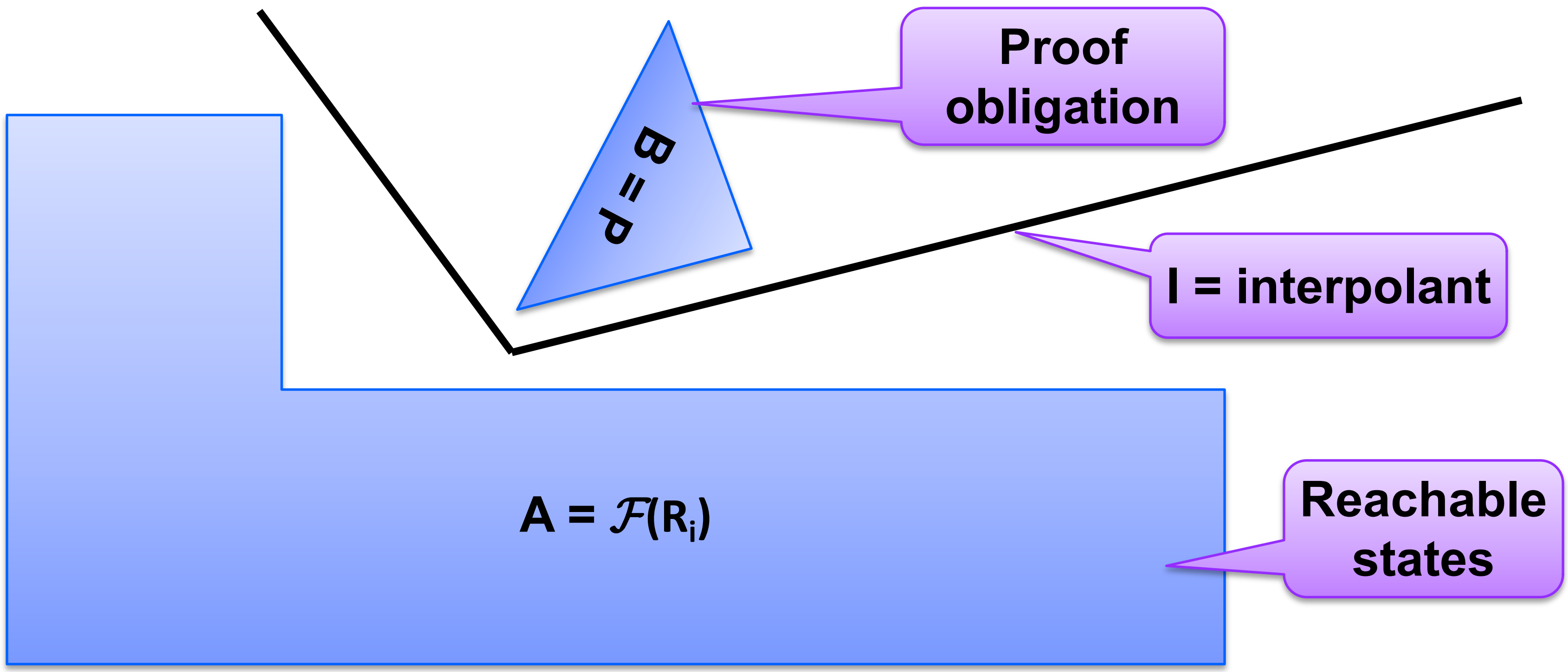
$$ITP(A, B) = p(b)$$

Linear Real Arithmetic (LRA)

$$A = (z + x + y > 10 \wedge z < 5) \qquad B = (x < -5 \wedge y < -3)$$

$$ITP(A, B) = x + y > 5$$

Craig Interpolation for Linear Arithmetic



Useful properties of existing interpolation algorithms [CGS10] [HB12]

- $I \in ITP(A, B)$ then $\neg I \in ITP(B, A)$
- if A is syntactically convex (a monomial), then I is convex
- if B is syntactically convex, then I is co-convex (a clause)
- if A and B are syntactically convex, then I is a half-space

Arithmetic Conflict

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

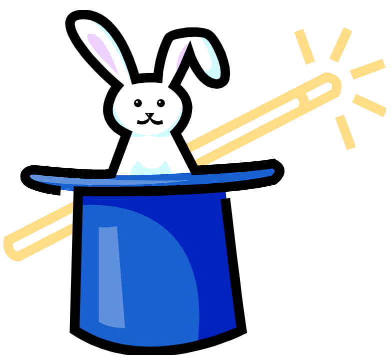
Conflict For $0 \leq i < N$, given a counterexample $\langle P, i+1 \rangle \in Q$ s.t. $\mathcal{F}(F_i) \wedge P'$ is unsatisfiable, add $P^{\uparrow} = \textsc{Itp}(\mathcal{F}(F_i), P')$ to F_j for $j \leq i+1$.

Counterexample is blocked using Craig Interpolation

- summarizes the reason why the counterexample cannot be extended

Generalization is not inductive

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



Computing Interpolants for IC3/PDR

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

- B is always a conjunction of literals
- A is dynamically split into DNF by the SMT solver
- DPLL(T) proofs do not introduce new literals

Interpolation algorithm is reduced to analyzing all theory lemmas in a DPLL(T) proof produced by the 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 PDR
- classical interpolation focuses on eliminating non-shared literals
- in PDR, the focus is on finding good generalizations

Farkas Lemma

Let $\Phi = t_1 \geq b_1 \wedge \ldots \wedge t_n \geq b_n$, t_i are linear terms and b_i are constants

Φ is *unsatisfiable* iff $0 \geq 1$ is derivable from Φ by resolution

- $x + 2y > 10,$
- $-x > 5,$
- $-y > 3$
- $0 = (x + 2y - x - 2y) > (10 + 5 + 2 \cdot 3) > 21$

Proof uses *Farkas* coefficients $g_1, \ldots, g_n$ such that

- $g_i > 0$
- $g_1 \cdot t_1 + \ldots + g_n \cdot t_n = 0$
- $g_1 \cdot b_1 + \ldots + g_n \cdot bn > 1$

UNIVERSITY OF WATERLOO

Frakas Lemma Example

Interpolants

$$z + x + y > 10 \qquad \times 1$$
$$-z > -5 \qquad \times 1$$

$$\Bigg\} \quad x + y > 5$$

$$-x > 5 \qquad \times 1$$
$$-y > 3 \qquad \times 1$$

$$\Bigg\} \quad x + y < -8$$

$$0 > 13$$

UNIVERSITY OF WATERLOO

Interpolation for Linear Real Arithmetic

Let A ∧ B be UNSAT, where

- $A = t_1 \geq b_1 \wedge \ldots \wedge t_i \geq bi$, and
- $B = t_{i+1} \geq bi \wedge \ldots \wedge t_n \geq b_n$

Let $g_1, \ldots, g_n$ be the Farkas coefficients witnessing UNSAT

Then

- $g_1 \cdot (t_1 - b_1) + \ldots + g_i \cdot (t_i - b_i) \geq 0$ is an interpolant between A and B
- $g_{i+1} \cdot (t_{i+1} - b_{i+1}) + \ldots + g_n \cdot (t_n - b_n) \geq 0$ is an interpolant between B and A

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);
```

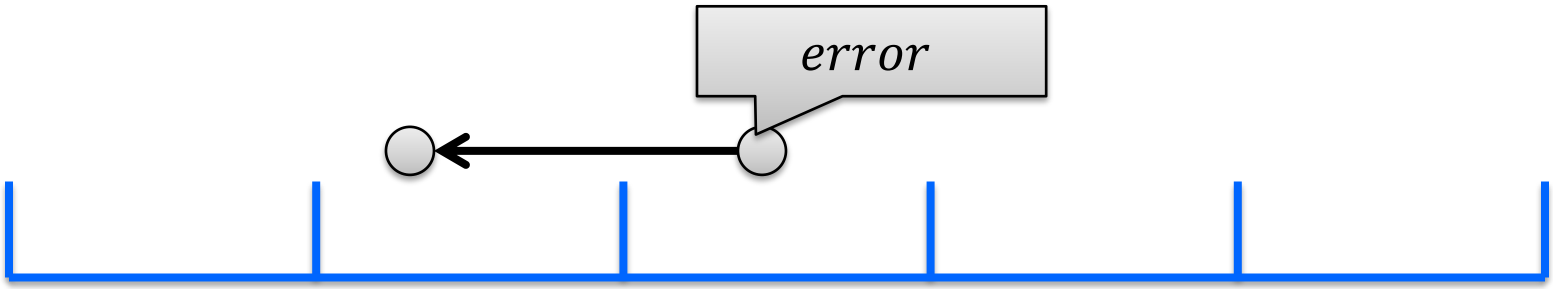
IS SAT?

```
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
```

UNIVERSITY OF WATERLOO

Lemma Generation Example

MkSafe



Transition Relation

$x = x_0 \wedge z = z_0+1 \wedge i=i_0+1 \wedge y > i_0$

Pob

$i >= y \wedge x + y > z$

Farkas explanation for unsat

$$\frac{\dfrac{x_0 + y_0 <= z_0,\ \ x <= x_0,\ z_0 < z,\ i <= i_0 + 1}{x + i <= z} \qquad \dfrac{i >= y,\ x+y > z}{x + i > z}}{\text{false}}$$

Learn lemma: $x + i <= z$

Interpolation Problem in Spacer

Given an arbitrary LRA formula A and a conjunction of literals s such that $A \wedge s$ are UNSAT, compute an interpolant I such that

- $s \Rightarrow I$ $\quad$ $I \wedge A \Rightarrow FALSE$ $\quad$ I is over symbols common to s and A

Use an SMT solver to decide that $s \wedge A$ are UNSAT

- SMT solver uses LRA theory lemmas (called Farkas Theory Lemmas) of the form:

  $\neg ((s_1 \wedge \ldots \wedge s_k) \wedge (a_1 \wedge \ldots \wedge a_m))$

  where s_i are literals from s and a_i are literals from A
- For each such lemma L_j, $((s_1 \wedge \ldots \wedge s_k) \wedge (a_1 \wedge \ldots \wedge a_m)$ is UNSAT
- Let t_j be an interpolant corresponding to L_j

Then, an interpolant between s and A is a clause of the form

$(\neg t_1 \vee \ldots \vee \neg t_k)$ with one literal per each theory lemma

- in practice, interpolation is optimized by examining and restructuring SMT resolution proof, dealing with Boolean reasoning, and global optimization

UNIVERSITY OF WATERLOO

Computing Interpolants in Spacer

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

- B is always a conjunction of literals
- A is dynamically split into DNF by the SMT solver
- DPLL(T) proofs do not introduce new literals

Interpolation algorithm is reduced to analyzing all theory lemmas in a DPLL(T) proof produced by the 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 PDR
- classical interpolation focuses on eliminating non-shared literals
- in PDR, the focus is on finding good generalizations

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

$$\equiv$$

$$\boldsymbol{s \Rightarrow \exists X'.\, Tr(X, X') \land cex(X')}$$

Computing a predecessor ***s*** of a counterexample ***cex***

DECIDE (ARITHMETIC)

UNIVERSITY OF WATERLOO

Model Based Projection

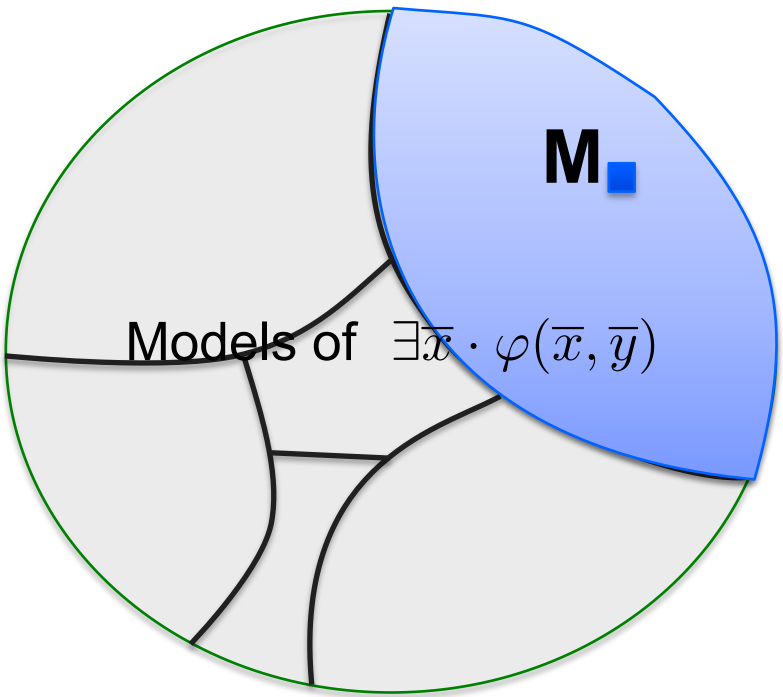
Definition: Let φ be a formula, X a set of variables, and M a model of φ. Then $\psi = MBP\ (X, M, \varphi)$ is a Model Based Projection of X, M, φ iff

1. ψ *is a monomial*
2. $Vars(\psi) \subseteq Vars(\varphi) \setminus X$
3. $M \models \psi$
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 partition containing M

Quantifier Elimination

Quantifier elimination procedure:

- **Input**: formula $\exists\, x\, \psi(x)$
- **Output**: equivalent φ without existential quantifier. x is eliminated.
- $\mathrm{QELIM}(\exists\, x\, \psi(x)) = \varphi$ and $\exists\, x\, \psi(x) \iff \varphi$

Quantifier elimination in propositional logic

- $\mathrm{QELIM}(\exists\, x\, \psi(x)) = \psi(\mathrm{TRUE}) \vee \psi(\mathrm{FALSE})$

Many theories support quantifier elimination (e.g., linear arithmetic)

- but not all. No quantifier elimination for EUF,
  - e.g., $(\exists x\, f(x) \neq g(x))$ cannot be expressed without the existential quantifier

Quantifier elimination is usually expensive

- e.g., propositional QELIM is exponential in the number of variables quantified

Loos-Weispfenning Quantifier Elimination for LRA

ϕ is LRA formula in Negation Normal Form

E is set of x=t atoms, U set of x < t atoms, and L set of s < x atoms

There are no other occurrences of x in ϕ[x]

$$\exists x . \varphi[x] \equiv \varphi[\infty] \vee \bigvee_{x=t \in E} \varphi[t] \vee \bigvee_{x<t \in U} \varphi[t - \epsilon]$$

where

$$(x < t')[t - \epsilon] \equiv t \leq t' \qquad (s < x)[t - \epsilon] \equiv s < t \qquad (x = e)[t - \epsilon] \equiv \mathit{false}$$

The case of lower bounds is dual

- using –∞ and t+ϵ

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

UNIVERSITY OF WATERLOO

Quantifier Elimination with Assumptions

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

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 an the assumptions?!

- MBP == use the order chosen by the model

UNIVERSITY OF WATERLOO

MBP for Linear Rational Arithmetic

Compute a **single** disjunct from LW-QE that includes the model

- Use the Model to uniquely pick a substitution term for x

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

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

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

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

MBP techniques have been developed for

- Linear Rational Arithmetic, Linear Integer Arithmetic
- Theories of Arrays, and Recursive Data Types

UNIVERSITY OF WATERLOO

Arithmetic Decide

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

Decide 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 $P_\downarrow = \text{MBP}(X', m, \mathcal{F}(F_i) \wedge P')$.

Compute a predecessor using Model Based Projection

To ensure progress, Decide must be finite

- finitely many possible predecessors when all other arguments are fixed

Alternatively

- Completeness can follow from an interaction of **Decide** and **Conflict**
  - but requires more rules to propagate implicants backward (as in PDR) and forward (as in Spacer and Quip)

PolyPDR: 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

Decide

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

Conflict

- construct a lemma to explain why cex cannot be extended
- Find an interpolant ***L*** s.t. $L \Rightarrow \neg cex$, $Init \Rightarrow L$, 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

$$Init(X) \to P(X)$$
$$P(X) \land P(X^o) \land Tr(X, X^o, X') \to P(X')$$
$$P(X) \to \neg Bad(X)$$

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

UNIVERSITY OF WATERLOO

Generalized GPDR

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 $\langle c_0, \ldots, c_k\rangle \in Q$ is a tuple, each $c_j = \langle m, i\rangle$, m is a cube over state variables, and $i \in \mathbb{N}$. A level N. 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: $Q = \emptyset$, $N = 0$, $F_0 = Init$, $\forall i > 0 \cdot F_i = \emptyset$
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 exists $t \in Q$ s.t. for all $\langle c, i\rangle \in t$, $i = 0$, **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 $\langle\langle m, N\rangle\rangle$ to Q.
- **Decide** If there is a $t \in Q$, with $c = \langle m, i+1\rangle \in t$, $m_1 \to m$, $l_0 \wedge m_0^o \wedge m_1'$ is satisfiable, and $l_0 \wedge m_0^o \wedge m_1' \to F_i \wedge F_i^o \wedge Tr \wedge m'$ then add $\hat{t}$ to Q, where $\hat{t} = t$ with c replaced by two tuples $\langle l_0, i\rangle$, and $\langle m_0, i\rangle$.
- **Conflict** If there is a $t \in Q$ with $c = \langle m, i+1\rangle \in t$, s.t. $\mathcal{F}(F_i) \wedge m'$ is unsatisfiable. Then, add $\varphi = \textsc{Itp}(\mathcal{F}(F_i), m')$ to F_j, for all $0 \le j \le i+1$.
- **Leaf** If there is $t \in Q$ with $c = \langle m, i\rangle \in t$, $0 < i < N$ and $\mathcal{F}(F_{i-1}) \wedge m'$ is unsatisfiable, then add $\hat{t}$ to Q, where $\hat{t}$ is t with c replaced by $\langle m, i+1\rangle$.
- **Induction** For $0 \le i < N$ and a clause $(\varphi \vee \psi) \in F_i$, if $\varphi \notin F_{i+1}$, $\mathcal{F}(\phi \wedge F_i) \to \phi'$, then add φ to F_j, for all $j \le i+1$.

until ∞;

counterexample is a tree

two predecessors

theory-aware **Conflict**

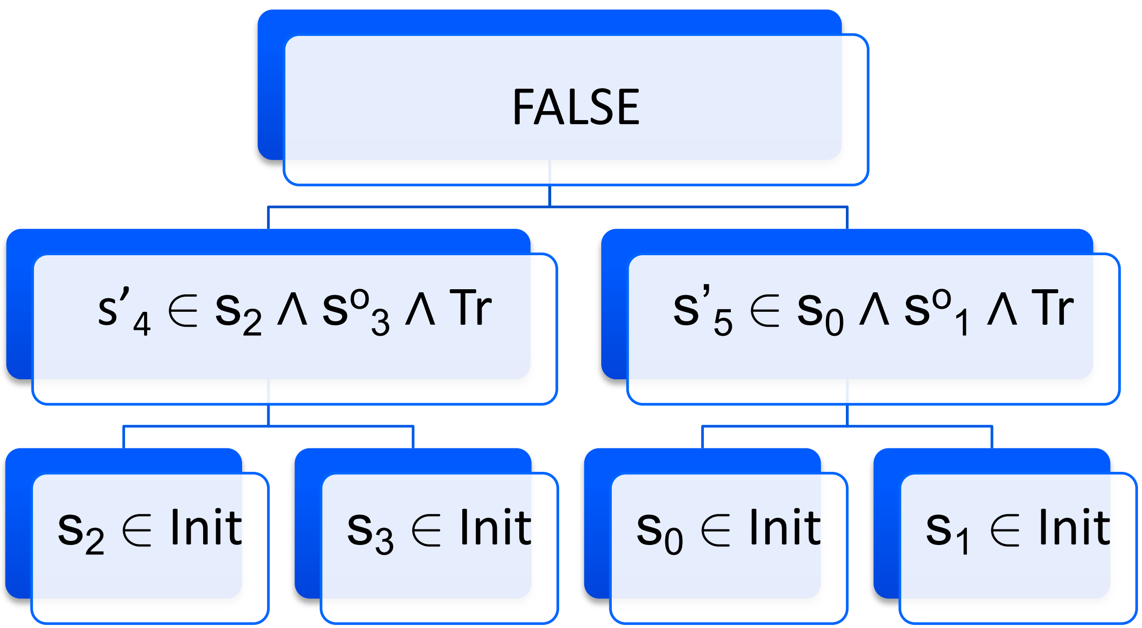
Counterexamples to non-linear CHC

A set S of CHC is unsatisfiable iff S can derive FALSE

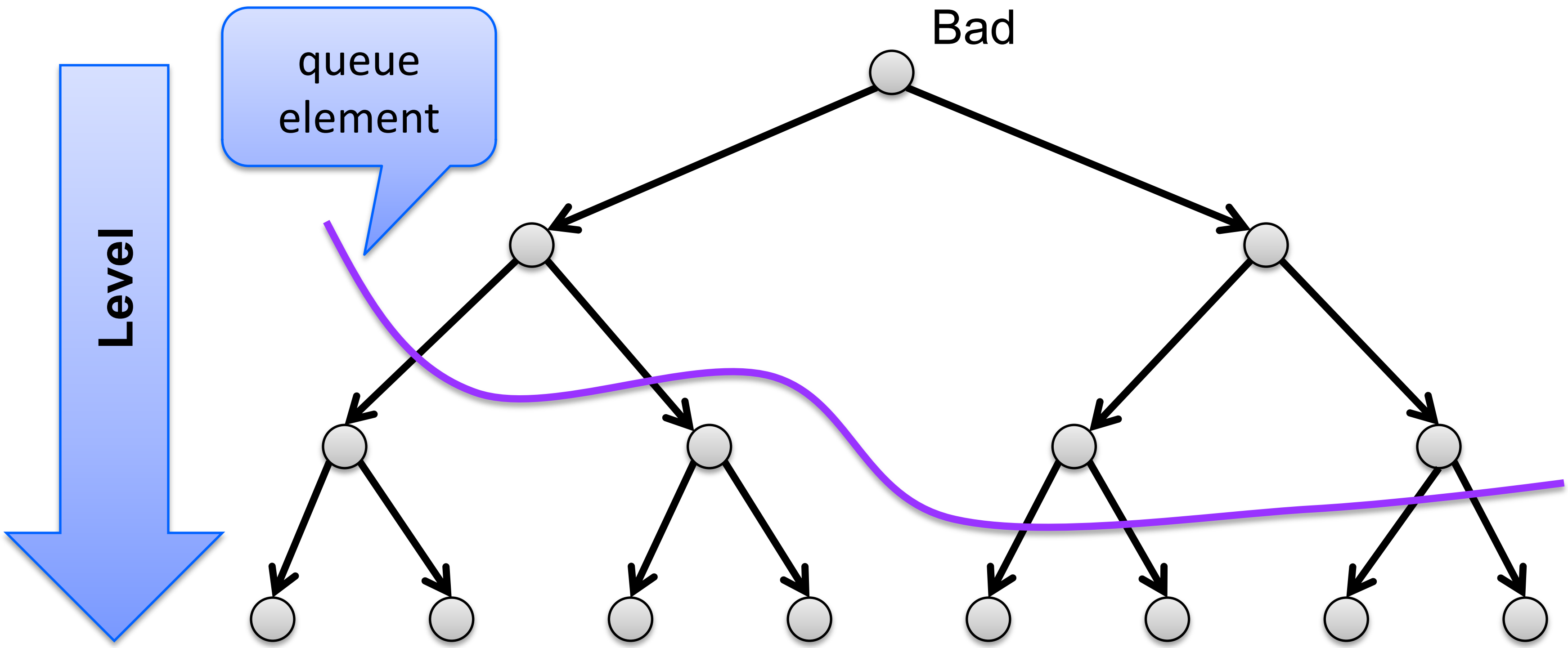
- we call such a derivation a counterexample

For linear CHC, the counterexample is a path

For non-linear CHC, the counterexample is a tree



GPDR Search Space



In Decide, one POB in the frontier is chosen and its two children are expanded

GPDR: Splitting predecessors

Consider a clause

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

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

Predecessor over the constraint is:

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

Need to create two separate proof obligation

- one for P(x) and one for P(y)
- gpdr solution: split by substituting values from the model (incomplete)

UNIVERSITY OF WATERLOO

GPDR: Deciding predecessors

Decide If there is a $t \in Q$, with $c = \langle m, i+1 \rangle \in t$, $m_1 \to m$, $l_0 \wedge m_0^o \wedge m_1'$ is satisfiable, and $l_0 \wedge m_0^o \wedge m_1' \to F_i \wedge F_i^o \wedge Tr \wedge m'$ then add $\hat{t}$ to Q, where $\hat{t} = t$ with c replaced by two tuples $\langle l_0, i \rangle$, and $\langle m_0, i \rangle$.

Compute two predecessors at each application of **GPDR/Decide**

Can explore both predecessors in parallel

- e.g., BFS or DFS exploration order

Number of predecessors is unbounded

- incomplete even for finite problem (i.e., non-recursive CHC)

No caching/summarization of previous decisions

- worst-case exponential for Boolean Push-Down Systems

Spacer

Same queue as in IC3/PDR

Cache Reachable states

Three variants of **Decide**

Same **Conflict** as in APDR/GPDR

Input: A safety problem $\langle \mathit{Init}(X), \mathit{Tr}(X, X^o, X'), \mathit{Bad}(X) \rangle$.
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 $\textsc{Reach}$. A trace $F_0, F_1, \ldots$
Notation: $\mathcal{F}(A, B) = \mathit{Init}(X') \vee (A(X) \wedge B(X^o) \wedge \mathit{Tr})$, and $\mathcal{F}(A) = \mathcal{F}(A, A)$
Initially: $Q = \emptyset$, $N = 0$, $F_0 = \mathit{Init}$, $\forall i > 0 \cdot F_i = \emptyset$, $\textsc{Reach} = \mathit{Init}$
Require: $\mathit{Init} \to \neg \mathit{Bad}$
repeat

Unreachable If there is an $i < N$ s.t. $F_i \subseteq F_{i+1}$ **return** *Unreachable*.

Reachable If $\textsc{Reach} \wedge \mathit{Bad}$ is satisfiable, **return** *Reachable*.

Unfold If $F_N \to \neg \mathit{Bad}$, then set $N \leftarrow N + 1$ and $Q \leftarrow \emptyset$.

Candidate If for some m, $m \to F_N \wedge \mathit{Bad}$, then add $\langle m, N \rangle$ to Q.

Successor If there is $\langle m, i+1 \rangle \in Q$ and a model M $M \models \psi$, where $\psi = \mathcal{F}(\vee \textsc{Reach}) \wedge m'$. Then, add s to $\textsc{Reach}$, where $s' \in \text{MBP}(\{X, X^o\}, \psi)$.

DecideMust If there is $\langle m, i+1 \rangle \in Q$, and a model M $M \models \psi$, where $\psi = \mathcal{F}(F_i, \vee \textsc{Reach}) \wedge m'$. Then, add s to Q, where $s \in \text{MBP}(\{X^o, X'\}, \psi)$.

DecideMay If there is $\langle m, i+1 \rangle \in Q$ and a model M $M \models \psi$, where $\psi = \mathcal{F}(F_i) \wedge m'$. Then, add s to Q, where $s^o \in \text{MBP}(\{X, X'\}, \psi)$.

Conflict If there is an $\langle m, i+1 \rangle \in Q$, s.t. $\mathcal{F}(F_i) \wedge m'$ is unsatisfiable. Then, add $\varphi = \textsc{Itp}(\mathcal{F}(F_i), m')$ to F_j, for all $0 \leq j \leq i + 1$.

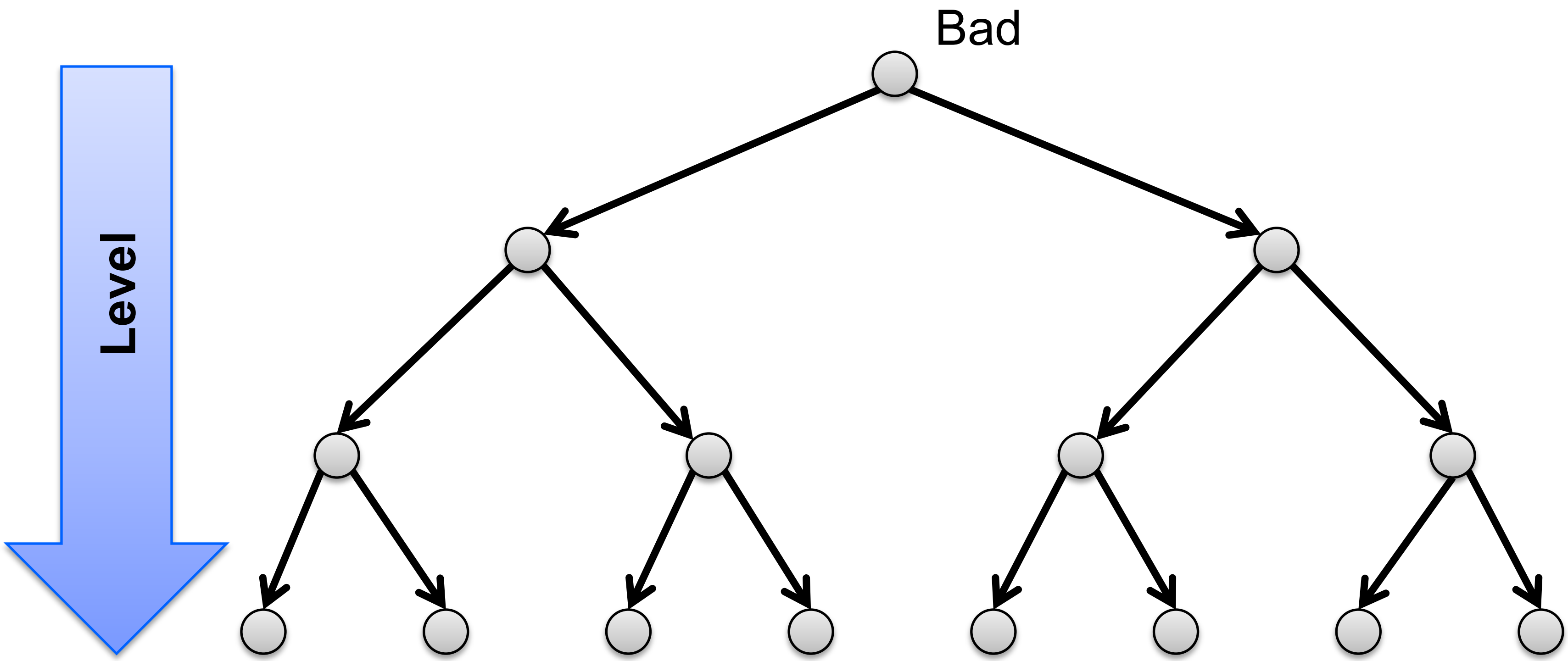
Leaf If $\langle m, i \rangle \in Q$, $0 < i < N$ and $\mathcal{F}(F_{i-1}) \wedge m'$ is unsatisfiable, then add $\langle m, i+1 \rangle$ to Q.

Induction For $0 \leq i < N$ and a clause $(\varphi \vee \psi) \in F_i$, if $\varphi \notin F_{i+1}$, $\mathcal{F}(\phi \wedge F_i) \to \phi'$, then add φ to F_j, for all $j \leq i + 1$.

until ∞;

UNIVERSITY OF WATERLOO

SPACER Search Space



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

- use MBP to decide on counterexamples

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

- use MBP to propagate facts bottom up

Successor Rule: Computing Reachable States

Successor If there is $\langle m, i+1\rangle \in Q$ and a model M $M \models \psi$, where $\psi = \mathcal{F}(\vee \textsc{Reach}) \wedge m'$. Then, add s to $\textsc{Reach}$, where $s' \in \text{MBP}(\{X, X^o\}, \psi)$.

Computing new reachable states by under-approximating forward image using MBP

- since MBP is finite, guarantee to exhaust all reachable states

Second use of MBP

- orthogonal to the use of MBP in Decide
- can allow REACH to contain auxiliary variables, but this might explode

For Boolean CHC, the number of reachable states is bounded

- complexity is polynomial in the number of states
- same as reachability in Push Down Systems

Decide Rule: Must and May refinement

DecideMust If there is $\langle m, i+1 \rangle \in Q$, and a model M $M \models \psi$, where $\psi = \mathcal{F}(F_i, \vee \textsc{Reach}) \wedge m'$. Then, add s to Q, where $s \in \text{MBP}(\{X^o, X'\}, \psi)$.

DecideMay If there is $\langle m, i+1 \rangle \in Q$ and a model M $M \models \psi$, where $\psi = \mathcal{F}(F_i) \wedge m'$. Then, add s to Q, where $s^o \in \text{MBP}(\{X, X'\}, \psi)$.

DecideMust

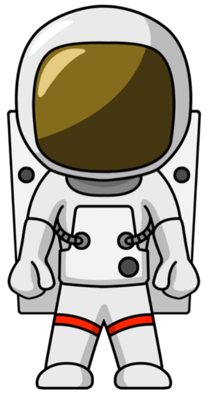
- use computed summary (REACH) to skip over a call site

DecideMay

- use over-approximation of a calling context to guess an approximation of the call-site
- the call-site either refutes the approximation (**Conflict**) or refines it with a witness (**Successor**)

UNIVERSITY OF WATERLOO

Art, Science, and Magic



Verification of Safety Properties is FOL satisfiability

- Logic: Constrained Horn Clauses (CHC)
- “Decision” procedure: Spacer
- Now with (universal) quantifiers!

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

UNIVERSITY OF WATERLOO

THE END