Constrained Horn Clauses (CHC) over Linear Integer Arithmetic (LIA)

Automated Program Verification (APV)
Fall 2018

Prof. Arie Gurfinkel

UNIVERSITY OF
WATERLOO

CONSTRAINED HORN CLAUSES

Precise Logic-based Program Verification

Low-Level Bounded Model Checking (BMC)

- decide whether a low level program/circuit has an execution of a given length that violates a safety property
- effective decision procedure via encoding to propositional SAT

High-Level (Word-Level) Bounded Model Checking

- decide whether a program has an execution of a given length that violates a safety property
- efficient decision procedure via encoding to SMT

What is an SMT-like equivalent for Safety Verification?

- Logic: SMT-Constrained Horn Clauses
- Decision Procedure: Spacer / GPDR
  - extend IC3/PDR algorithms from Hardware Model Checking

Constrained Horn Clauses (CHCs)

A Constrained Horn Clause (CHC) is a FOL formula

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

where

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

CHC Satisfiability

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

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

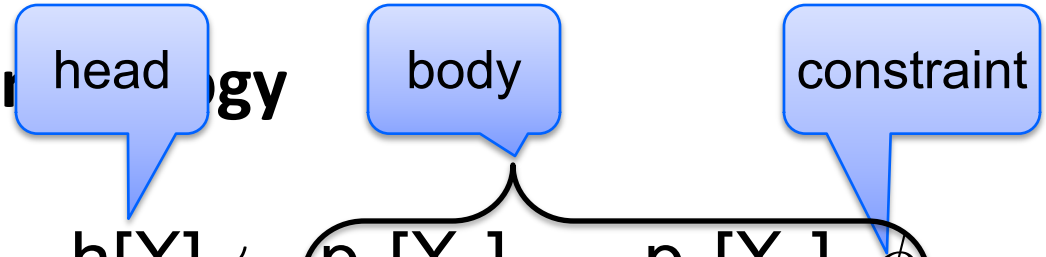
- This is the usual FOL satisfiability

A $\mathcal{T}$**-solution** of a set of CHCs 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

- a program satisfies a property iff corresponding CHCs are satisfiable
- solutions are inductive invariants
- refutation proofs are counterexample traces

CHC Notation and Terminology



Rule

$$h[X] \leftarrow p_1[X_1], \ldots, p_n[X_n], \phi.$$

Query

$$\text{false} \leftarrow p_1[X_1], \ldots, p_n[X_n], \phi.$$

Fact

$$h[X] \leftarrow \phi.$$

Linear CHC

$$h[X] \leftarrow p[X_1], \phi.$$

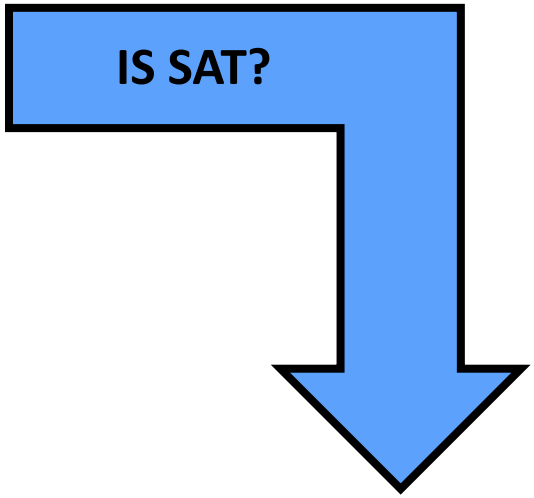
Non-Linear CHC

$$h[X] \leftarrow p_1[X_1], \ldots, p_n[X_n], \phi.$$

for $n > 1$

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);
```




```
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)))
)
```

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

$$z = x + i$$

$$z <= x + y$$

Horn Clauses for Program Verification

with the edges are formulated as follows:

$$p_{init}(x_0, \boldsymbol{w}, \bot) \leftarrow x = x_0 \qquad \text{where } x \text{ occurs in } \boldsymbol{w}$$
$$p_{exit}(x_0, ret, \top) \leftarrow \ell(x_0, \boldsymbol{w}, \top) \qquad \text{for each label } \ell, \text{ and } re$$
$$p(x, ret, \bot, \bot) \leftarrow p_{exit}(x, ret, \bot)$$
$$p(x, ret, \bot, \top) \leftarrow p_{exit}(x, ret, \top)$$

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

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

```
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.
```

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

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

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

Bjørner, Gurfinkel, McMillan, and Rybalchenko:

Horn Clause Solvers for Program Verification

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.

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

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

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

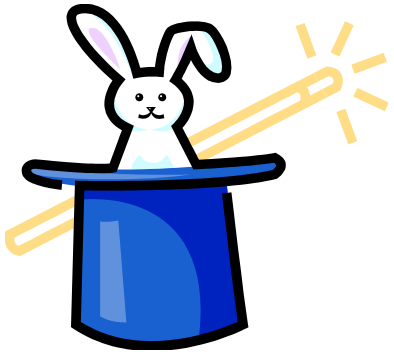
(safe) $\quad \mathit{Inv}(g, \ell_1, x_1, \ldots, \ell_k, x_k) \wedge \mathsf{err}(g, \ell_1, x_1, \ldots, \ell_m, x_m) \rightarrow \mathit{false}$

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 an
from ℓ_i to ℓ'_i and, respectively, from $\ell^\dagger$ to some other location in the control flow graph)

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

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

UNIVERSITY OF WATERLOO



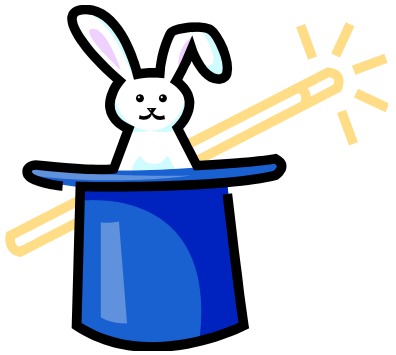
SOLVING CONSTRAINED HORN CLAUSES

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

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

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 (1)

IC3: A SAT-based Hardware Model Checker

- Incremental Construction of Inductive Clauses for Indubitable Correctness
- A. Bradley: SAT-Based Model Checking without Unrolling. VMCAI 2011

PDR: Explained and extended the implementation

- Property Directed Reachability
- N. Eén, A. Mishchenko, R. K. Brayton: Efficient implementation of property directed reachability. FMCAD 2011

PDR with Predicate Abstraction (easy extension of IC3/PDR to SMT)

- A. Cimatti, A. Griggio, S. Mover, St. Tonetta: IC3 Modulo Theories via Implicit Predicate Abstraction. TACAS 2014
- J. Birgmeier, A. Bradley, G. Weissenbacher: Counterexample to Induction-Guided Abstraction-Refinement (CTIGAR). CAV 2014

IC3, PDR, and Friends (2)

GPDR: Non-Linear CHC with Arithmetic constraints

- Generalized Property Directed Reachability
- K. Hoder and N. Bjørner: Generalized Property Directed Reachability. SAT 2012

SPACER: Non-Linear CHC with Arithmetic

- fixes an incompleteness issue in GPDR and extends it with under-approximate summaries
- A. Komuravelli, A. Gurfinkel, S. Chaki: SMT-Based Model Checking for Recursive Programs. CAV 2014

PolyPDR: Convex models for Linear CHC

- simulating Numeric Abstract Interpretation with PDR
- N. Bjørner and A. Gurfinkel: Property Directed Polyhedral Abstraction. VMCAI 2015

ArrayPDR: CHC with constraints over Airthmetic + Arrays

- Required to model heap manipulating programs
- A. Komuravelli, N. Bjørner, A. Gurfinkel, K. L. McMillan:Compositional Verification of Procedural Programs using Horn Clauses over Integers and Arrays. FMCAD 2015

IC3, PDR, and Friends (3)

Quip: Forward Reachable States + Conjectures

- Use both forward and backward reachability information
- A. Gurfinkel and A. Ivrii: Pushing to the Top. FMCAD 2015

Avy: Interpolation with IC3

- Use SAT-solver for blocking, IC3 for pushing
- Y. Vizel, A. Gurfinkel: Interpolating Property Directed Reachability. CAV 2014

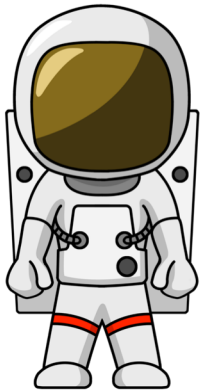
uPDR: Constraints in EPR fragment of FOL

- Universally quantified inductive invariants (or their absence)
- A. Karbyshev, N. Bjørner, S. Itzhaky, N. Rinetzky, S. Shoham: Property-Directed Inference of Universal Invariants or Proving Their Absence. CAV 2015

Quic3: Universally quantified invariants for LIA + Arrays

- Extending Spacer with quantified reasoning
- A. Gurfinkel, S. Shoham, Y. Vizel: Quantifiers on Demand. ATVA 2018

Spacer: Solving SMT-constrained CHC



Spacer: a solver for SMT-constrained Horn Clauses

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

Supported SMT-Theories

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

Support for Non-Linear CHC

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

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. $(s \wedge F_i \wedge Tr \wedge cex')$ 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 $L \wedge F_i \wedge Tr \Rightarrow L'$

Induction

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

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. $(s \wedge R_i \wedge Tr \wedge cex')$ 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 $L \wedge R_i \wedge Tr \Rightarrow L'$

Theory dependent

Induction

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

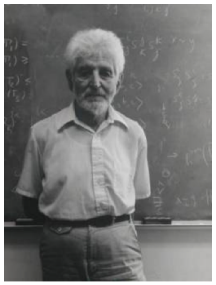
$$((F_i \wedge Tr) \vee Init') \Rightarrow \varphi'$$

$$\varphi' \Rightarrow \neg\neg c'$$

Looking for ϕ’

ARITHMETIC CONFLICT

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

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)

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

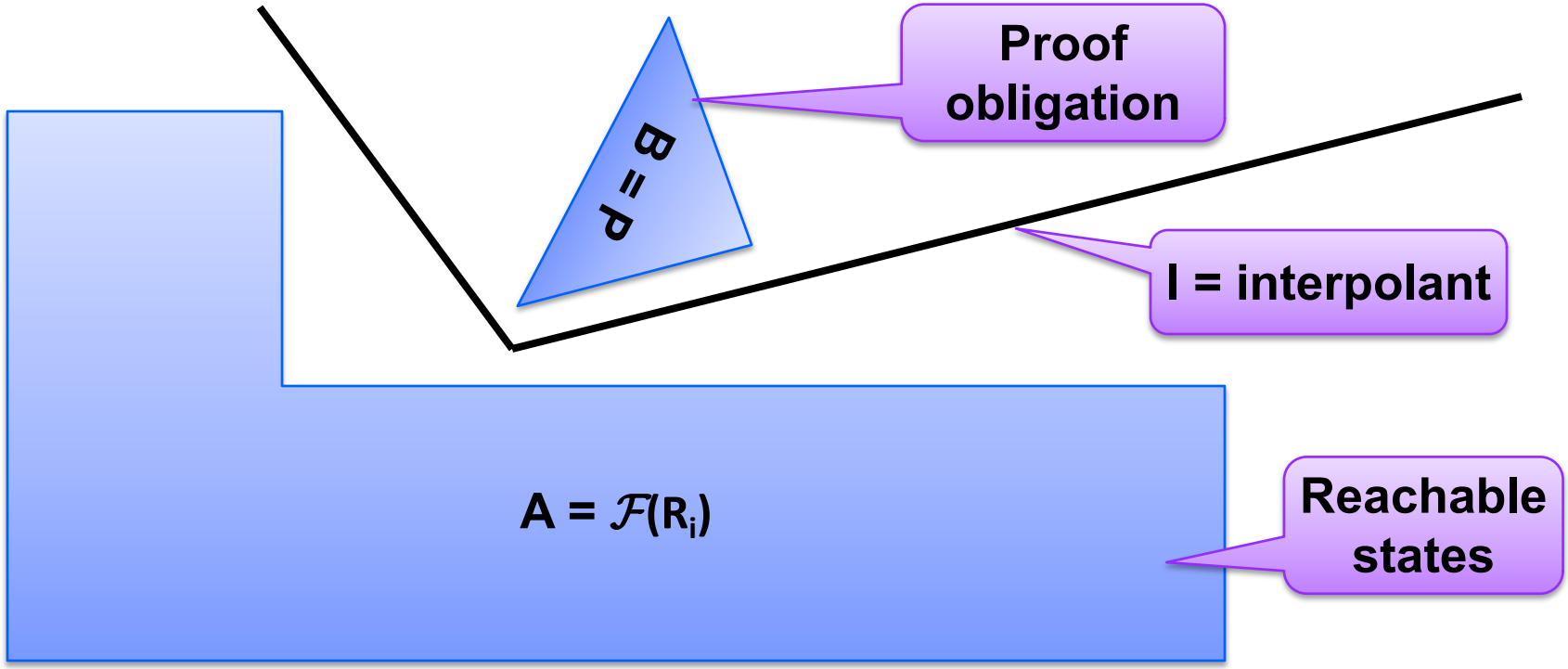
$$ITP(A, B) = p(b)$$

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

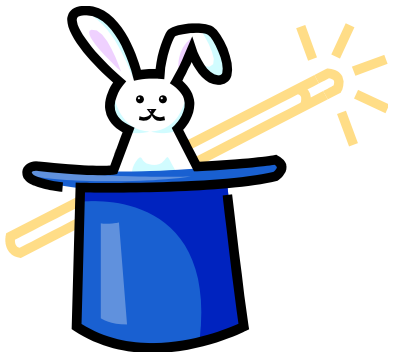
Conflict For $0 \leq i < N$, given a counterexample $\langle P, i+1 \rangle \in Q$ s.t. $\mathcal{F}(F_i) \wedge P'$ is unsatisfiable, add $P^{\uparrow} = \textsc{Itp}(\mathcal{F}(F_i), P')$ to F_j for $j \leq i+1$.

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 $(\wedge B_i \Rightarrow \vee A_j)$

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

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

M is *unsatisfiable* iff $M \vdash 0 \geq 1$

- e.g., $x + y > 10, -x > 5, -y > 3 \vdash (x+y-x-y) > (10 + 5 + 3) \vdash 0 > 18$

M is unsatisfiable iff there exist *Farkas* coefficients $g_1, \ldots, g_n$ such that

- $g_i \geq 0$
- $g_1 \times t_1 + \ldots + g_n \times t_n = 0$
- $g_1 \times b_1 + \ldots + g_n \times b_n \geq 1$

Frakas Lemma Example

Interpolants

$$\left.\begin{array}{rl} z + x + y > 10 & \times 1 \\ -z > -5 & \times 1 \end{array}\right\} \; x + y > 5$$

$$\left.\begin{array}{rl} -x > 5 & \times 1 \\ -y > 3 & \times 1 \end{array}\right\} \; x + y < -8$$

$$0 > 13$$

Interpolation for Linear Real Arithmetic

Let $M = A \wedge B$ be UNSAT, where

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

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

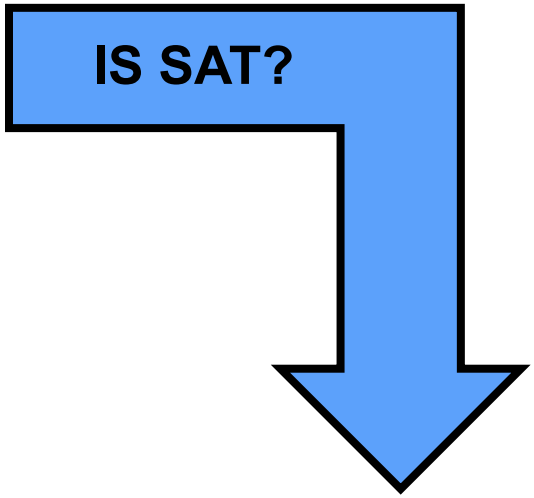
Then

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

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

Program Verification with HORN(LIA)

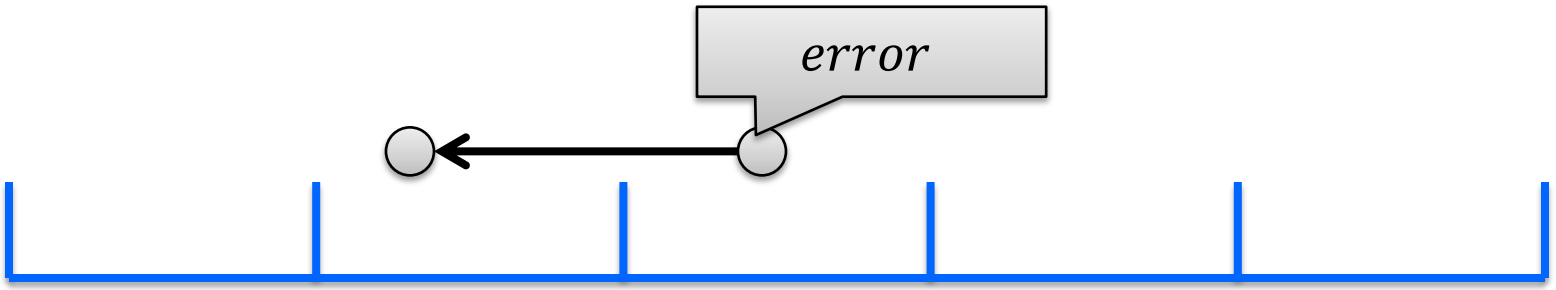
```
z = x; i = 0;
assume (y > 0);
while (i < y) {
  z = z + 1;
  i = i + 1;
}
assert(z == x + y);
```




```
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
```

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 ∧ s are UNSAT, compute an interpolant I such that

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

Use an SMT solver to decide that s ∧ 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

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 $(\wedge B_i \Rightarrow \vee A_j)$

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

$$s \subseteq pre(c)$$

$$\equiv \quad s \Rightarrow \exists X' \,.\, Tr \wedge c'$$

Computing a predecessor ***s*** of a counterexample ***c***

ARITHMETIC DECIDE

Model Based Projection

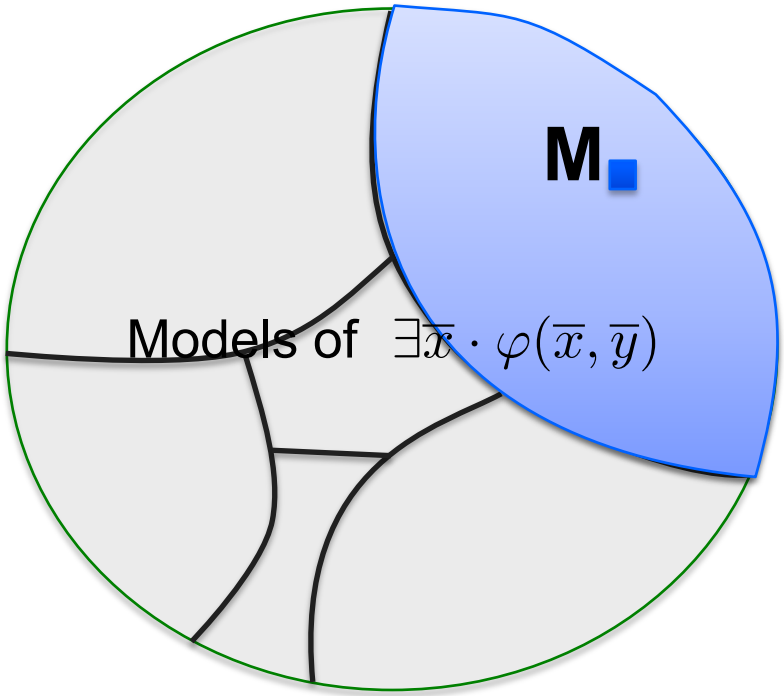
Definition: Let ϕ be a formula, U a set of variables, and M a model of ϕ. Then ψ = MBP (U, M, ϕ) is a Model Based Projection of U, M and ϕ iff

1. ψ is a monomial
2. Vars(ψ) $\subseteq$ Vars(ϕ) $\setminus$ U
3. M $\models \psi$
4. $\psi \Rightarrow \exists$ U . ϕ

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

A quantifier elimination is a procedure that takes a formula of the form $\exists x\, \psi(x)$ and returns an equivalent formula φ without existential quantifier and without the variable x

- QELIM($\exists x\, \psi(x)$) = φ and $\exists x\, \psi(x) \iff \varphi$

Quantifier elimination in propositional logic

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

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

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

The case of lower bounds is dual

- using $-\infty$ and $t+\epsilon$

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

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

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

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

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. $(F_i \wedge Tr \wedge cex')$, and let **s** = $MBP(X', F_i \wedge Tr \wedge cex')$

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 $F_i \wedge Tr \Rightarrow L'$

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

$$Init(X) \rightarrow P(X)$$

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

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

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

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: $Init \rightarrow \neg Bad$
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 $F_N \rightarrow \neg Bad$, then set $N \leftarrow N + 1$ and $Q \leftarrow \emptyset$.

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

Decide If there is a $t \in Q$, with $c = \langle m, i + 1\rangle \in t$, $m_1 \rightarrow m$, $l_0 \wedge m_0^o \wedge m_1'$ is satisfiable, and $l_0 \wedge m_0^o \wedge m_1' \rightarrow F_i \wedge F_i^o \wedge Tr \wedge m'$ 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 $0 \leq j \leq i + 1$.

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

until ∞;

counterexample is a tree

two predecessors

theory-aware **Conflict**

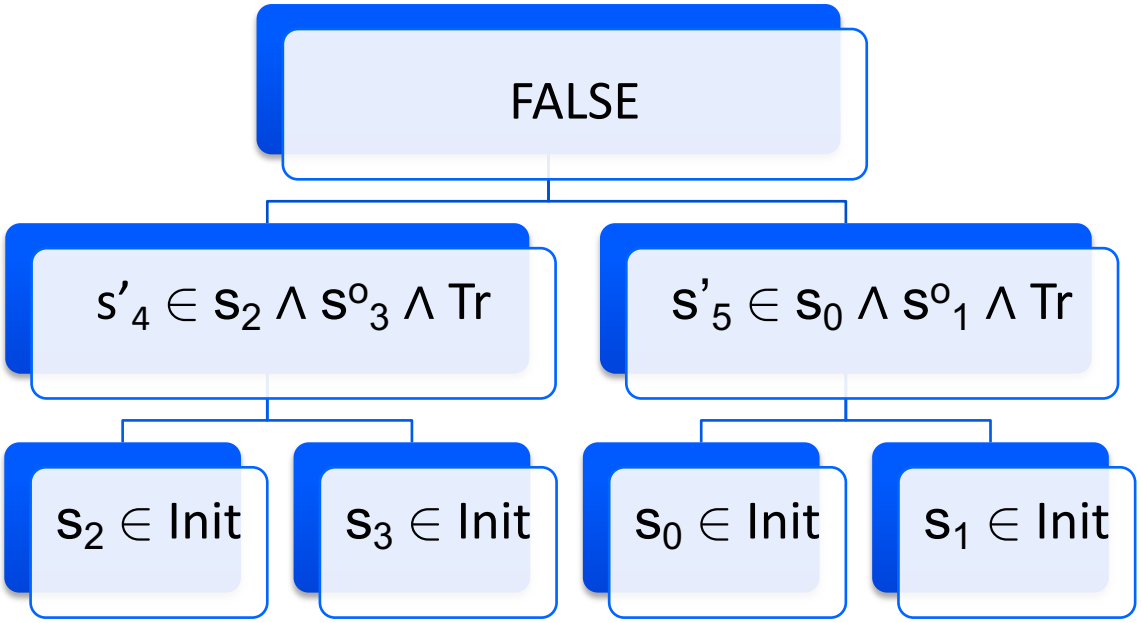
Counterexamples to non-linear CHC

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

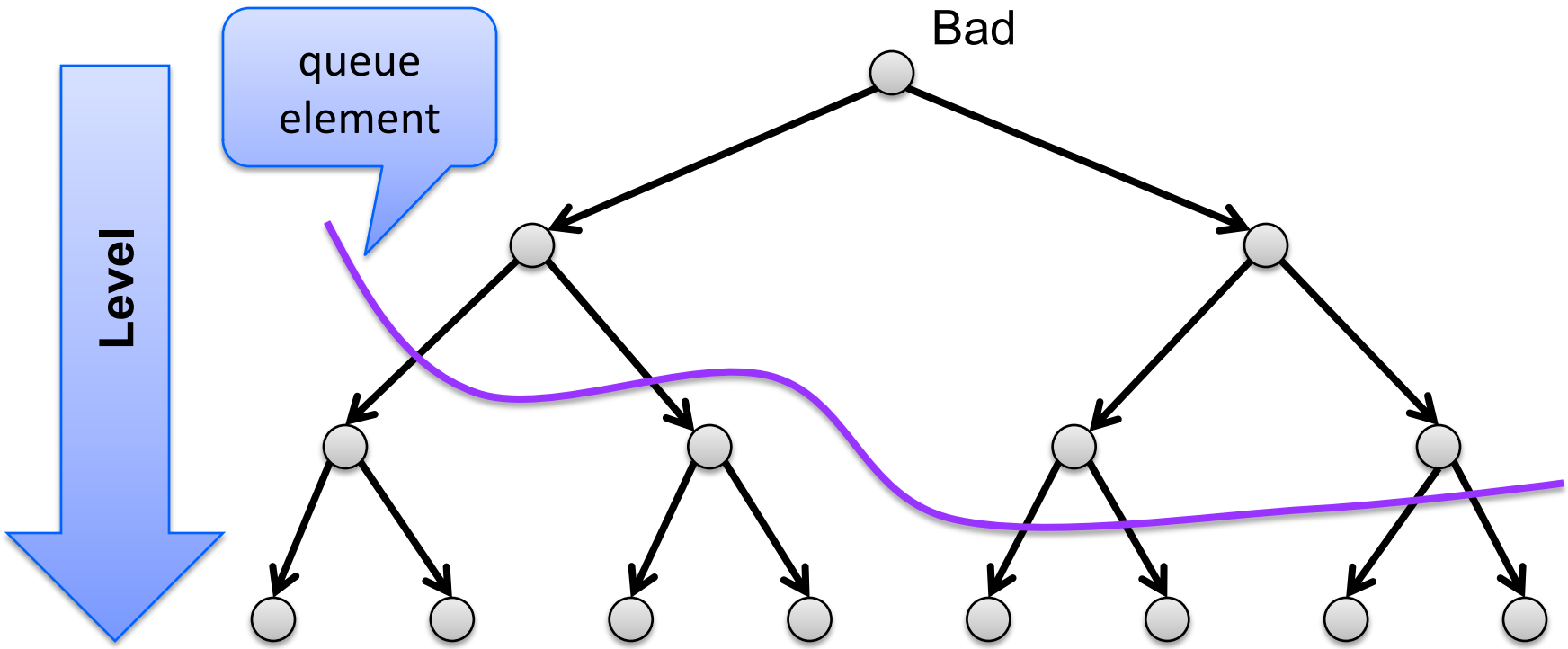
- we call such a derivation a counterexample

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)

GPDR: Deciding predecessors

Decide If there is a $t \in Q$, with $c = \langle m, i+1 \rangle \in t$, $m_1 \rightarrow m$, $l_0 \wedge m_0^o \wedge m_1'$ is satisfiable, and $l_0 \wedge m_0^o \wedge m_1' \rightarrow F_i \wedge F_i^o \wedge Tr \wedge m'$ 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 $\langle Init(X), Tr(X, X^o, X'), Bad(X)\rangle$.
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: $Init \rightarrow \neg Bad$
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 $F_N \rightarrow \neg Bad$, then set $N \leftarrow N + 1$ and $Q \leftarrow \emptyset$.

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

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

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

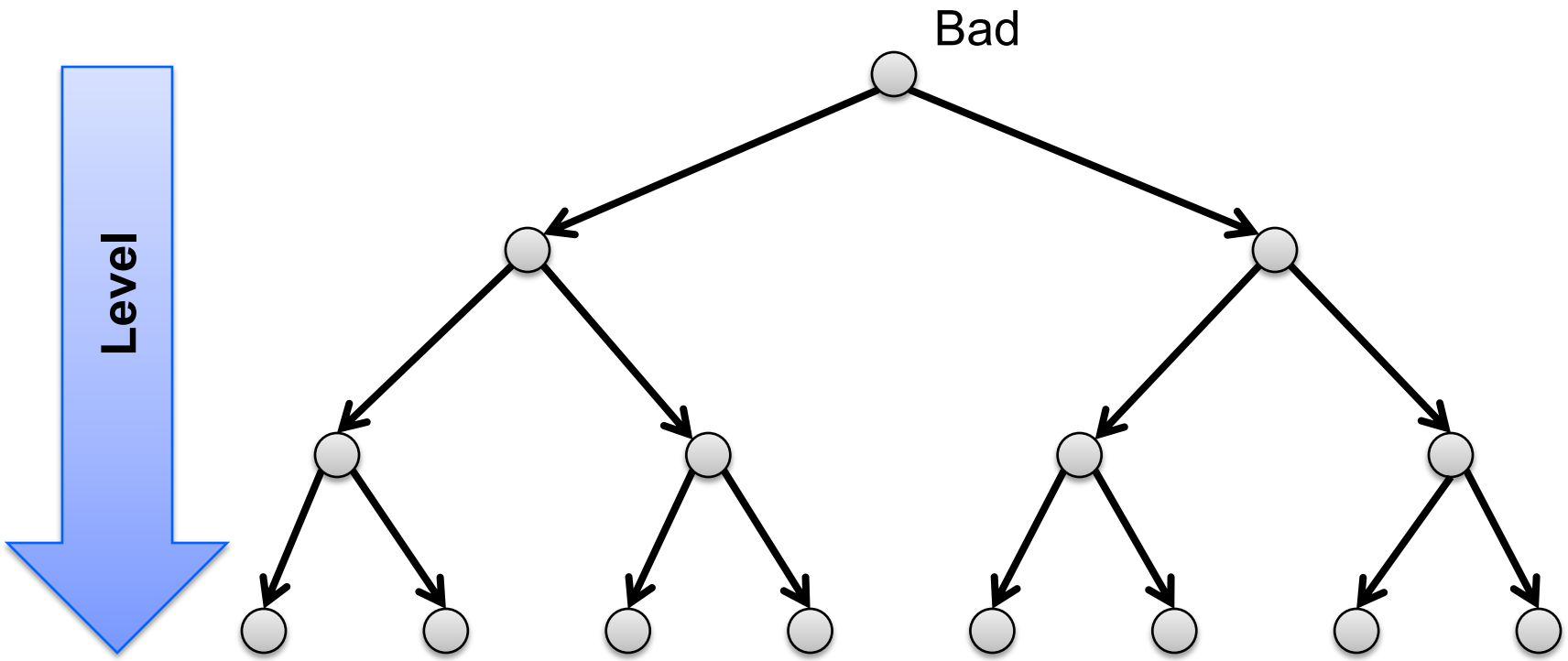
Conflict If there is an $\langle m, i+1\rangle \in Q$, s.t. $\mathcal{F}(F_i) \wedge m'$ is unsatisfiable. Then, add $\varphi = \text{Itp}(\mathcal{F}(F_i), m')$ to F_j, for all $0 \leq j \leq i+1$.

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 $\varphi \notin F_{i+1}$, $\mathcal{F}(\phi \wedge F_i) \rightarrow \phi'$, then add φ to F_j, for all $j \leq i+1$.

until ∞;

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 $\psi = \mathcal{F}(F_i, \vee\text{REACH}) \wedge m'$. 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 $s^o \in \text{MBP}(\{X, X'\}, \psi)$.

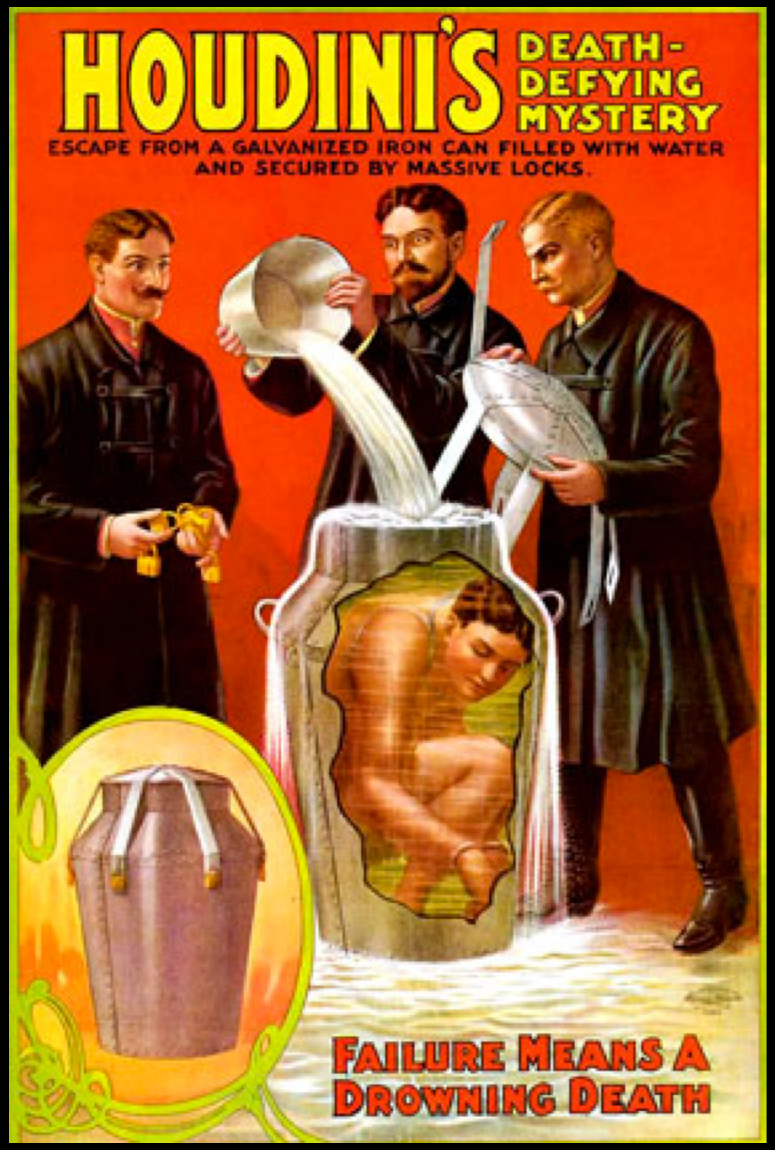
DecideMust

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

DecideMay

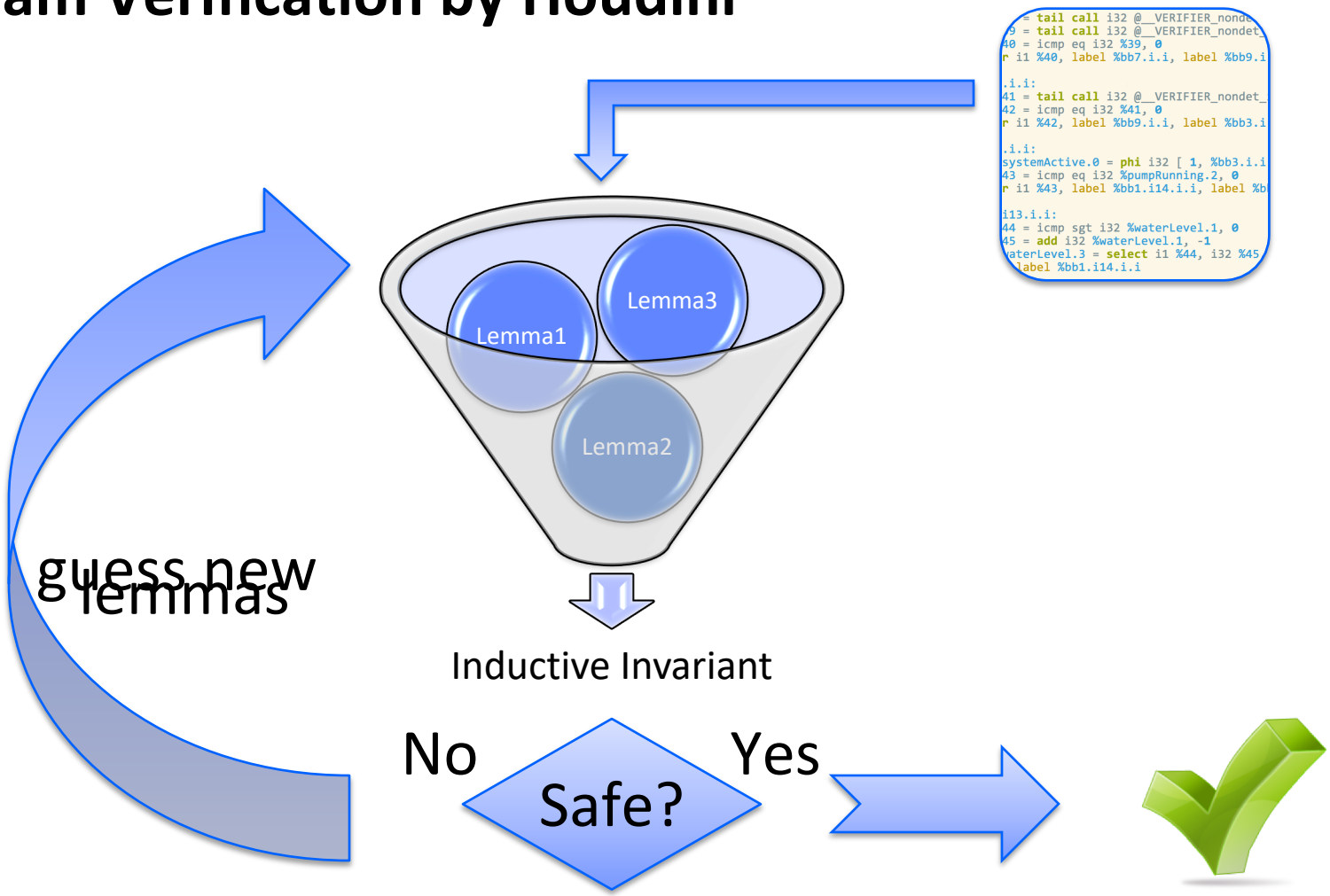
- use over-approximation of a calling context to guess an approximation of the call-site
- the call-site either refutes the approximation (**Conflict**) or refines it with a witness (**Successor**)

CHC VIA MACHINE LEARNING



Cormac Flanagan, K. Rustan M. Leino: Houdini, an Annotation Assistant for ESC/Java. FME 2001: 500-517

Program Verification by Houdini



Finding an Inductive Invariant

Discovering an inductive invariants involves two steps

Step 1: find a candidate inductive invariant **Inv**

Step 2: check whether **Inv** is an inductive invariant

Invariant Inference is the process of automating both of these phases

Finding an Inductive Invariant

Two popular approaches to invariant inference:

Machine Learning based Invariant Synthesis (**MLIS**)

- e.g. ICE: Pranav Garg, Christof Löding, P. Madhusudan, Daniel Neider: ICE: A Robust Framework for Learning Invariants. CAV 2014: 69-87
- referred to as a Black-Box approach

SAT-based Model Checking (**SAT-MC**)

- e.g. IC3: Aaron R. Bradley: SAT-Based Model Checking without Unrolling. VMCAI 2011: 70-87
- referred to as a White-Box approach

Our Goal

Study the Relationship between SAT-MC and MLIS

Or, is there a difference between White-Box and Black-Box?

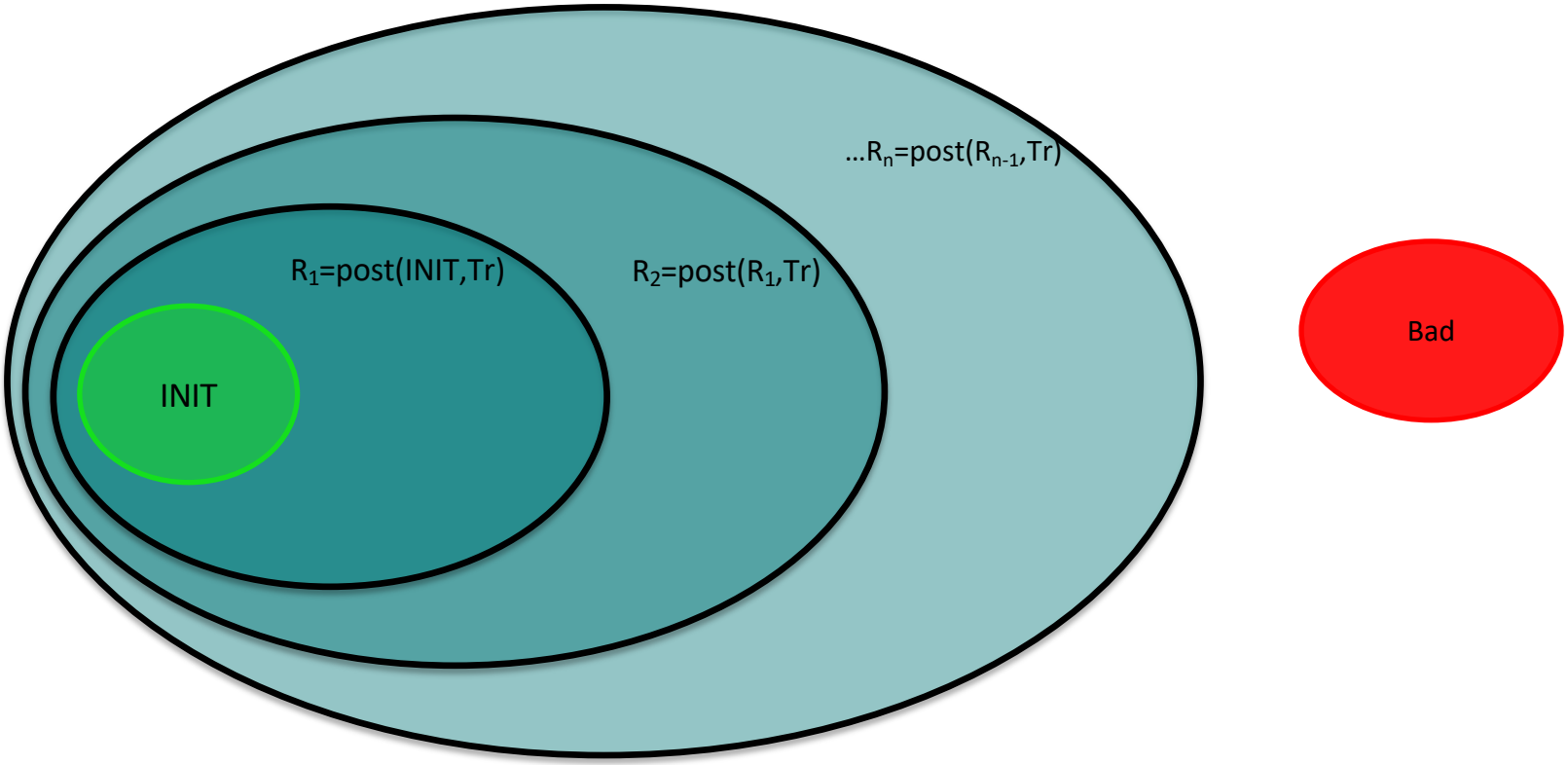
Our Goal

Study the Relationship between SAT-MC and MLIS

Or, is there a difference between White-Box and Black-Box?

- Study two state-of-the-art algorithms: ICE and IC3
- In other words: **can we describe IC3 as an instance of ICE?**

Reachability Analysis



Reachability Analysis

Computing states reachable from a set of states S using the post operator

$$\begin{cases} post^0(S) = S \\ post^{i+1} = post^i(S) \cup \{t \mid s \in S \wedge (s,t) \in Tr\} \end{cases}$$

Computing states reaching a set of states S using the pre operator

$$\begin{cases} pre^0(S) = S \\ pre^{i+1} = pre^i(S) \cup \{t \mid s \in S \wedge (t,s) \in Tr\} \end{cases}$$

Transitive closure is denoted by post* and pre*

SAT-based Model Checking

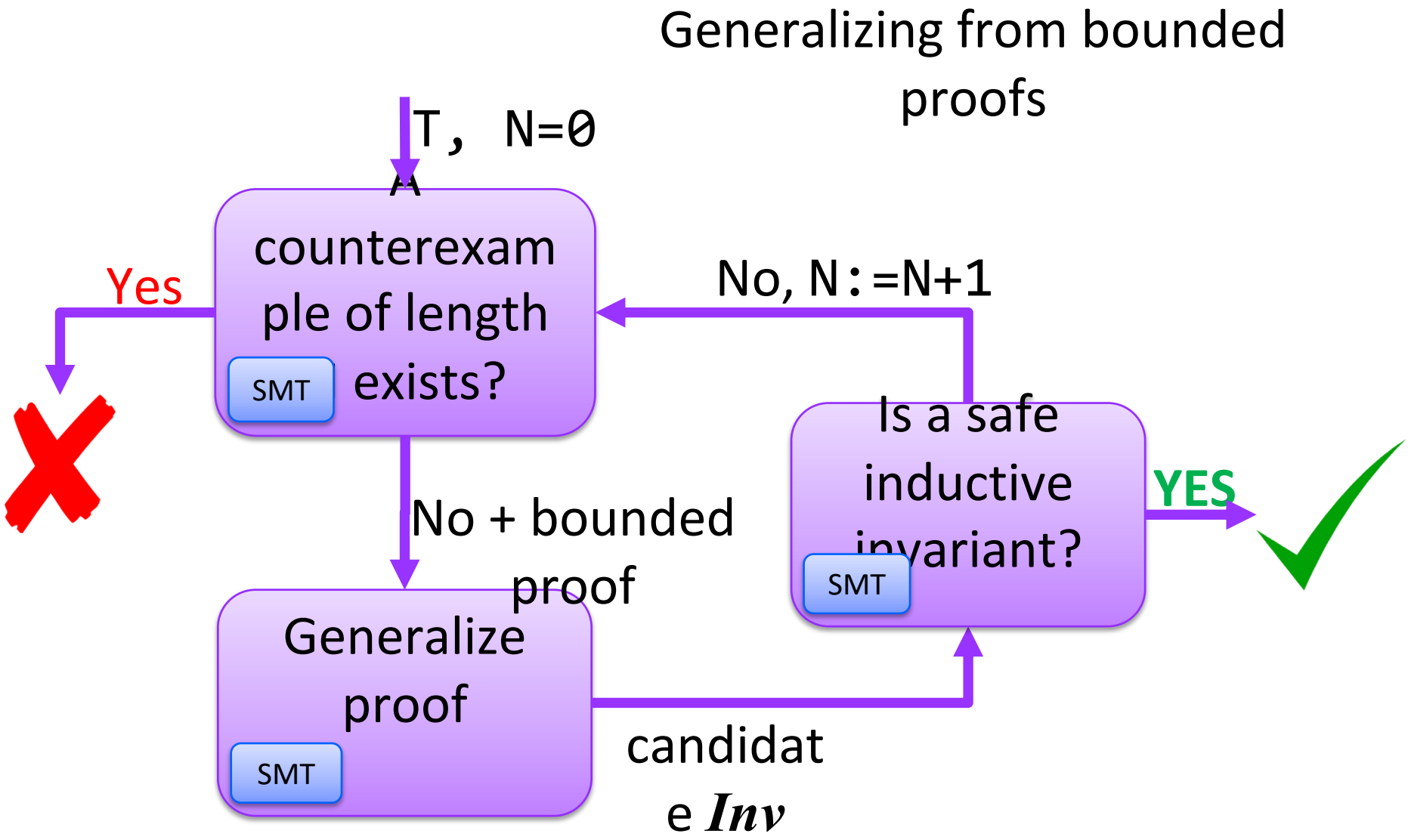
Search for a counterexample for a specific length

If a counterexample does not exist, generalize the bounded proof into a candidate Inv

Check if Inv is a safe inductive invariant

Referred to as White-Box: Rely on a close interaction between the main algorithm and the decision procedure used

SMT-based Model Checking



8

Machine Learning-based Invariant Synthesis

MLIS consists of two entities: Teacher and Learner

Learner comes up with a candidate Inv

- Agnostic of the transition system
- Using machine learning techniques

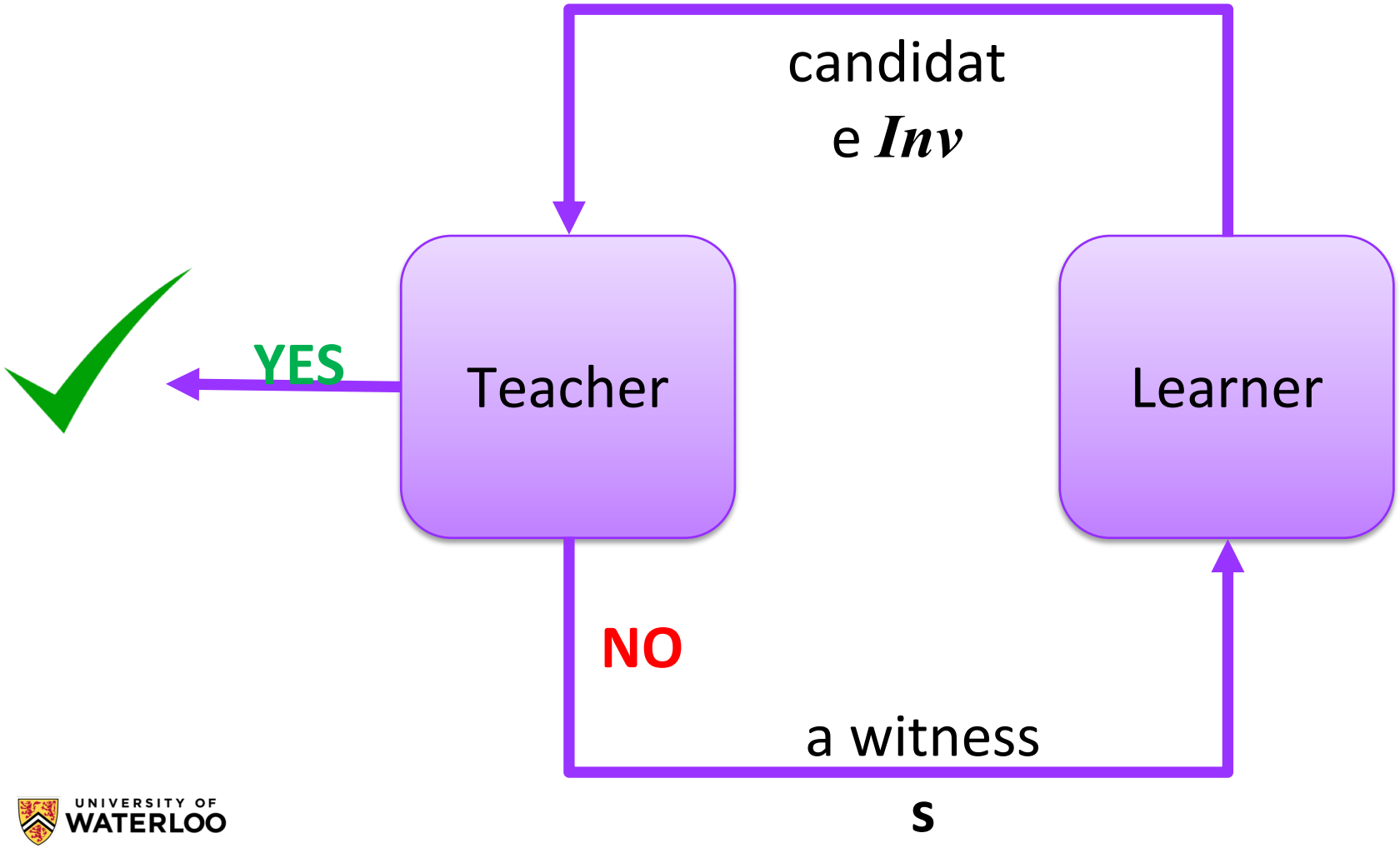
Learner asks the Teacher if Inv is a safe inductive invariant

If not, Teacher replies with a witness: positive or negative

- Aware of the transition system

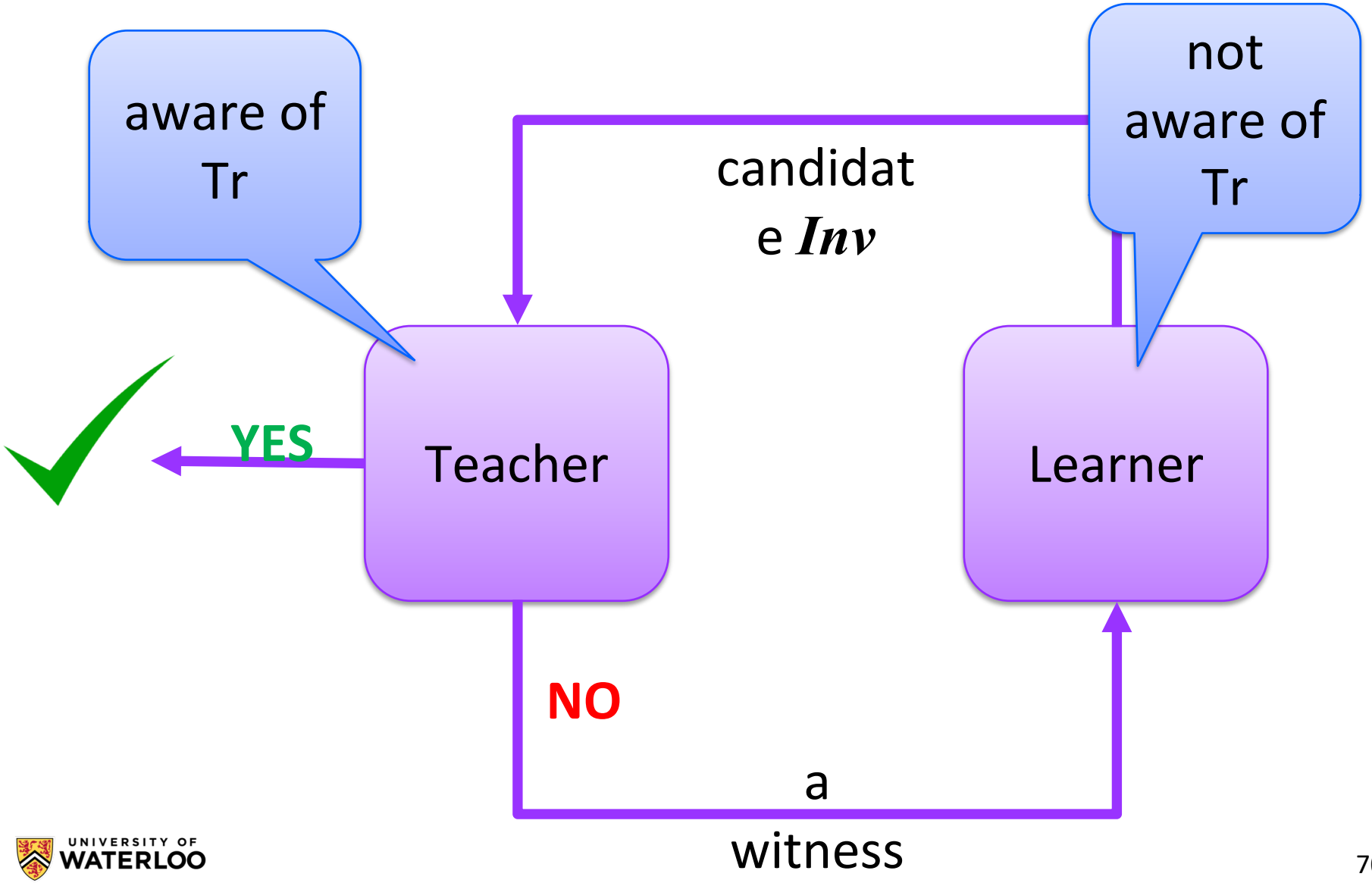
Referred to as Black-Box

Machine Learning-based Invariant Synthesis



0

Machine Learning-based Invariant Synthesis



ICE: MLIS Framework

(Garg et al. CAV 2014)

Given a transition system T=(INIT, Tr, Bad) and a candidate *Inv* generated by the Learner

When the Teacher determines *Inv* is not a safe inductive invariant, a witness is returned:

- E-example: $s \in post^*(INIT)$ but $s \notin Inv$
- C-example: $s \in pre^*(Bad)$ and $s \in Inv$
- I-example: $(s,t) \in T$ such that $s \in Inv$ but $t \notin Inv$

Given a set of states S, the triple (E, C, I) is an **ICE state**

- $E \subseteq S$, $C \subseteq S$, $I \subseteq S \times S$

A set $J \subseteq S$ is **consistent** with ICE state iff

- $E \subseteq J$ and $J \cap C = \emptyset$
- for $(s,t) \in I$, if $s \in J$ then $t \in J$

ICE

(Garg et al. CAV 2014)

Input: A transition system $T = (\mathcal{V}, Init, Tr, Bad)$

$Q \leftarrow \emptyset$ $\textsc{Learner}(T)$; $\textsc{Teacher}(T)$;

repeat

  $J \leftarrow \textsc{Learner.SynCandidate}(Q)$;

  $\varepsilon \leftarrow \textsc{Teacher.IsInd}(J)$;

  if $\varepsilon = \bot$ **then return** SAFE;

  $Q \leftarrow Q \cup \{\varepsilon\}$;

until ∞;

ICE

(Garg et al. CAV 2014)

Input: A transition system $T = (\mathcal{V}, I$

$Q \leftarrow \emptyset$ $\textsc{Learner}(T)$; $\textsc{Teacher}(T)$;

repeat

  $J \leftarrow \textsc{Learner.SynCandidate}(Q)$;

  $\varepsilon \leftarrow \textsc{Teacher.IsInd}(J)$;

  if $\varepsilon = \bot$ **then return** SAFE;

  $Q \leftarrow Q \cup \{\varepsilon\}$;

until ∞;

No requirement for incrementality

J must be consistent with Q

The Learner is passive - has no control over the Teacher

4

PDR/IC3 – SAT Queries

Trace $[F_0,\ldots,F_N]$, and $Q \subseteq pre^*(Bad)$, a state $s \in Q \cap F_{i+1}$

Strengthening

- $(F_i \wedge \neg s) \wedge T \wedge s'$
- is $(F_i \wedge \neg s) \wedge T \rightarrow \neg s'$ valid?

If this is satisfiable then there exists a state t in F_i that can reach Bad

- This looks like a C-example

In order to ”fix” F_i t must be removed

Now check

- $(F_{i-1} \wedge \neg t) \wedge T \wedge t'$

PDR/IC3 – SAT Queries

Trace $[F_0,\ldots,F_N]$, try to push a lemma $c \in F_i$ to F_{i+1}

Pushing

- $(F_i \wedge c) \wedge T \wedge \neg c'$
- is $(F_i \wedge c) \wedge T \rightarrow c'$ valid?

If this is satisfiable then there exists a pair $(s,t) \in T$ s.t. $s \in F_i$ and $t \notin F_{i+1}$

- It looks like an I-example
  - Also, can be either an E- or C-example

In order to "fix" F_i, either s is removed from F_i or t is added to it

- Strengthening vs Weakening

The Problem

IC3 reasons about relative induction

F is inductive relative to G when:

- INIT $\rightarrow$ F, and
- $G(V) \wedge F(V) \wedge T(V,V') \rightarrow F(V')$

But, in ICE, the Learner (Teacher) asks (answers) about induction

and, the Learner in ICE is passive

- cannot control the Teacher in any way
- No guarantee for incrementality

RICE – ICE + Relative Induction

Input: A transition system $T = (\mathcal{V}, Init, Tr, Prd)$
$Q \leftarrow \emptyset$;
$\textsc{Learner}(T)$; $\textsc{Teacher}(T)$;
repeat
  $(F, G) \leftarrow \textsc{Learner.SynCandAndBase}(Q)$;
  $\varepsilon \leftarrow \textsc{Teacher.IsRelInd}(F, G)$;
  if $\varepsilon = \bot \wedge G = true$ **then return** SAFE;
  $Q \leftarrow Q \cup \{\varepsilon\}$;
until ∞;

G allows the Learner to have some control over the Teacher

When G is true it is a regular inductive check

8

RICE – ICE + Relative Induction

The Teacher in RICE reacts to queries about relative induction

The Learner can "manipulate" the Teacher using relative induction

RICE is a generalization of ICE where the Learner is an active learning algorithm

9

RICE – ICE + Relative Induction

The Teacher in RICE reacts to queries about relative induction

Is F inductive relative to G?

If not, a witness is returned:

- E-example: s ∈ post*(INIT) but s ∉ F
- C-example: s ∈ pre*(Bad) and s ∈ F

- I-example: (s,t) ∈ T such that s ∈ F ^ G but t ∉ F

0

IC3 AS AN INSTANCE OF RICE

1

IC3 Learner

The IC3 Learner is active and incremental

Maintains the following:

- a trace $[F_0, \ldots, F_N]$ of candidates
- RICE state Q=(E, C, I)

The Learner must be consistent with the RICE state

E-examples and C-examples may exist when F is inductive relative to G

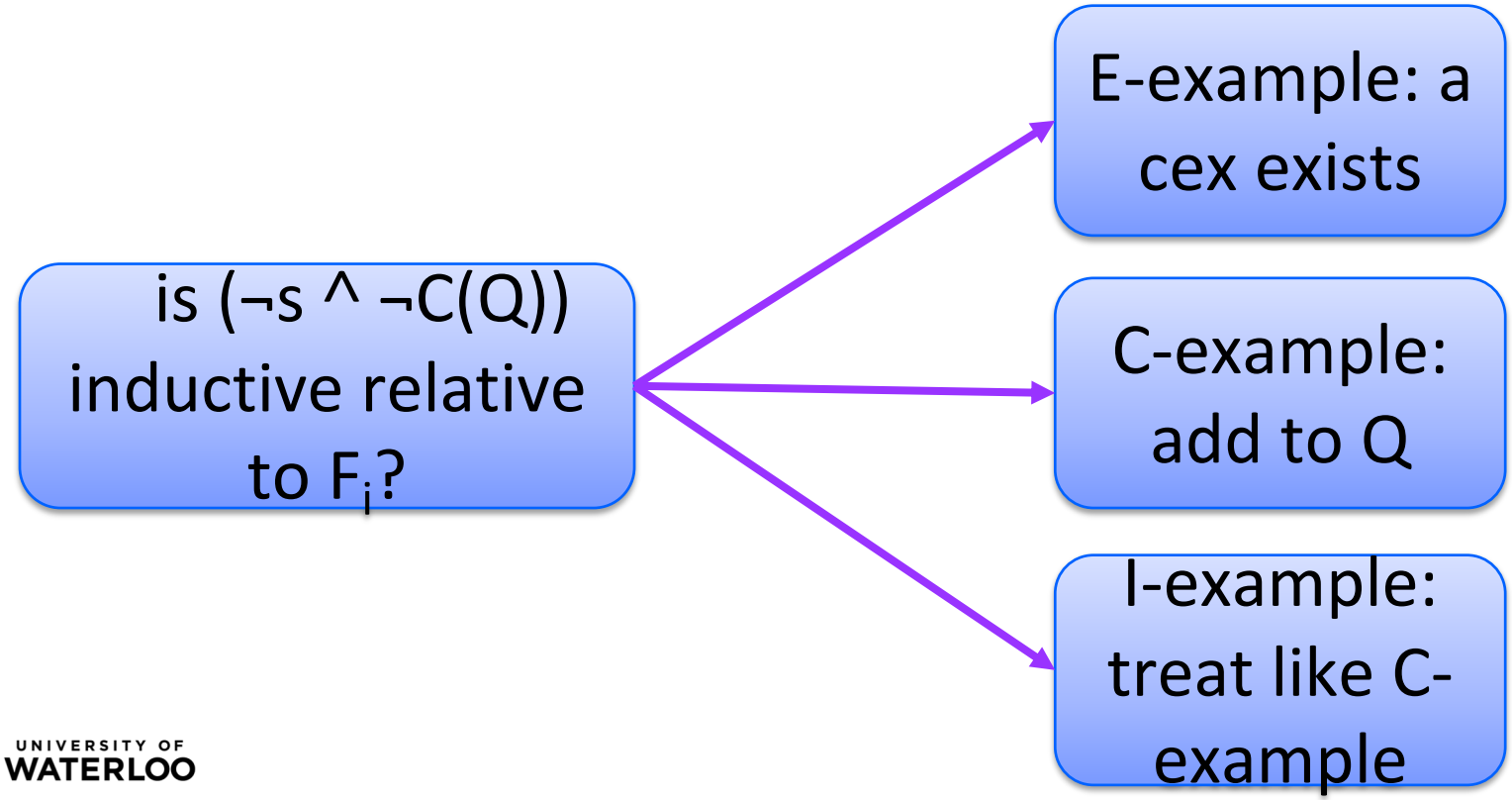
- The Teacher may return an E-example or C-example when F is inductive relative to G

IC3 Learner - Strengthening

INIT ⟶ F, and
G(V) ∧ F(V) ∧ T(V,V') ⟶ F(V')

Strengthening:

- a C-example s in F_i
- $(F_i \wedge \neg s \wedge \neg C(Q)) \wedge T \wedge (s \vee C(Q))'$

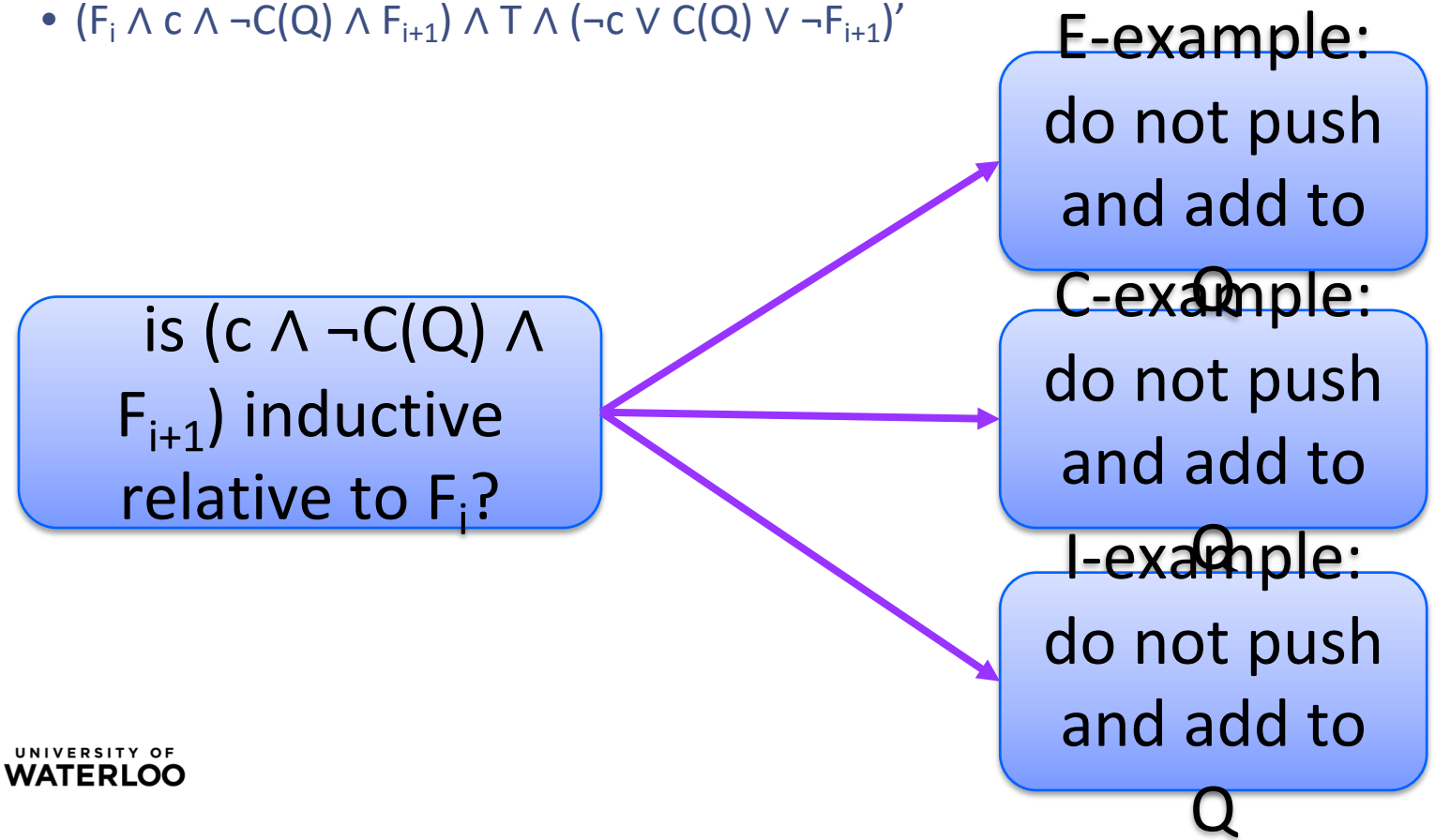


IC3 Learner - Pushing

INIT ⟶ F, and
G(V) ∧ F(V) ∧ T(V,V') ⟶ F(V')

Pushing:

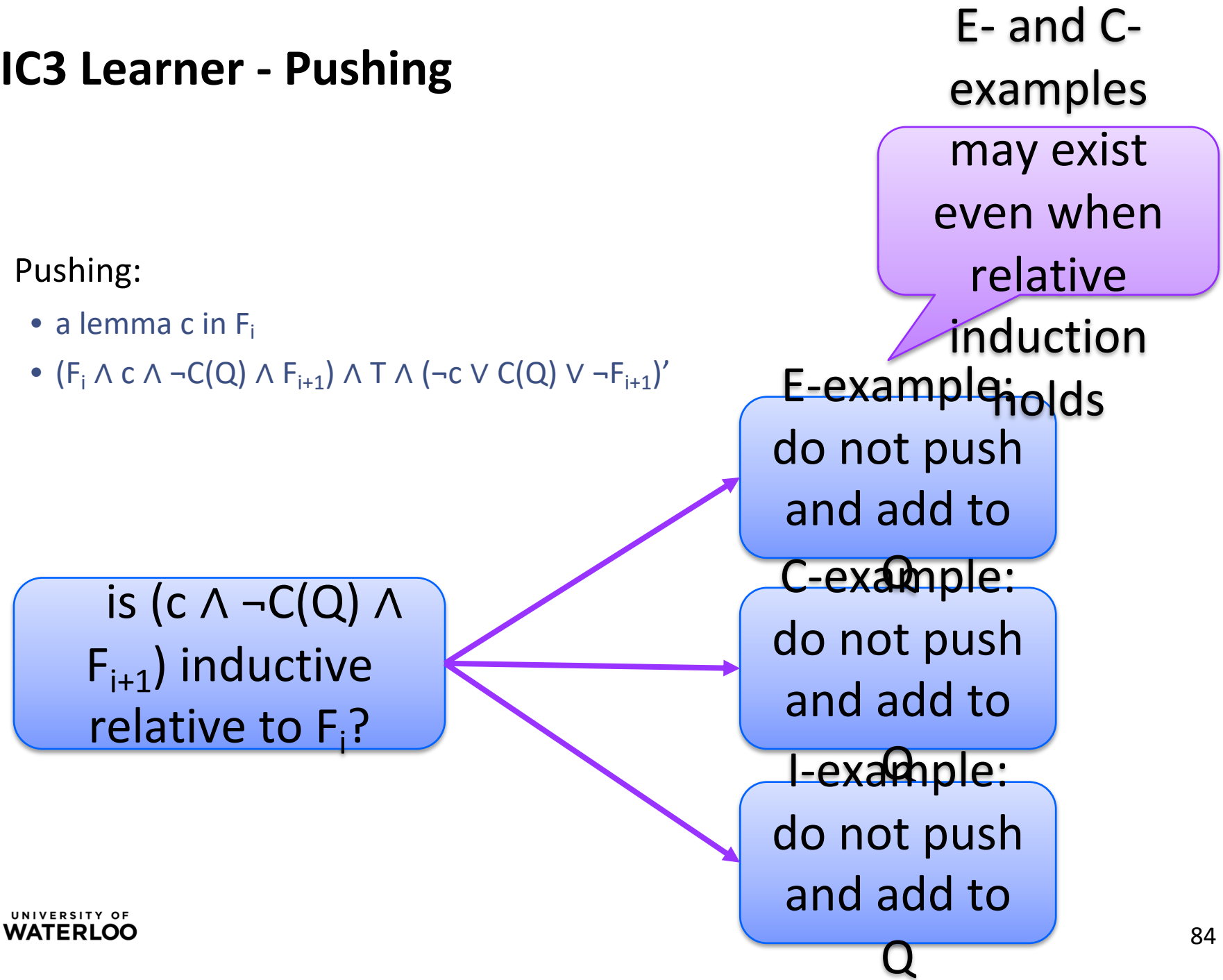
- a lemma c in F_i
- $(F_i \wedge c \wedge \neg C(Q) \wedge F_{i+1}) \wedge T \wedge (\neg c \vee C(Q) \vee \neg F_{i+1})'$



IC3 Learner - Pushing

Pushing:

- a lemma c in F_i
- $(F_i \wedge c \wedge \neg C(Q) \wedge F_{i+1}) \wedge T \wedge (\neg c \vee C(Q) \vee \neg F_{i+1})'$



5

IC3 Teacher

Using a general Teacher, the described Learner computes a trace $[F_0, \ldots, F_N]$ such that

- post*(INIT) → F_i → ¬pre*(Bad)

Generic Teacher is infeasible

- required to look arbitrary far into the future (for E-examples)
- required to look arbitrary far into the past (for C-examples)

Solution: add restrictions on E- and C-examples

IC3 Teacher

Is F inductive relative to G?

If not, a witness is returned:

- C-example: s ∈ pre^m(Bad) and s ∈ F
- I-example: (s,t) ∈ T such that s ∈ F ^ G but t ∉ F
- E-example: s ∈ $post^0$(INIT) but s ∉ F

Claim: Using this IC3 Teacher and the IC3 Learner results in an algorithm that behaves like (simulates) IC3

7

What Can We Learn?

Can we lift the restriction that requires E-example to be in INIT only?

- Yes, a variant of IC3, called Quip, does that

There is no “real” weakening mechanism in IC3

- Future work…

Can we introduce other active Learners for MLIS?

8

Conclusions

An extension of ICE to RICE

- Taking ques from IC3: incrementality, active Learner
- Overcomes a deficiency in ICE

IC3 can benefit from (R)ICE

- Weakening, E-examples, …

CHC-COMP: CHC Solving Competition

First edition on July 13, 2018 at HVCS@FLO

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

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

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

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

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