Constrained Horn Clauses (CHC)

Automated Program Verification (APV)
Fall 2018

Prof. Arie Gurfinkel

UNIVERSITY OF WATERLOO

PREDICATE ABSTRACTION

Predicate Abstraction

Extends Boolean reasoning methods to non-Boolean domains

Given a set of predicates P, abstract transition relation by restricting its effects to the set P

- Each step of Tr sets some predicates in P to true and some to false
- Computing abstraction requires theory reasoning
- Abstract transition relation is Boolean, so Boolean methods can be applied

Predicate abstraction is an over-approximation

- May introduce spurious counterexamples that cannot be replayed in the real system

Abstraction-Refinement: replay counterexamples using theory reasoner

- Use BMC to replay
- Use Interpolation to learn new predicates

Implicit Predicate Abstraction with IC3

Idea: do not compute abstract transition relation upfront!

IC3 only requires computing one predecessor at a time

- Use theory reasoning to compute a predecessor
- Each POB/CTI/state is a Boolean valuations to all predicates

The rest is exactly like Boolean IC3

- Except that predecessor generalization does not work

To refine, replay the counterexamples using theory solver

- use interpolation to learn new predicates

Interesting idea to implement in Z3 using Spacer/CHC for refinement

Implicit Predicate Abstraction Construction

Boolean state variables

Predicates over state variables

$$\left(\bigwedge_i (b_i \leftrightarrow p_i(V)) \right) \wedge$$

Original transition relation

$$Tr(V, V') \wedge$$

Post-state

$$\left(\bigwedge (b'_i \leftrightarrow p_i(V')) \right)$$

There is a counter-example over b_i variables iff there are no lemmas over p_i predicates that can block the counter-example

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

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 $\mathcal{T}$
- $p_1, \ldots, p_n, h$ are n-ary predicates
- $p_i[X]$ is an application of a predicate to first-order terms

UNIVERSITY OF WATERLOO

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

$$\overset{\text{head}}{h[X]} \leftarrow \overbrace{p_1[X_1], \ldots, p_n[X_n], \overset{\text{constraint}}{\phi}}^{\text{body}}.$$

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);
```

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

In SMT-LIB

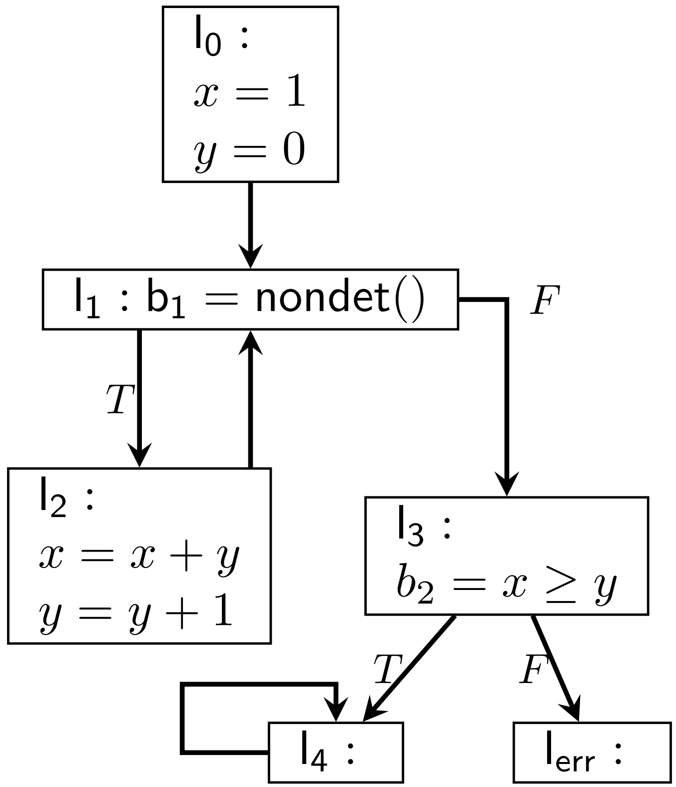
```
(set-logic HORN)

;; 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
```

Programs, CFG, Horn Clauses

$\text{int } x = 1;$
$\text{int } y = 0;$
$\text{while } (*) \ \{$
$\quad x = x + y;$
$\quad y = y + 1;$
$\}$
$\text{assert}(x \geq y);$



$$\langle 1 \rangle \ \mathsf{p}_0.$$
$$\langle 2 \rangle \ \mathsf{p}_1(x, y) \leftarrow \mathsf{p}_0, x = 1, y = 0.$$
$$\langle 3 \rangle \ \mathsf{p}_2(x, y) \leftarrow \mathsf{p}_1(x, y) \ .$$
$$\langle 4 \rangle \ \mathsf{p}_3(x, y) \leftarrow \mathsf{p}_1(x, y) \ .$$
$$\langle 5 \rangle \ \mathsf{p}_1(x', y') \leftarrow \mathsf{p}_2(x, y), x' = x + y, y' = y + 1.$$
$$\langle 6 \rangle \ \mathsf{p}_4 \leftarrow (x \geq y), \mathsf{p}_3(x, y).$$
$$\langle 7 \rangle \ \mathsf{p}_{\mathsf{err}} \leftarrow (x < y), \mathsf{p}_3(x, y).$$
$$\langle 8 \rangle \ \mathsf{p}_4 \leftarrow \mathsf{p}_4.$$
$$\langle 9 \rangle \ \bot \leftarrow \mathsf{p}_{\mathsf{err}}.$$

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:

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

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

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

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

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

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

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

Bjørner, Gurfinkel, McMillan, and Rybalchenko:

Horn Clause Solvers for Program Verification

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',

$$
\begin{array}{lll}
\text{CM1}: & init(V) & \rightarrow R_i(V) \\
\text{CM2}: & R_i(V) \wedge \rho_i(V, V') & \rightarrow R_i(V') \\
\text{CM3}: & (\bigvee_{i \in 1..N \setminus \{j\}} R_i(V) \wedge \rho_i(V, V')) & \rightarrow E_j(V, V') \\
\text{CM4}: & R_i(V) \wedge E_i(V, V') \wedge \rho_i^{=}(V, V') & \rightarrow R_i(V') \\
\text{CM5}: & R_1(V) \wedge \cdots \wedge R_N(V) \wedge error(V) & \rightarrow false \\
\hline
& \multicolumn{2}{c}{\text{multi-threaded program } P \text{ is safe}}
\end{array}
$$

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

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

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

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

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

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

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

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

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

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

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[n edge] 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

Relationship between CHC and Verification

A program satisfies a property iff corresponding CHCs are satisfiable

- satisfiability-preserving transformations == safety preserving

Models for CHC correspond to verification certificates

- inductive invariants and procedure summaries

Unsatisfiability (or derivation of FALSE) corresponds to counterexample

- the resolution derivation (a path or a tree) is the counterexample

CAVEAT: In SeaHorn the terminology is reversed

- SAT means there exists a counterexample – a BMC at some depth is SAT
- UNSAT means the program is safe – BMC at all depths are UNSAT

Semantics of Programming Languages

Denotational Semantics

- Meaning of a program is defined as the mathematical object it computes (e.g., partial functions).
- example: Abstract Interpretation

Axiomatic Semantics

- Meaning of a program is defined in terms of its effect on the truth of logical assertions.
- example: Hoare Logic, Weakest precondition calculus

Operational Semantics

- Meaning of a program is defined by formalizing the individual computation steps of the program.
- example: Natural (Big-Step) Semantics, Structural (Small-Step) Semantics

A Simple Programming Language (WHILE or IMP)

```
Prog  ::= def Main(x) { body_M }, …, def P (x) { body_P }

body  ::= stmt (; stmt)*

stmt  ::= x = E | assert (E) | assume (E) |
          while E do S | y = P(E) |
          L:stmt | goto L              (optional)

E     := expression over program variables
```

Axiomatic Semantics

An axiomatic semantics consists of:

- a language for stating assertions about programs;
- rules for establishing the truth of assertions.

Some typical kinds of assertions:

- This program terminates.
- If this program terminates, the variables x and y have the same value throughout the execution of the program.
- The array accesses are within the array bounds.

Some typical languages of assertions

- First-order logic
- Other logics (temporal, linear, separation)
- Special-purpose specification languages (Z, Larch, JML)

Assertions for WHILE

The assertions we make about WHILE programs are of the form:

$$\{A\}\ c\ \{B\}$$

with the meaning that:

- If A holds in state q and $q \to q'$
- then B holds in q'

A is the precondition and B is the post-condition

For example:

$$\{\ y \leq x\ \}\ z := x;\ z := z + 1\ \{\ y < z\ \}$$

is a valid assertion

These are called Hoare triples or Hoare assertions

Weakest Liberal Pre-Condition

Validity of Hoare triples is reduced to FOL validity by applying a **predicate transformer**

Dijkstra's weakest liberal pre-condition calculus [Dijkstra'75]

wlp (P, Post)

weakest pre-condition ensuring that executing P ends in Post

{Pre} P {Post} is valid IFF Pre ⇒ **wlp** (P, Post)

Horn Clauses by Weakest Liberal Precondition

`Prog ::= def Main(x) { body`$_M$` }, …, def P (x) { body`$_P$` }`

wlp (x=E, Q) = **let** x=E **in** Q

wlp (**`assert`**`(E)` , Q) = E ∧ Q

wlp (**`assume`**`(E)`, Q) = E ⇒ Q

wlp (**`while`** `E` **`do`** S, Q) = I(w) ∧
∀w . ((I(w) ∧ E) ⇒ wlp (S, I(w))) ∧ ((I(w) ∧ ¬E) ⇒ Q))

wlp (y `=` `P(E)`, Q) = p_{pre}(E) ∧ (∀ r. p(E, r) ⇒ Q[r/y])

ToHorn (**`def`** `P(x) {S}`) = wlp (`x0=x;`**`assume`**`(`p_{pre}`(x))`; S, p(x0, ret))
ToHorn (Prog) = wlp (`Main()`, true) ∧ ∀{P ∈ Prog} . ToHorn (P)

UNIVERSITY OF WATERLOO

Example of a WLP Horn Encoding

```
{Pre: y≥ 0}
 xo = x;
 yo = y;
 while y > 0 do
   x = x+1;
   y = y-1;
{Post: x=xo+yo}
```

ToHorn

```
C1: I(x,y,x,y) ← y>=0.
C2: I(x+1,y-1,xo,yo) ← I(x,y,xo,yo), y>0.
C3: false ← I(x,y,xo,yo), y<=0, x≠xo+yo
```

$\{y \geq 0\}$ P $\{x = x_{old}+y_{old}\}$ is **valid** IFF the $C_1 \land C_2 \land C_3$ is **satisfiable**

EXAMPLE

Control Flow Graph

A CFG is a graph of basic blocks

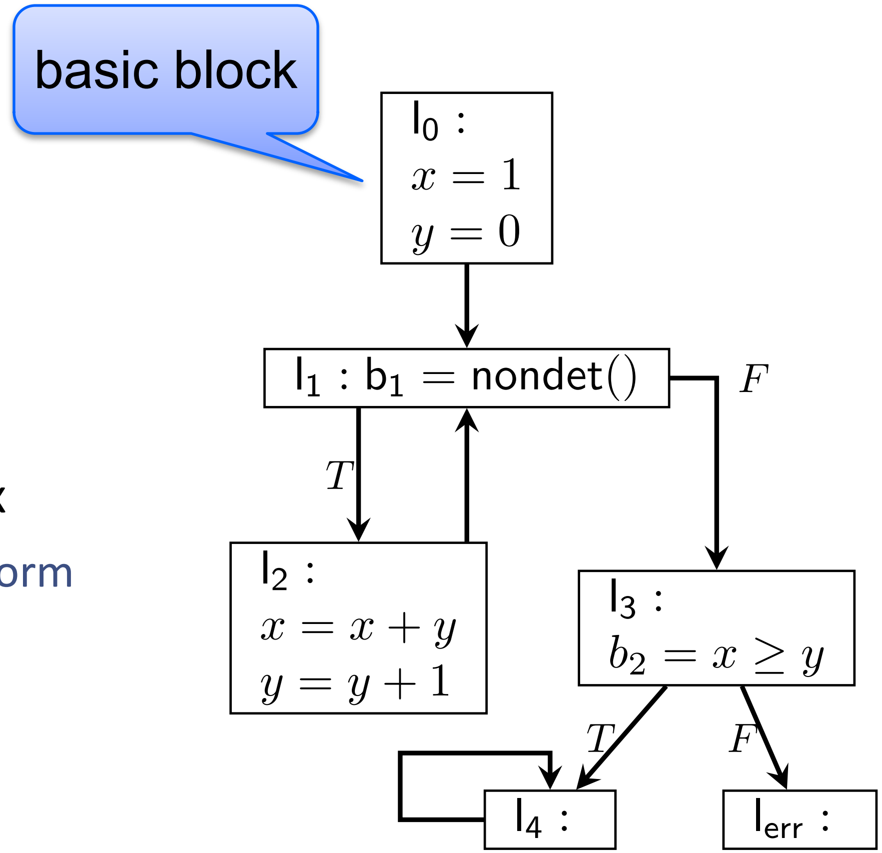
- edges represent different control flow

A CFG corresponds to a program syntax

- where statements are restricted to the form

L_i:S ; goto L_j

and S is control-free (i.e., assignments and procedure calls)



Dual WLP

Dual weakest liberal pre-condition

dual-wlp (P, Post) = ¬**wlp** (P, ¬Post)

s ∈ **dual-wlp** (P, Post) IFF there exists an execution of P that starts in s and ends in Post

dual-wlp (P, Post) is the weakest condition ensuring that an execution of P can reach a state in Post

Examples of dual-wlp

dual-wlp(**assume**(E), Q) = ¬wlp(**assume**(E), ¬ Q) = ¬(E ⇒ ¬ Q) = E ∧ Q

dual-wlp(x := x+y; y := y+1, x=x' ∧ y=y') = y+1=y' ∧ x+y=x'

wlp(x := x + y, ¬(y+1=y ∧ x=x'))

= let x = x+y in ¬ (y+1=y' ∧ x=x')

= ¬ (y+1=y' ∧ x+y=x')

wlp(y:=y+1, ¬(x=x' ∧ y=y'))

= let y = y+1 in ¬(y=y' ∧ x=x')

= ¬ (y+1=y ∧ x=x')

Horn Clauses by Dual WLP

Assumptions

- each procedure is represent by a control flow graph
  - i.e., statements of the form `l`$_i$`:S ; goto l`$_j$, where S is loop-free
- program is unsafe iff the last statement of `Main()` is reachable
  - i.e., no explicit assertions. All assertions are top-level.

For each procedure `P(x)`, create predicates

- `l(w)` for each label (i.e., basic block)
  - $p_{en}(x_0, x)$ for entry location of procedure p()
  - $p_{ex}(x_0, r)$ for exit location of procedure p()
- p(x,r) for each procedure P(x):r

UNIVERSITY OF WATERLOO

Horn Clauses by Dual WLP

The verification condition is a conjunction of clauses:

$p_{en}(x_0,x) \leftarrow x_0=x$

$l_j(x_0,w') \leftarrow l_i(x_0,w) \wedge \neg wlp\ (S, \neg(w=w'))$

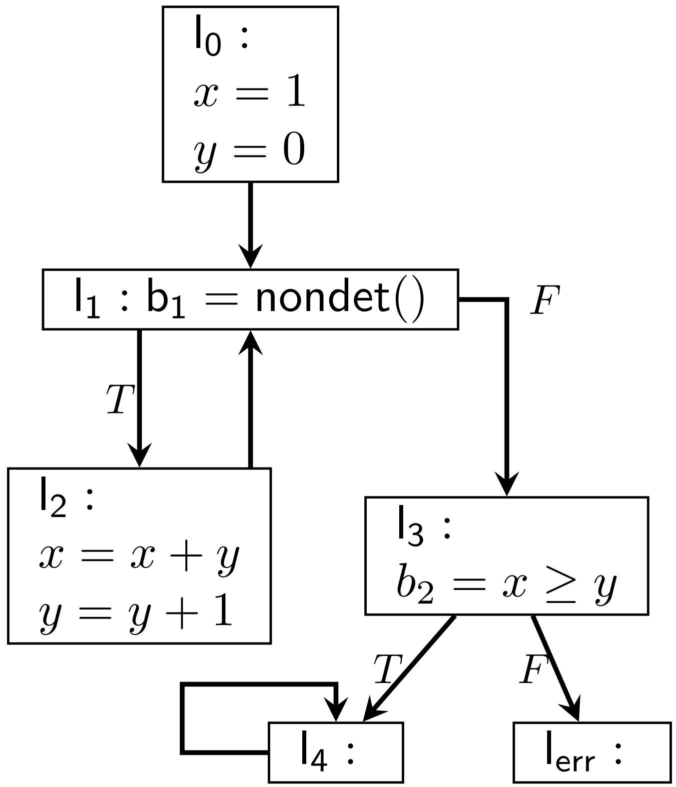
- for each statement $\texttt{l}_\texttt{i}\texttt{: S; goto l}_\texttt{j}$

$p\ (x_0,r) \leftarrow p_{ex}(x_0,r)$

$false \leftarrow Main_{ex}(x, ret)$

Example Horn Encoding

$\text{int } x = 1;$
$\text{int } y = 0;$
$\text{while } (*)\ \{$
$\quad x = x + y;$
$\quad y = y + 1;$
$\}$
$\text{assert}(x \geq y);$



$\langle 1 \rangle$ $\mathsf{p_0}.$
$\langle 2 \rangle$ $\mathsf{p_1}(x, y) \leftarrow \mathsf{p_0}, x = 1, y = 0.$
$\langle 3 \rangle$ $\mathsf{p_2}(x, y) \leftarrow \mathsf{p_1}(x, y)\ .$
$\langle 4 \rangle$ $\mathsf{p_3}(x, y) \leftarrow \mathsf{p_1}(x, y)\ .$
$\langle 5 \rangle$ $\mathsf{p_1}(x', y') \leftarrow \mathsf{p_2}(x, y), x' = x + y, y' = y + 1.$
$\langle 6 \rangle$ $\mathsf{p_4} \leftarrow (x \geq y), \mathsf{p_3}(x, y).$
$\langle 7 \rangle$ $\mathsf{p_{err}} \leftarrow (x < y), \mathsf{p_3}(x, y).$
$\langle 8 \rangle$ $\mathsf{p_4} \leftarrow \mathsf{p_4}.$
$\langle 9 \rangle$ $\perp \leftarrow \mathsf{p_{err}}.$

UNIVERSITY OF WATERLOO

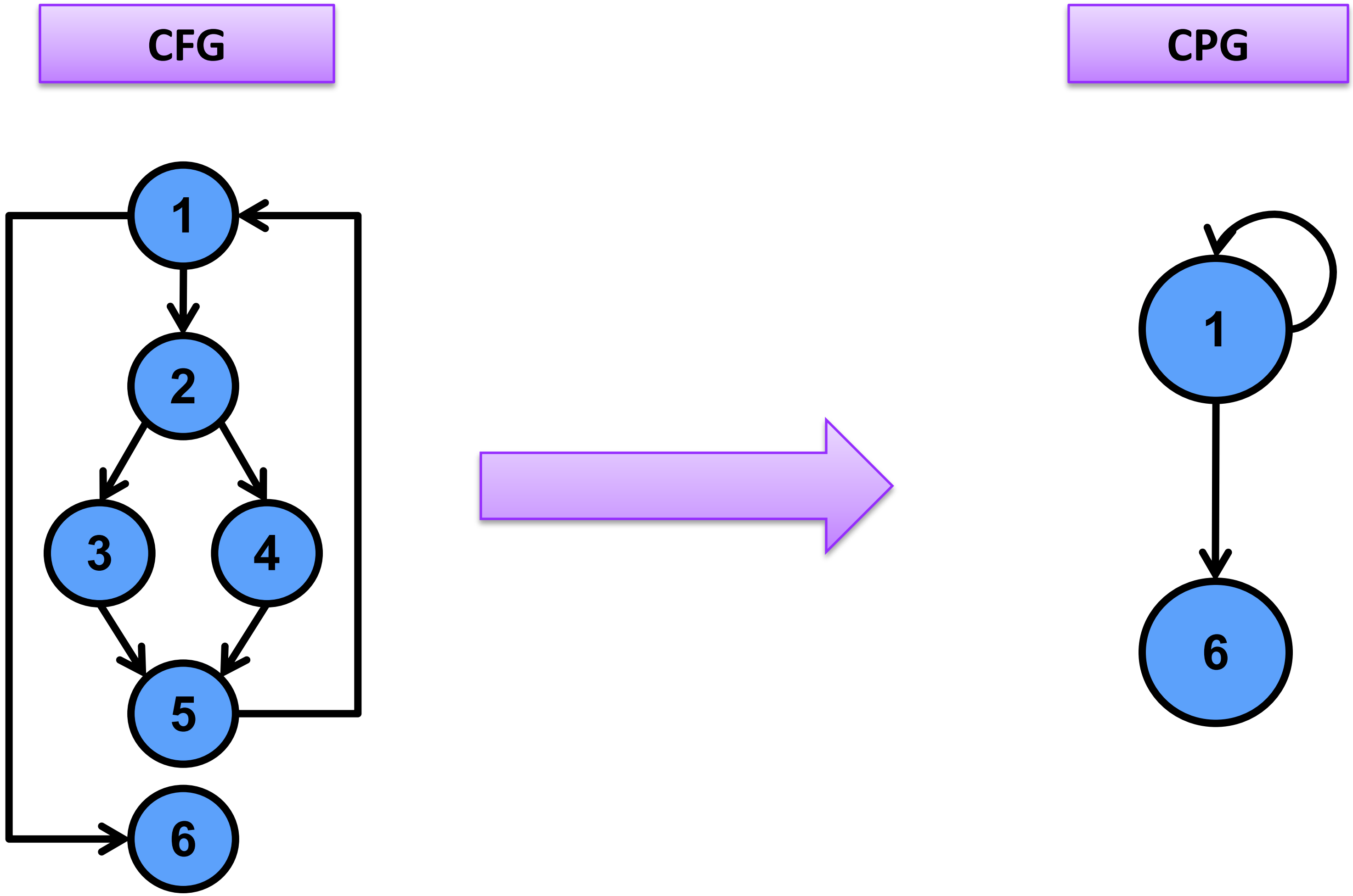
From CFG to Cut Point Graph

A *Cut Point Graph* hides (summarizes) fragments of a control flow graph by (summary) edges

Vertices (called, *cut points*) correspond to *some* basic blocks

An edge between cut-points *c* and *d* summarizes all finite (loop-free) executions from *c* to *d* that do not pass through any other cut-points

Cut Point Graph Example



From CFG to Cut Point Graph

A *Cut Point Graph* hides (summarizes) fragments of a control flow graph by (summary) edges

Cut Point Graph preserves reachability of (not-summarized) control location.

Summarizing loops is undecidable! (Halting program)

A *cutset summary* summarizes all location except for a *cycle cutset* of a CFG. Computing minimal cutset summary is NP-hard (minimal feedback vertex set).

A reasonable compromise is to summarize everything but heads of loops. (Polynomial-time computable).

Single Static Assignment

SSA == every value has a unique assignment (a *definition*)

A procedure is in SSA form if every variable has exactly one definition

SSA form is used by many compilers

- explicit def-use chains
- simplifies optimizations and improves analyses

PHI-function are necessary to maintain unique definitions in branching control flow

$x = PHI (v_0{:}bb_0, \ldots, v_n{:}bb_n))$ (phi-assignment)

“x gets v_i if previously executed block was bb_i”

UNIVERSITY OF WATERLOO

Single Static Assignment: An Example

val:bb

```
int x, y, n;

x = 0;
while (x < N) {
  if (y > 0)
    x = x + y;
  else
    x = x - y;
  y = -1 * y;
}
```

```
0: goto 1
1: x_0 = PHI(0:0, x_3:5);
   y_0 = PHI(y:0, y_1:5);
   if (x_0 < N) goto 2 else goto 6

2: if (y_0 > 0) goto 3 else goto 4


3: x_1 = x_0 + y_0; goto 5

4: x_2 = x_0 - y_0; goto 5


5: x_3 = PHI(x_1:3, x_2:4);
   y_1 = -1 * y_0;
   goto 1
6:
```

Large Step Encoding

Problem: Generate a compact verification condition for a loop-free block of code

```
0: goto 1
1: x_0 = PHI(0:0, x_3:5);
   y_0 = PHI(y:0, y_1:5);
   if (x_0 < N) goto 2 else goto 6

2: if (y_0 > 0) goto 3 else goto 4

3: x_1 = x_0 + y_0; goto 5

4: x_2 = x_0 - y_0; goto 5

5: x_3 = PHI(x_1:3, x_2:4);
   y_1 = -1 * y_0;
   goto 1
6:
```

Large Step Encoding: Extract all Actions

$x_1 = x_0 + y_0$
$x_2 = x_0 - y_0$
$y_1 = -1 * y_0$

```
1: x_0 = PHI(0:0, x_3:5);
   y_0 = PHI(y:0, y_1:5);
   if (x_0 < N) goto 2 else goto 6

2: if (y_0 > 0) goto 3 else goto 4

3: x_1 = x_0 + y_0; goto 5

4: x_2 = x_0 - y_0; goto 5

5: x_3 = PHI(x_1:3, x_2:4);
   y_1 = -1 * y_0;
   goto 1
```

Example: Encode Control Flow

$x_1 = x_0 + y_0$
$x_2 = x_0 - y_0$
$y_1 = -1 * y_0$

$B_2 \rightarrow x_0 < N$

$B_3 \rightarrow B_2 \wedge y_0 > 0$

$B_4 \rightarrow B_2 \wedge y_0 \leq 0$
$B_5 \rightarrow (B_3 \wedge x_3{=}x_1) \vee (B_4 \wedge x_3{=}x_2)$

$B_5 \wedge x'_0{=}x_3 \wedge y'_0{=}y_1$

$p_1(x'_0, y'_0) \leftarrow p_1(x_0, y_0), \phi.$

```
1: x_0 = PHI(0:0, x_3:5);
   y_0 = PHI(y:0, y_1:5);
   if (x_0 < N) goto 2 else goto 6

2: if (y_0 > 0) goto 3 else goto 4

3: x_1 = x_0 + y_0; goto 5

4: x_2 = x_0 - y_0; goto 5

5: x_3 = PHI(x_1:3, x_2:4);
   y_1 = -1 * y_0;
   goto 1
```

Summary

Convert body of each procedure into SSA

For each procedure, compute a Cut Point Graph (CPG)

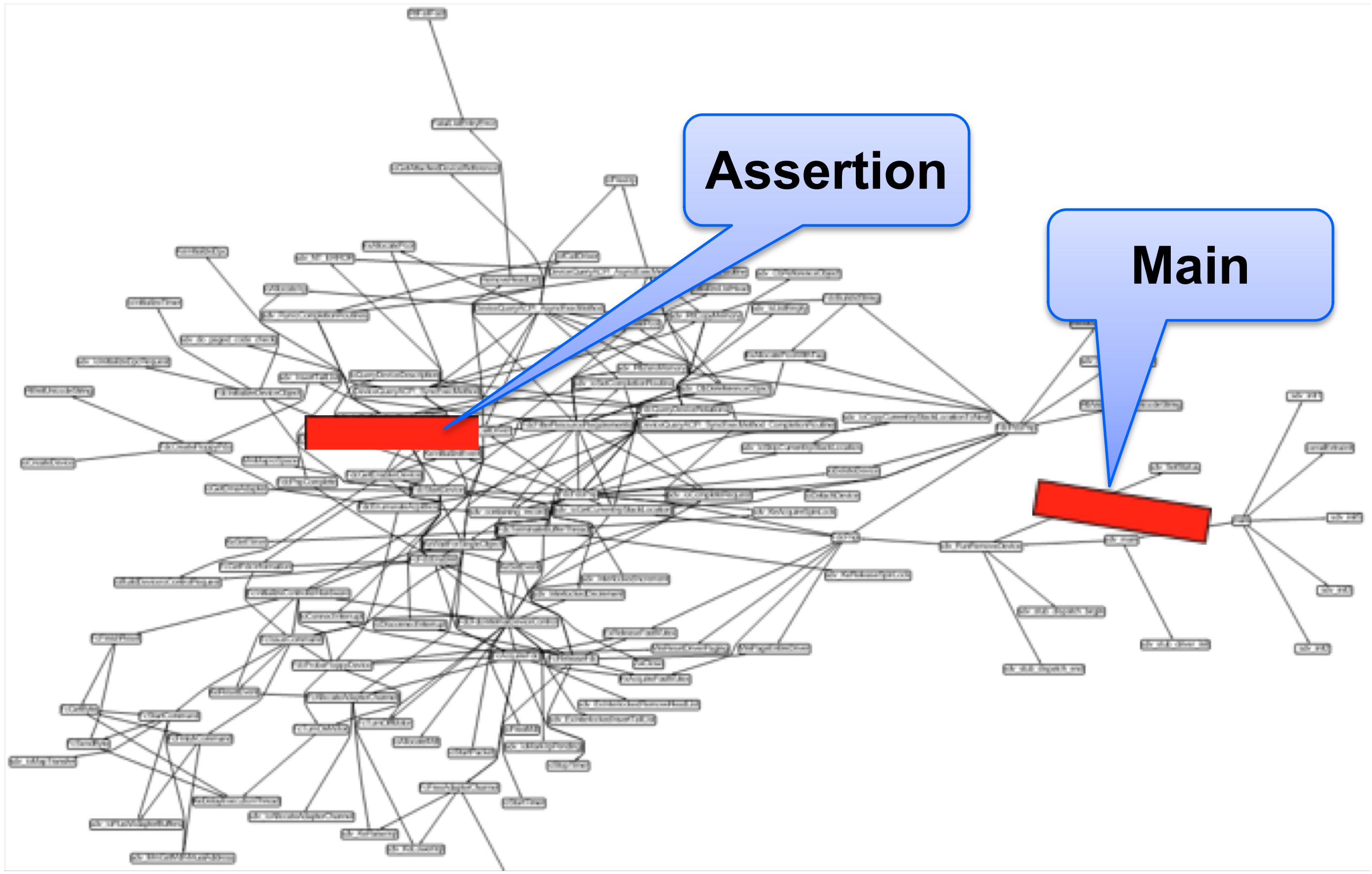
For each edge (s, t) in CPG use dual-wlp to construct the constraint for an execution to flow from s to t

Procedure summary is determined by constraints at the exit point of a procedure

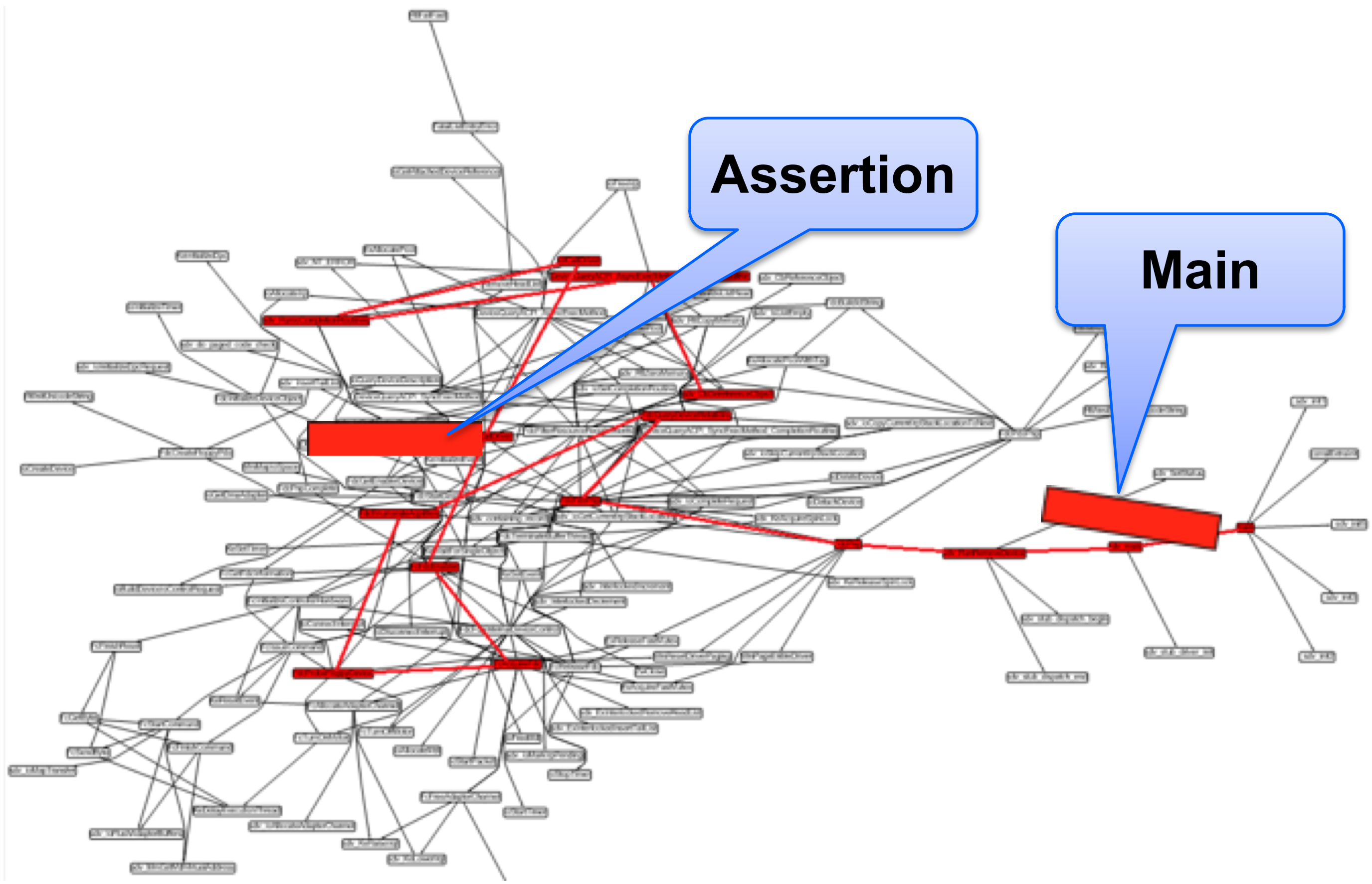
Mixed Semantics

PROGRAM TRANSFORMATION

Deeply nested assertions



Deeply nested assertions



Counter-examples are long

Hard to determine (from main) what is relevant

Mixed Semantics

[GWC'08,LQ'14]

Stack-free program semantics combining:

- operational (or small-step) semantics
  - i.e., usual execution semantics
- natural (or big-step) semantics: function summary [Sharir-Pnueli 81]
  - $(\sigma, \sigma`) \in ||f||$ iff the execution of f on input state σ terminates and results in state σ'
- some execution steps are big, some are small

Non-deterministic executions of function calls

- update top activation record using function summary, or
- enter function body, forgetting history records (i.e., no return!)

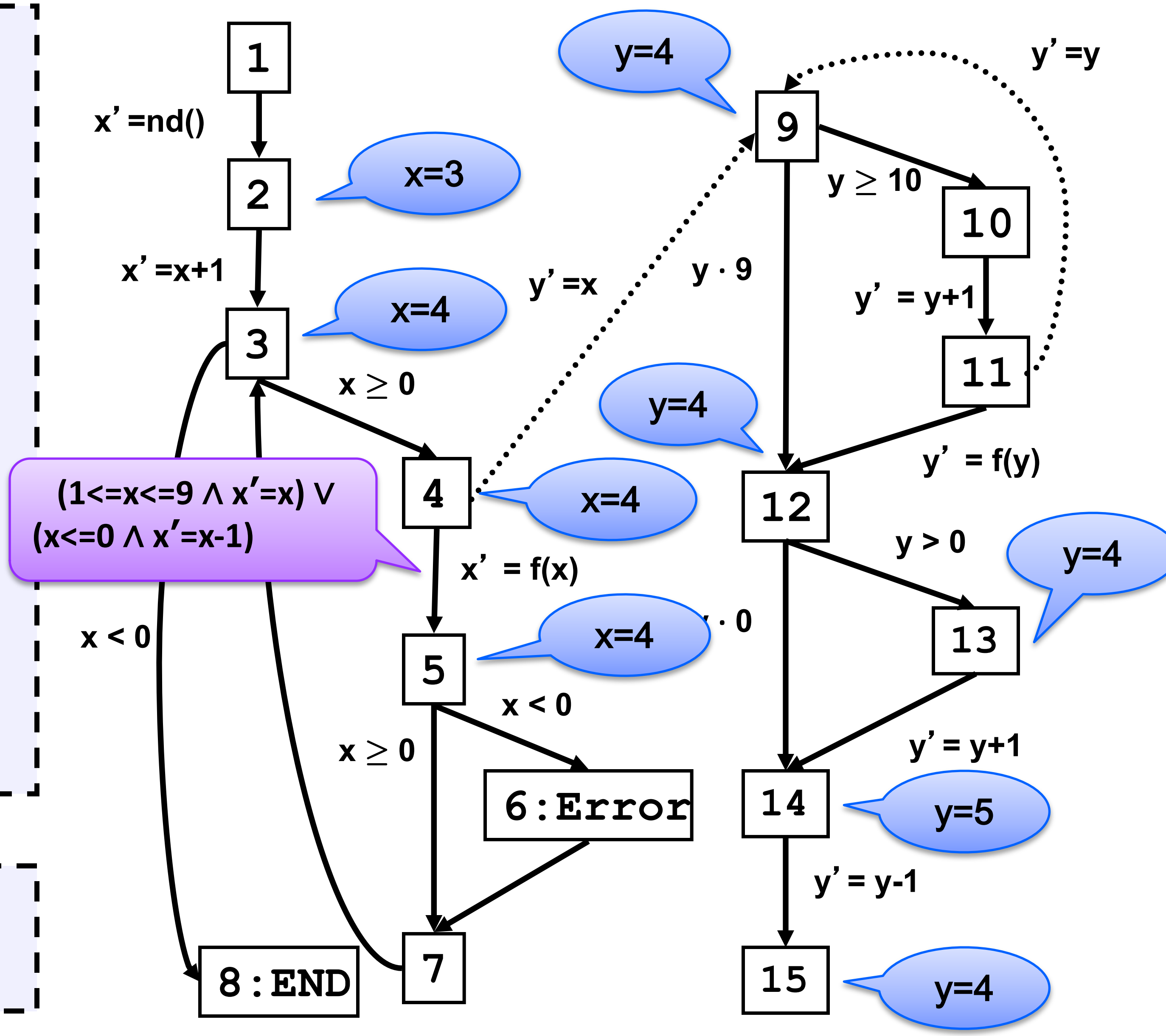
Preserves reachability and non-termination

<u>Theorem:</u> Let K be the operational semantics, K^m the stack-free semantics, and L a program location. Then,

$K \models EF\,(pc=L) \Leftrightarrow K^m \models EF\,(pc=L)$ and $K \models EG\,(pc\neq L) \Leftrightarrow K^m \models EG\,(pc\neq L)$

UNIVERSITY OF WATERLOO

```
def main()
1: int x = nd();
2: x = x+1;
3: while(x>=0)
4:   x=f(x);
5:   if(x<0)
6:       Error;
7:
8: END;

def f(int y): ret y
9:  if(y>=10){
10:     y=y+1;
11:     y=f(y);
12: else if(y>0)
13:    y=y+1;
14: y=y-1
15:
```



Summary of f(y)

$(1<=y<=9 \wedge y'=y) \vee$
$(y<=0 \wedge y'=y-1)$

Mixed Semantics Transformation via Inlining

```
void main() {
  p1(); p2();
  assert(c1);
}
void p1() {
  p2();
  assert(c2);
}
void p2() {
  assert(c3);
}
```

```
void main() {
  if(nd()) p1(); else goto p1;
  if(nd()) p2(); else goto p2;
  assert(c1);
  assume(false);
  p1: if (nd) p2(); else goto p2;
  assume(!c2);
  assert(false);
  p2: assume(!c3);
  assert(false);
}
```

```
void p1() {p2(); assume(c2);}
void p2() {assume(c3);}
```

UNIVERSITY OF WATERLOO

Mixed Semantics: Summary

Every procedure is inlined at most once

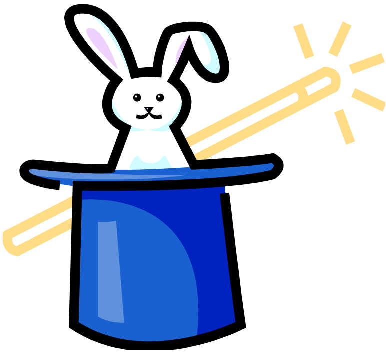
- in the worst case, doubles the size of the program
- can be restricted to only inline functions that directly or indirectly call errror() function

Easy to implement at compiler level

- create “failing” and “passing” versions of each function
- reduce “passing” functions to returning paths
- in main(), introduce new basic block bb.F for every failing function F(), and call failing.F in bb.F
- inline all failing calls
- replace every call to F to non-deterministic jump to bb.F or call to passing F

Increases context-sensitivity of context-insensitive analyses

- context of failing paths is explicit in main (because of inlining)
- enables / improves many traditional analyses



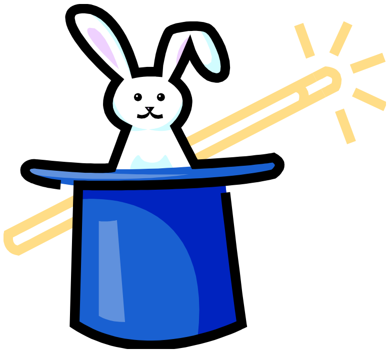
SOLVING CONSTRAINED HORN CLAUSES

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

$$
\begin{aligned}
Init(X) &\rightarrow P(X) \\
P(X) \land Tr(X, X') &\rightarrow P(X') \\
P(X) &\rightarrow \neg Bad(X)
\end{aligned}
$$

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

Proof:

add extra arguments to distinguish between predicates

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

University of Waterloo

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

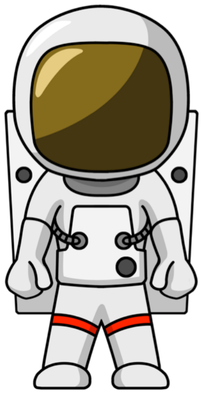
uPDR: Constraints in EPR fragment of FOL

- 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.

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
```

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

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

Theory dependent

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 $L \wedge R_i \wedge Tr \Rightarrow L'$

Induction

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

UNIVERSITY OF WATERLOO

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

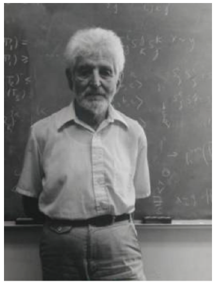
$$\varphi' \Rightarrow \neg c'$$

Looking for ϕ'

ARITHMETIC CONFLICT

UNIVERSITY OF WATERLOO

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)

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

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

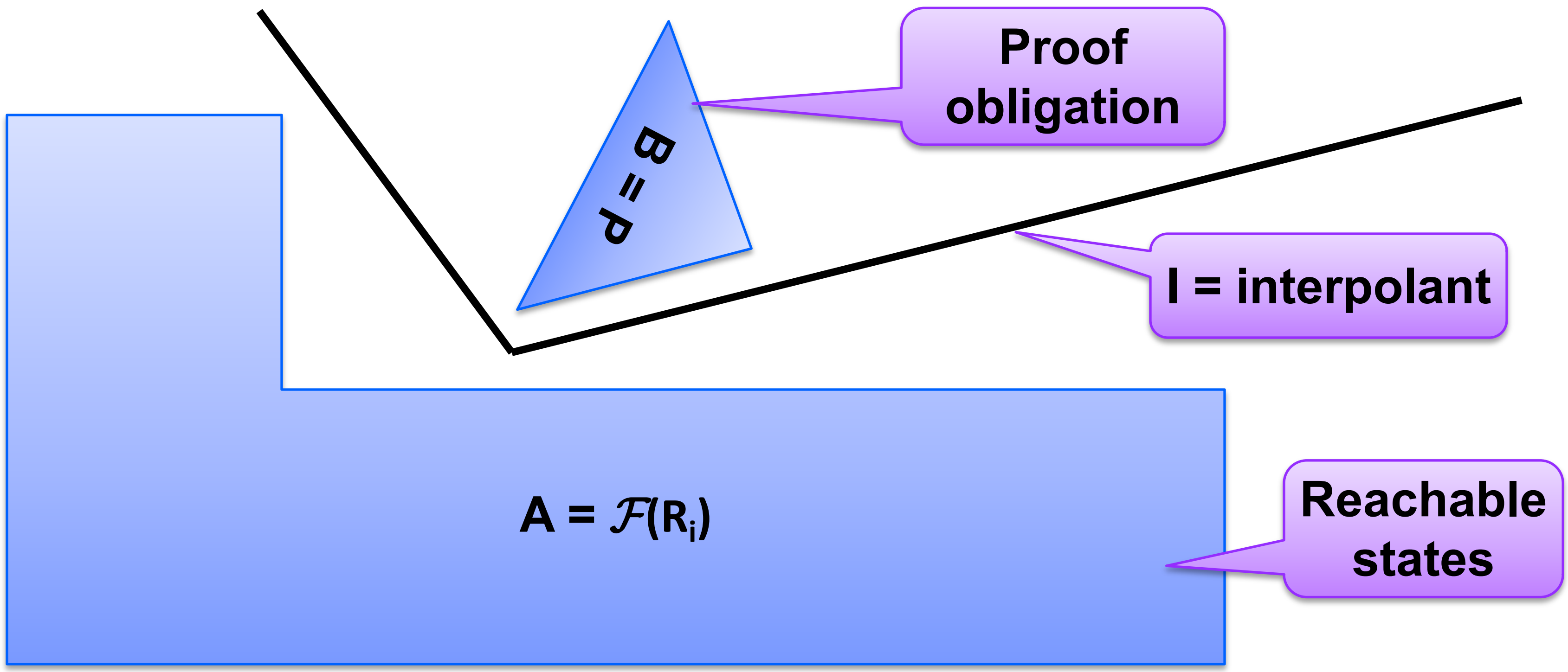
Linear Real Arithmetic (LRA)

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

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

UNIVERSITY OF WATERLOO

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

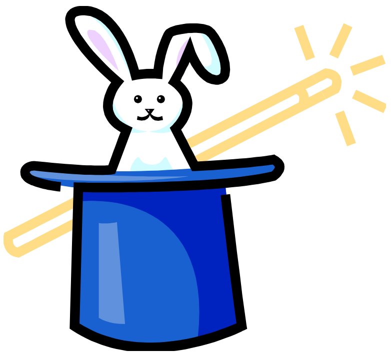
Conflict For $0 \leq i < N$, given a counterexample $\langle P, i+1 \rangle \in Q$ s.t. $\mathcal{F}(F_i) \wedge P'$ is unsatisfiable, add $P^{\uparrow} = \textsc{Itp}(\mathcal{F}(F_i), P')$ to F_j for $j \leq i+1$.

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



UNIVERSITY OF WATERLOO

Computing Interpolants for IC3/PDR

Much simpler than general interpolation problem for A ∧ 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

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

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

$$\overline{}$$

$$0 > 13$$

UNIVERSITY OF WATERLOO

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)

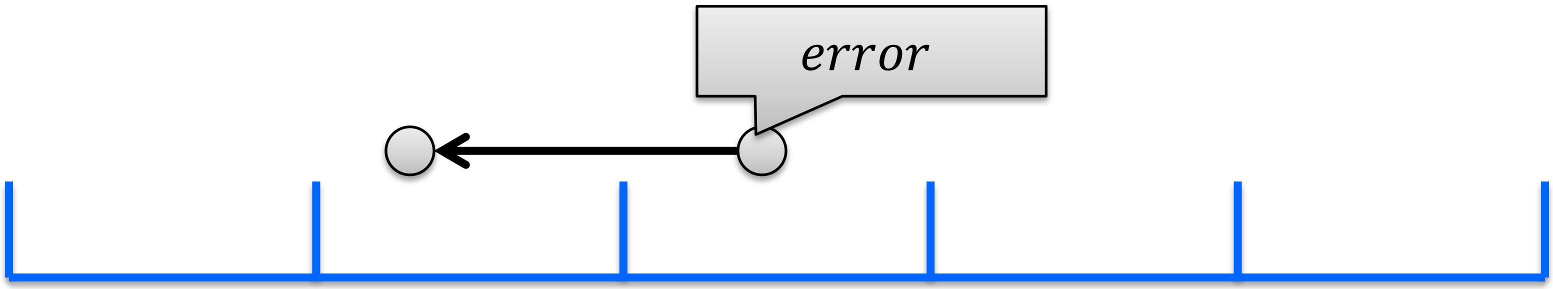
```
z = x; i = 0;
assume (y > 0);
while (i < y) {
  z = z + 1;
  i = i + 1;
}
assert(z == x + y);
```

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
```

Lemma Generation Example

MkSafe



Transition Relation | **Pob**

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

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

Farkas explanation for unsat

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

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

Computing Interpolants in Spacer

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

- 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 $(\land B_i \Rightarrow \lor A_j)$

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

UNIVERSITY OF WATERLOO

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

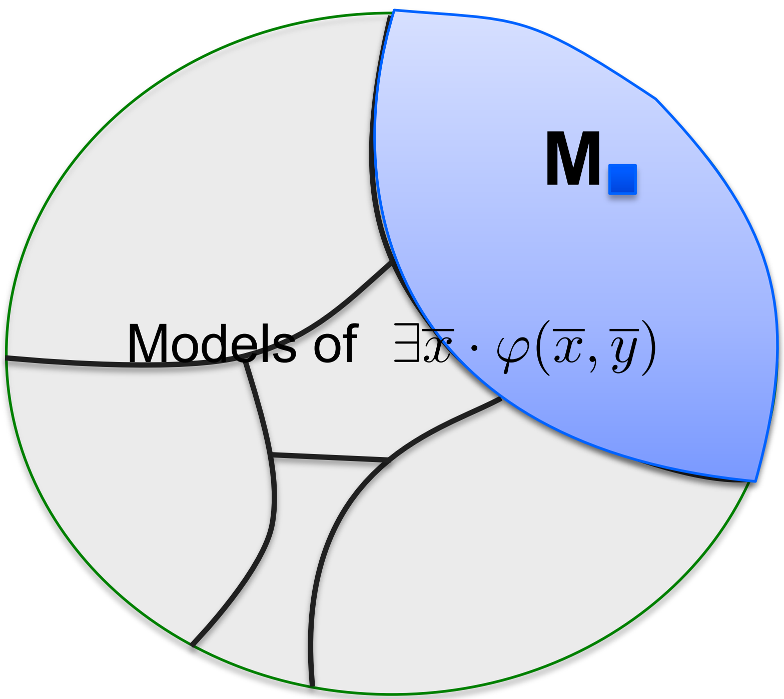
1. ψ is a monomial
2. Vars(ψ) $\subseteq$ Vars(ϕ) $\setminus$ U
3. M $\models \psi$
4. $\psi \Rightarrow \exists$ U . ϕ

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

UNIVERSITY OF WATERLOO

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

UNIVERSITY OF WATERLOO

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 $\exists\, x\, \psi(x) \Leftrightarrow \varphi$

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

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

The case of lower bounds is dual

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

UNIVERSITY OF WATERLOO

Fourier–Motzkin Quantifier Elimination for LRA

$$\begin{aligned} & \exists x \cdot \bigwedge_i s_i < x \land \bigwedge_j x < t_j \\ = \quad & \bigwedge_i \bigwedge_j resolve(s_i < x, x < t_j, x) \\ = \quad & \bigwedge_i \bigwedge_j s_i < t_j \end{aligned}$$

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

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

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. $(F_i \land Tr \land cex')$, 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

UNIVERSITY OF WATERLOO

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

$$
\begin{aligned}
Init(X) &\rightarrow P(X) \\
P(X) \wedge P(X^o) \wedge Tr(X, X^o, X') &\rightarrow P(X') \\
P(X) &\rightarrow \neg Bad(X)
\end{aligned}
$$

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 $F_0, F_1, \ldots$

counterexample is a tree

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 $0 \le j \le i+1$.
- **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 $\varphi \notin F_{i+1}$, $\mathcal{F}(\phi \wedge F_i) \rightarrow \phi'$, then add φ to F_j, for all $j \le i+1$.

until ∞;

two predecessors

theory-aware **Conflict**

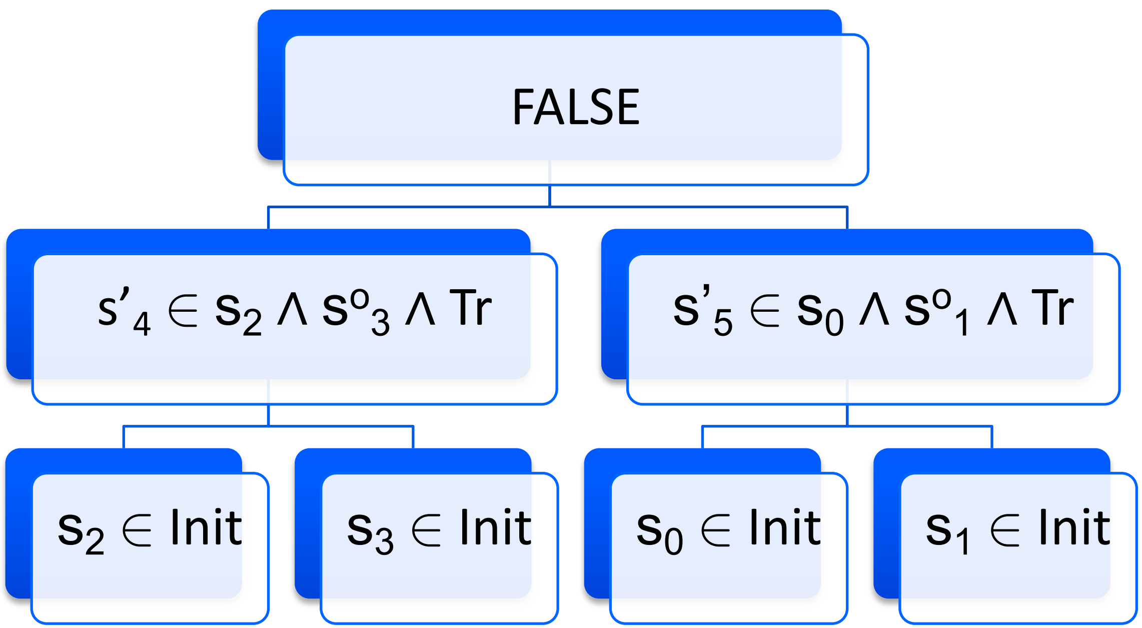
Counterexamples to non-linear CHC

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

- we call such a derivation a counterexample

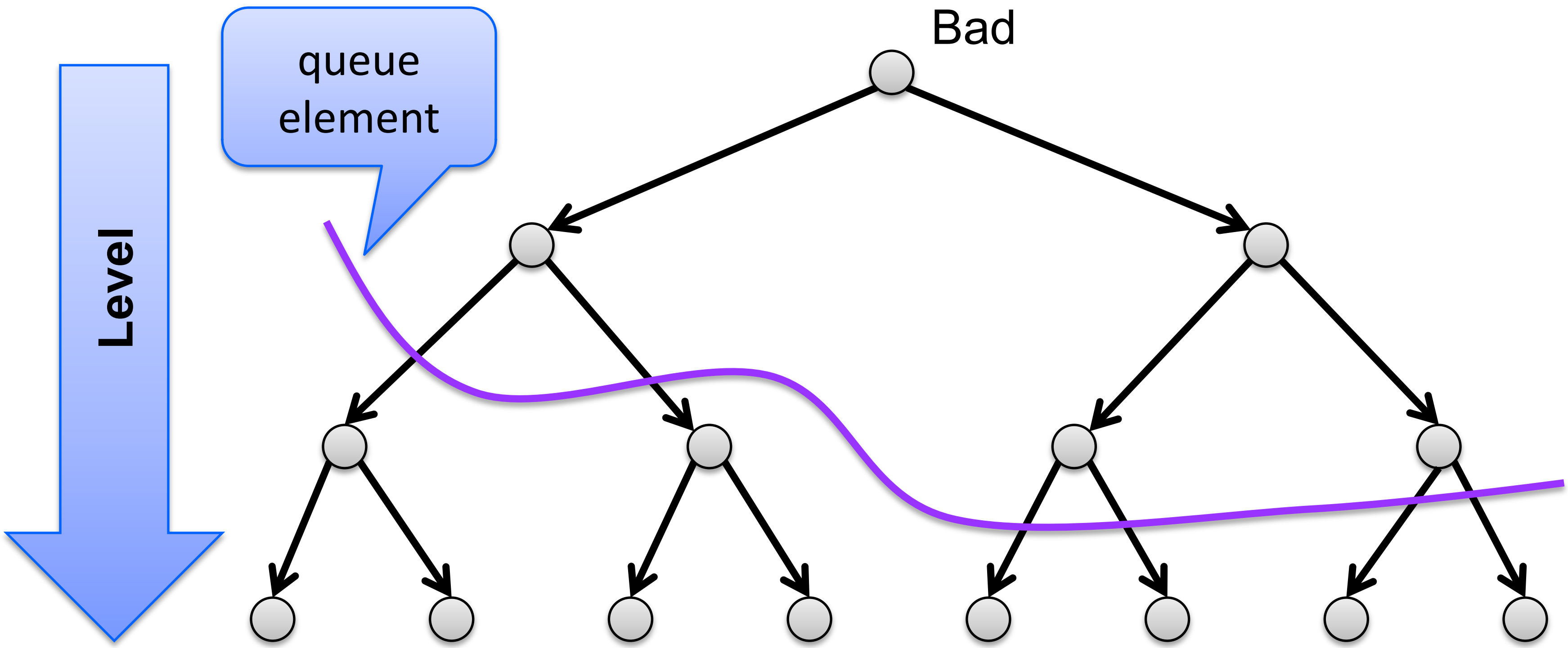
For linear CHC, the counterexample is a path

For non-linear CHC, the counterexample is a tree



UNIVERSITY OF WATERLOO

GPDR Search Space



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

UNIVERSITY OF WATERLOO

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

UNIVERSITY OF WATERLOO

Spacer

Same queue as in IC3/PDR

Cache Reachable states

Three variants of **Decide**

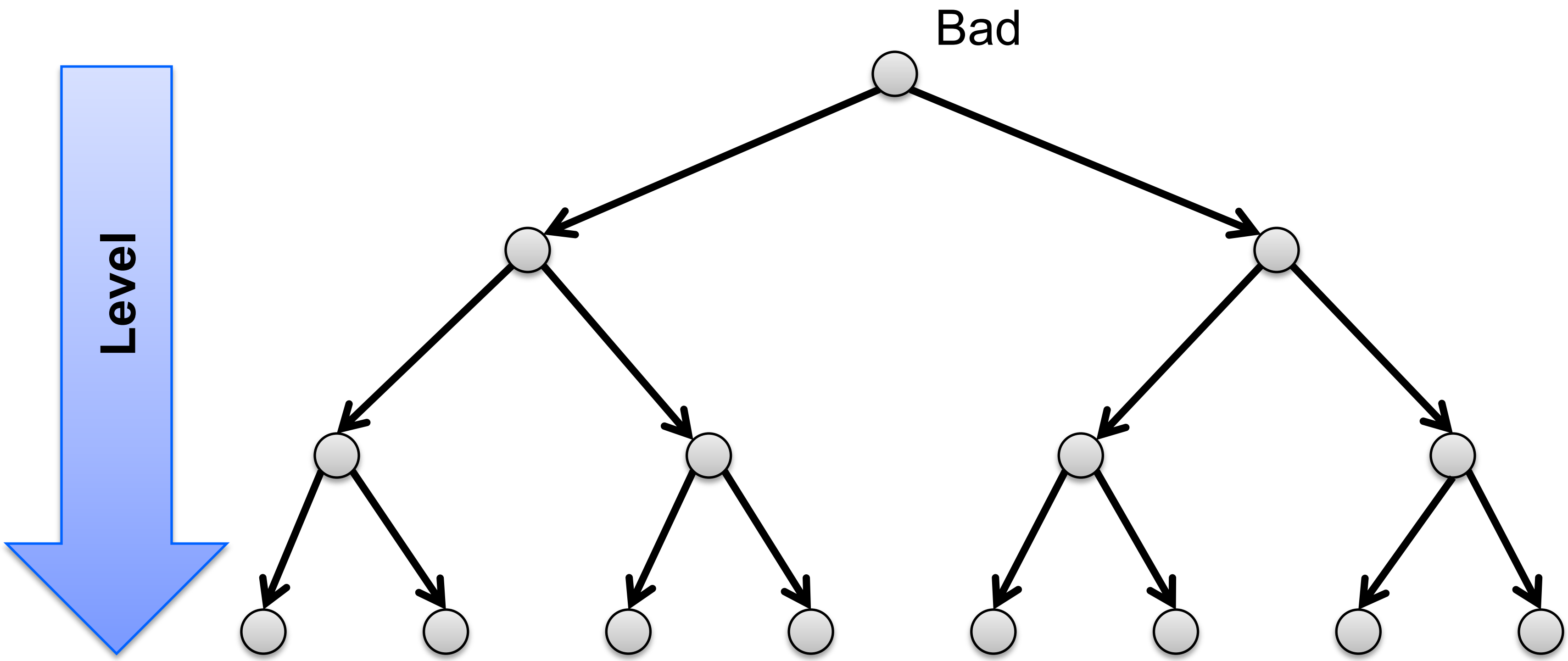
Same **Conflict** as in APDR/GPDR

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 $\textsc{Reach}$. 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: $Q = \emptyset$, $N = 0$, $F_0 = Init$, $\forall i > 0 \cdot F_i = \emptyset$, $\textsc{Reach} = Init$
Require: $Init \to \neg Bad$
repeat

- **Unreachable** If there is an $i < N$ s.t. $F_i \subseteq F_{i+1}$ **return** *Unreachable*.
- **Reachable** If $\textsc{Reach} \wedge Bad$ is satisfiable, **return** *Reachable*.
- **Unfold** If $F_N \to \neg Bad$, then set $N \leftarrow N + 1$ and $Q \leftarrow \emptyset$.
- **Candidate** If for some m, $m \to F_N \wedge Bad$, then add $\langle m, N\rangle$ to Q.
- **Successor** If there is $\langle m, i+1\rangle \in Q$ and a model M $M \models \psi$, 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 $0 \le j \le i+1$.
- **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 $0 \le i < N$ 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 ∞;

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 $s' \in \mathrm{MBP}(\{X, X^o\}, \psi)$.

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**)