Solving Constrained Horn Clauses with SMT

Arie Gurfinkel

International Summer School on Satisfiability, Satisfiability Modulo Theories, and Automated Reasoning

SAT/SMT/AR 2018

July 6, 2018

UNIVERSITY OF WATERLOO

Automated Verification

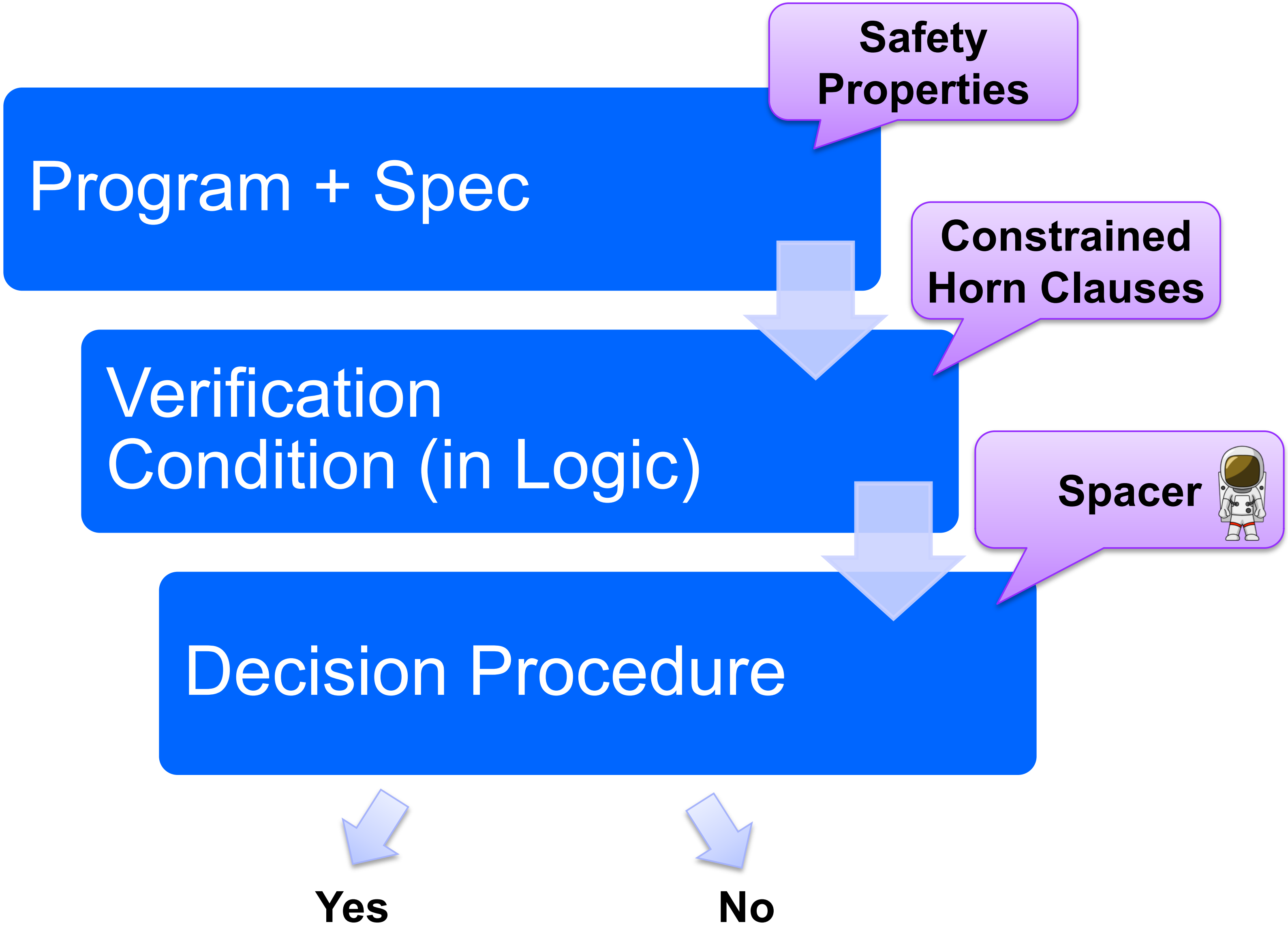
Deductive Verification

- A user provides a program and a verification certificate
  - e.g., inductive invariant, pre- and post-conditions, function summaries, etc.
- A tool automatically checks validity of the certificate
  - this is not easy! (might even be undecidable)
- Verification is manual but machine certified

Algorithmic Verification (My research area)

- A user provides a program and a desired specification
  - e.g., program never writes outside of allocated memory
- A tool automatically checks validity of the specification
  - and generates a verification certificate if the program is correct
  - and generates a counterexample if the program is not correct
- Verification is completely automatic – “push-button”

Algorithmic Logic-Based Verification



Program + Spec

Verification Condition (in Logic)

Decision Procedure

Yes

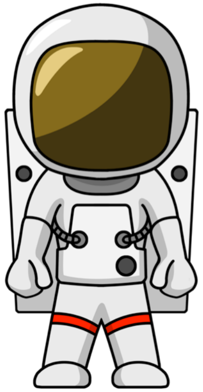
No

Safety Properties

Constrained Horn Clauses

Spacer

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 (work in progress)*
- Best-effort support for many other SMT-theories
  - data-structures, bit-vectors, non-linear arithmetic

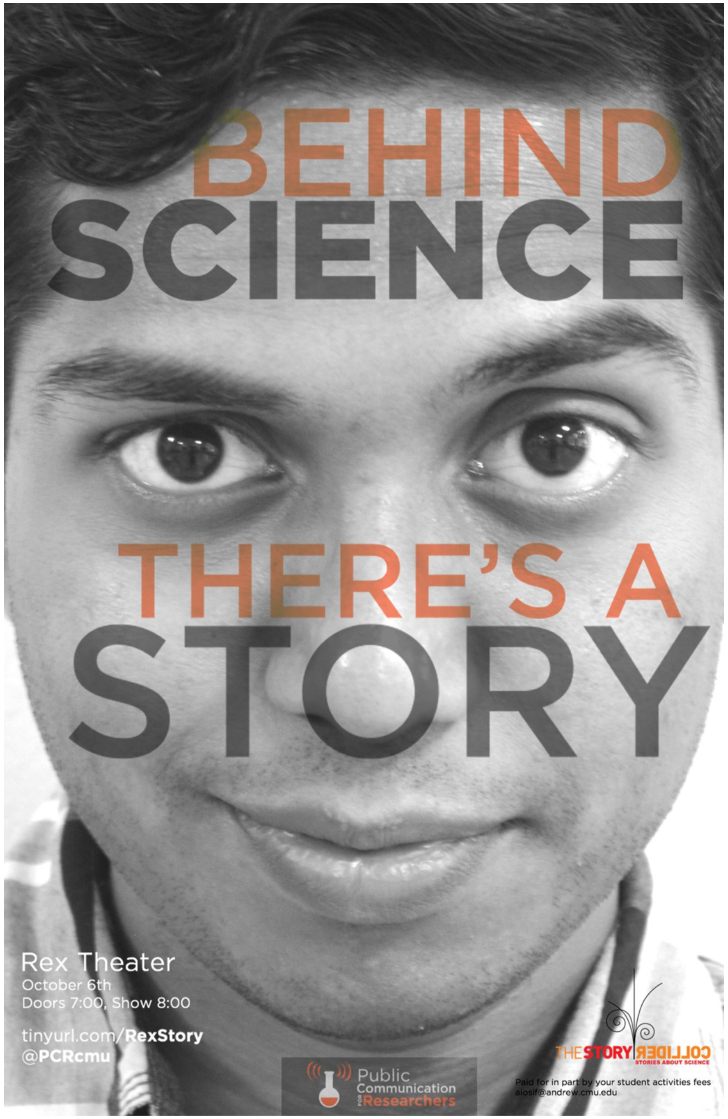
Support for Non-Linear CHC

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

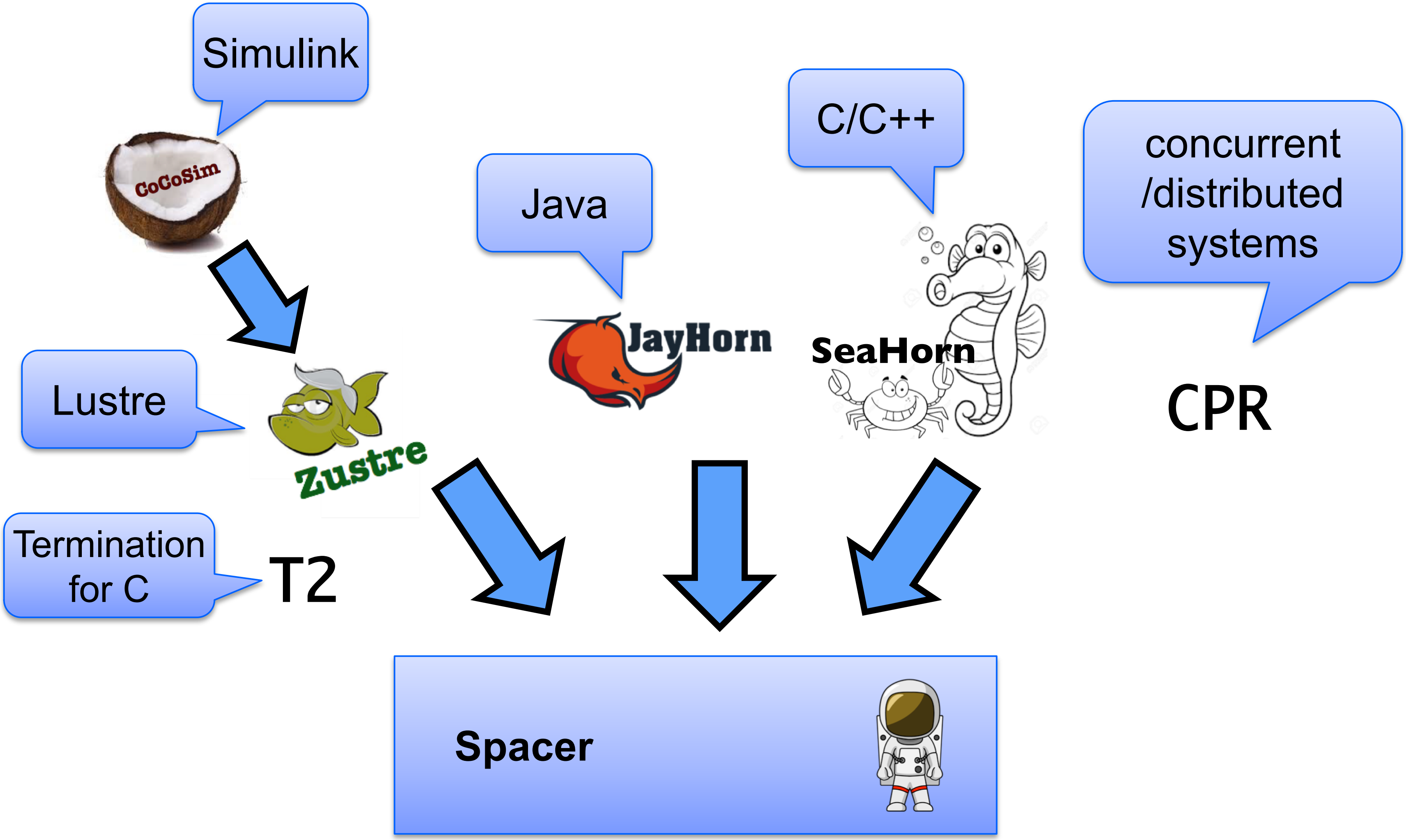
Contributors

Arie Gurfinkel
Anvesh Komuravelli

Nikolaj Bjorner
(Krystof Hoder)
Yakir Vizel
Bernhard Gleiss
Matteo Marescotti



Logic-based Algorithmic Verification



Simulink

Java

C/C++

concurrent /distributed systems

Lustre

Termination for C

T2

CPR

Spacer

Constrained Horn Clauses (CHC)

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

$$\forall V \,.\, (\phi \wedge p_1[X_1] \wedge \ldots \wedge p_n[X_n] \rightarrow h[X]),$$

where

- A is a background theory (e.g., Linear Arithmetic, Arrays, Bit-Vectors, or combinations of the above)
- ϕ is a constrained in the background theory A
- $p_1, \ldots, p_n$, h are n-ary predicates
- $p_i[X]$ is an application of a predicate to first-order terms

CHC Notation and T holog

head

body

constraint

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

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 $n > 1$

CHC Satisfiability

A **model** of a set of clauses Π is an extension of the model of the background theory with an interpretation of each predicate p_i that makes all clauses in Π valid

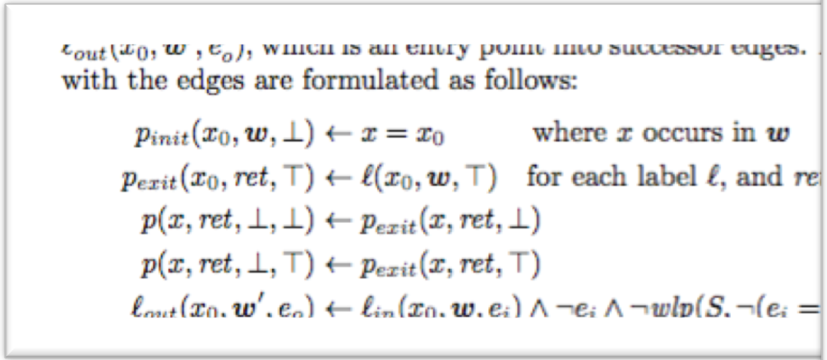
A set of clauses is **satisfiable** if it has a model, and is unsatisfiable otherwise

Given a theory A, a model M is **A-definable**, it each p_i in M is definable by a formula ψ_i in A

In the context of program verification

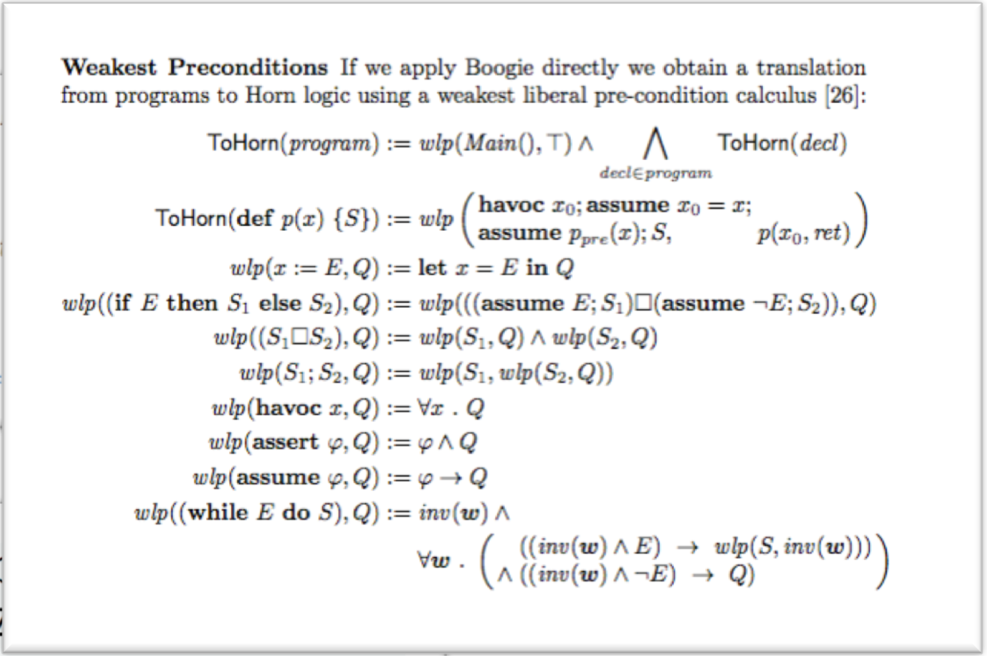
- a program satisfies a property iff corresponding CHCs are satisfiable
- verification certificates correspond to models
- counterexamples correspond to derivations of false

Horn Clauses for Program Verification



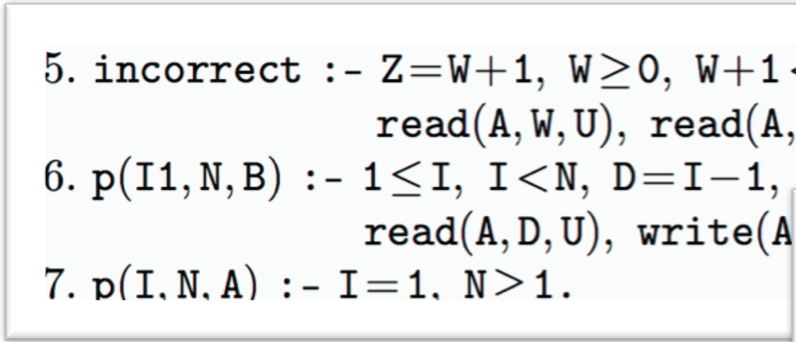
$\ell_{out}(x_0, \boldsymbol{w}', e_o)$, which is an entry point into successor edges.
with the edges are formulated as follows:

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



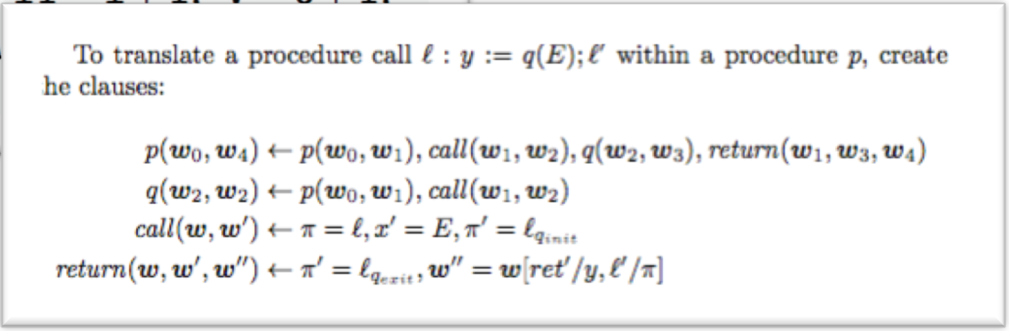
Weakest Preconditions If we apply Boogie directly we obtain a translation from programs to Horn logic using a weakest liberal pre-condition calculus [26]:

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



```
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



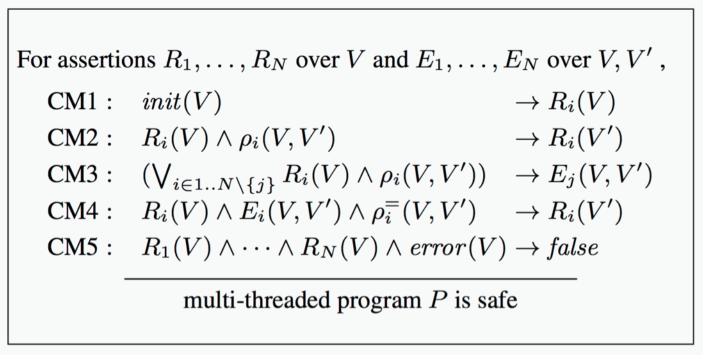
To translate a procedure call $\ell : y := q(E); \ell'$ within a procedure p, create he clauses:

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

Bjørner, Gurfinkel, McMillan, and Rybalchenko:

Horn Clause Solvers for Program Verification

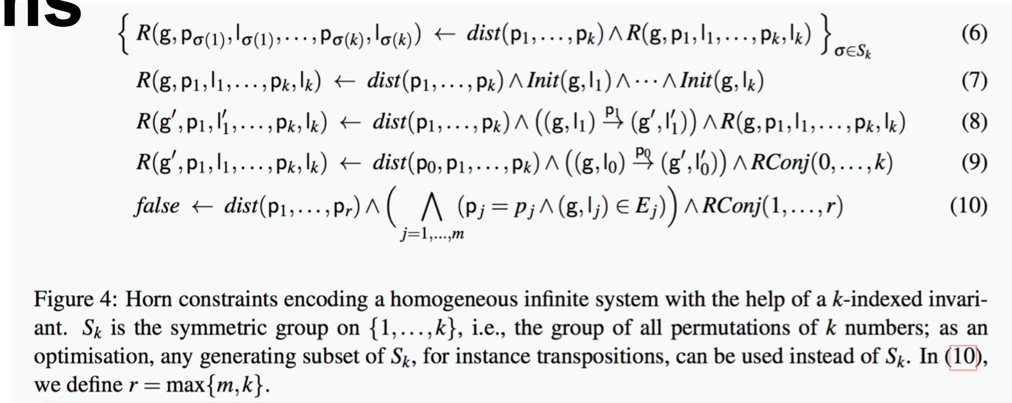
Horn Clauses for Concurrent / Distributed / Parameterized Systems



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

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

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



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

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

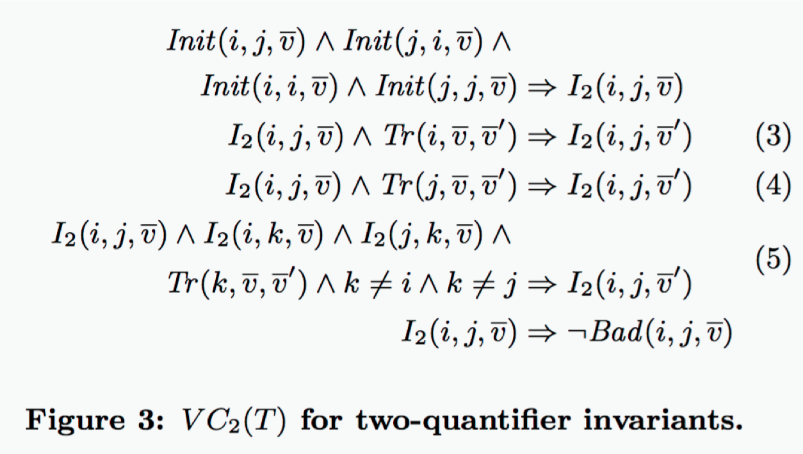
$$R(\mathsf{g}', \mathsf{p}_1, \mathsf{l}'_1, \dots, \mathsf{p}_k, \mathsf{l}_k) \leftarrow \mathit{dist}(\mathsf{p}_1, \dots, \mathsf{p}_k) \wedge \big((\mathsf{g}, \mathsf{l}_1) \xrightarrow{\mathsf{p}_1} (\mathsf{g}', \mathsf{l}'_1)\big) \wedge R(\mathsf{g}, \mathsf{p}_1, \mathsf{l}_1, \dots, \mathsf{p}_k,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

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

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

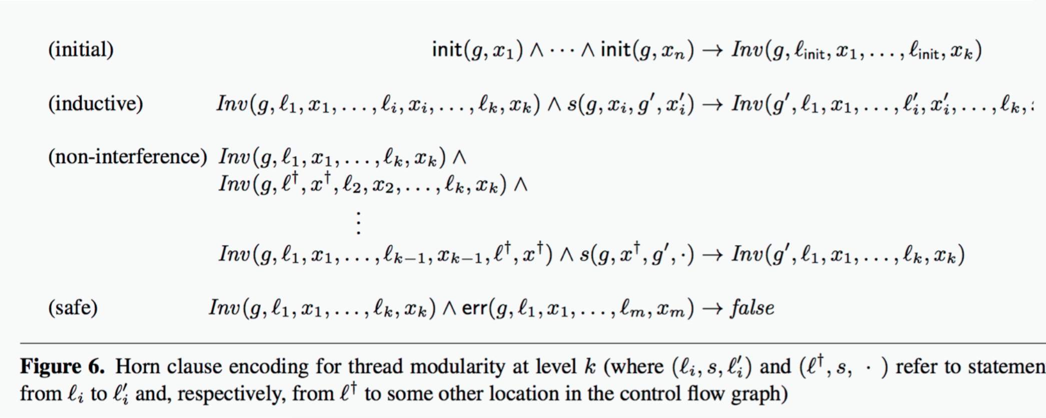
Figure 4: Horn constraints encoding a homogeneous infinite system with the help of a k-indexed invariant. S_k is the symmetric group on $\{1, \dots, k\}$, i.e., the group of all permutations of k numbers; as an optimisation, any generating subset of S_k, for instance transpositions, can be used instead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

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



$$\begin{array}{rr}
\mathit{Init}(i, j, \overline{v}) \wedge \mathit{Init}(j, i, \overline{v}) \wedge & \\
\mathit{Init}(i, i, \overline{v}) \wedge \mathit{Init}(j, j, \overline{v}) \Rightarrow I_2(i, j, \overline{v}) & \\
I_2(i, j, \overline{v}) \wedge \mathit{Tr}(i, \overline{v}, \overline{v}') \Rightarrow I_2(i, j, \overline{v}') & (3) \\
I_2(i, j, \overline{v}) \wedge \mathit{Tr}(j, \overline{v}, \overline{v}') \Rightarrow I_2(i, j, \overline{v}') & (4) \\
I_2(i, j, \overline{v}) \wedge I_2(i, k, \overline{v}) \wedge I_2(j, k, \overline{v}) \wedge & \\
\mathit{Tr}(k, \overline{v}, \overline{v}') \wedge k \neq i \wedge k \neq j \Rightarrow I_2(i, j, \overline{v}') & (5) \\
I_2(i, j, \overline{v}) \Rightarrow \neg \mathit{Bad}(i, j, \overline{v}) &
\end{array}$$

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



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

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

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

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

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

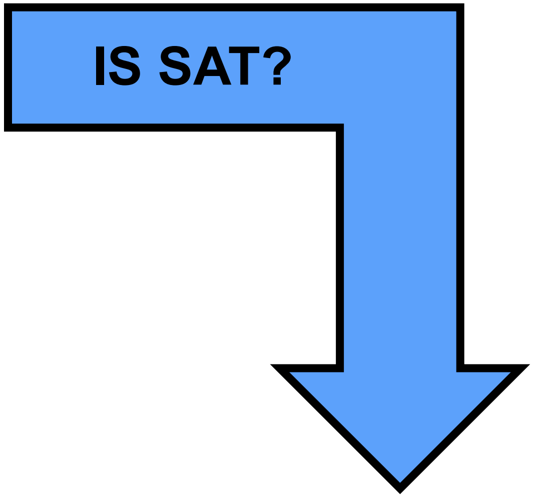
Gurfinkel et al. SMT-Based Verification of Parameterized Systems. FSE 2016

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

UNIVERSITY OF WATERLOO

Is this program correct?

```
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
```

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)

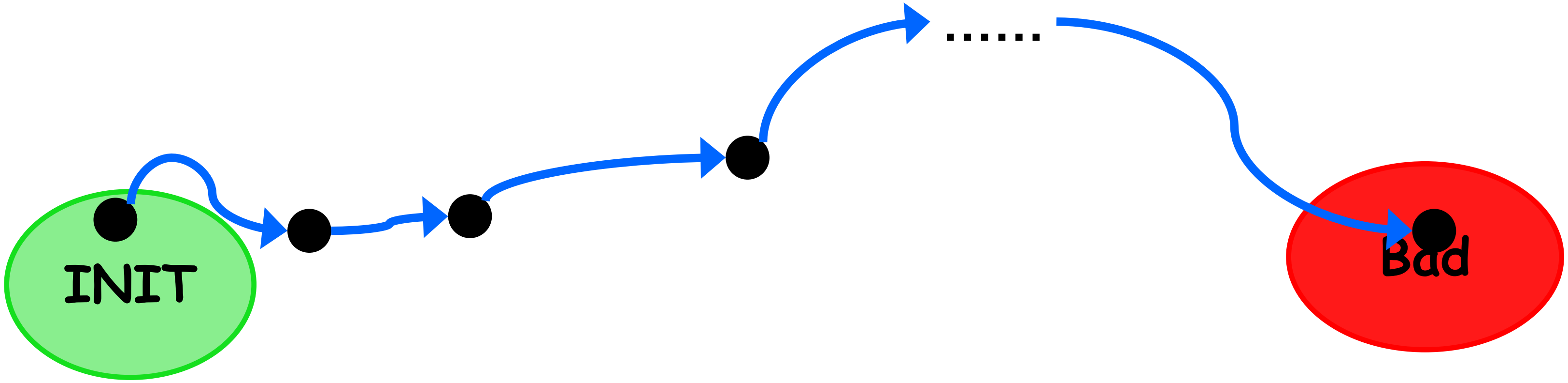
- Spacer, Implicit Predicate Abstraction

Safety Verification Problem

Is Bad reachable?

Safety Verification Problem

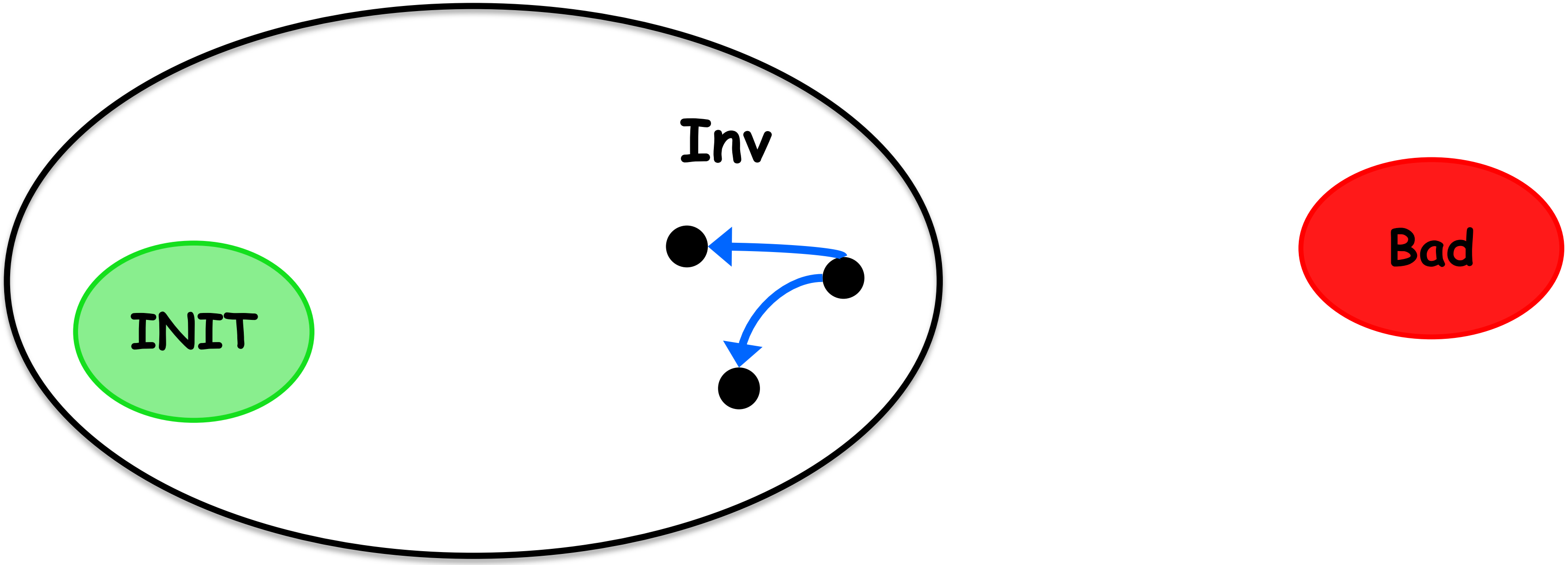
Is Bad reachable?



Yes. There is a counterexample!

Safety Verification Problem

Is Bad reachable?



No. There is an inductive invariant

Programs, Cexs, Invariants

A program *P = (V, Init, $\mathcal{Tr}$, Bad)*

- Notation: $\mathcal{F}(X) = \exists\, \boldsymbol{u} \,.\, (X \wedge \mathcal{Tr}) \vee Init$

P is UNSAFE if and only if there exists a number *N* s.t.

$$Init(X_0) \wedge \left(\bigwedge_{i=0}^{N-1} Tr(X_i, X_{i+1}) \right) \wedge Bad(X_N) \;\not\Rightarrow\; \bot$$

P is SAFE if and only if there exists a *safe inductive invariant Inv s.t.*

$$\left.\begin{array}{r} Init \Rightarrow Inv \\ Inv(X) \wedge Tr(X, X') \Rightarrow Inv(X') \end{array}\right\} \quad \boxed{\text{Inductive}}$$

$$Inv \Rightarrow \neg Bad \qquad \boxed{\text{Safe}}$$

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

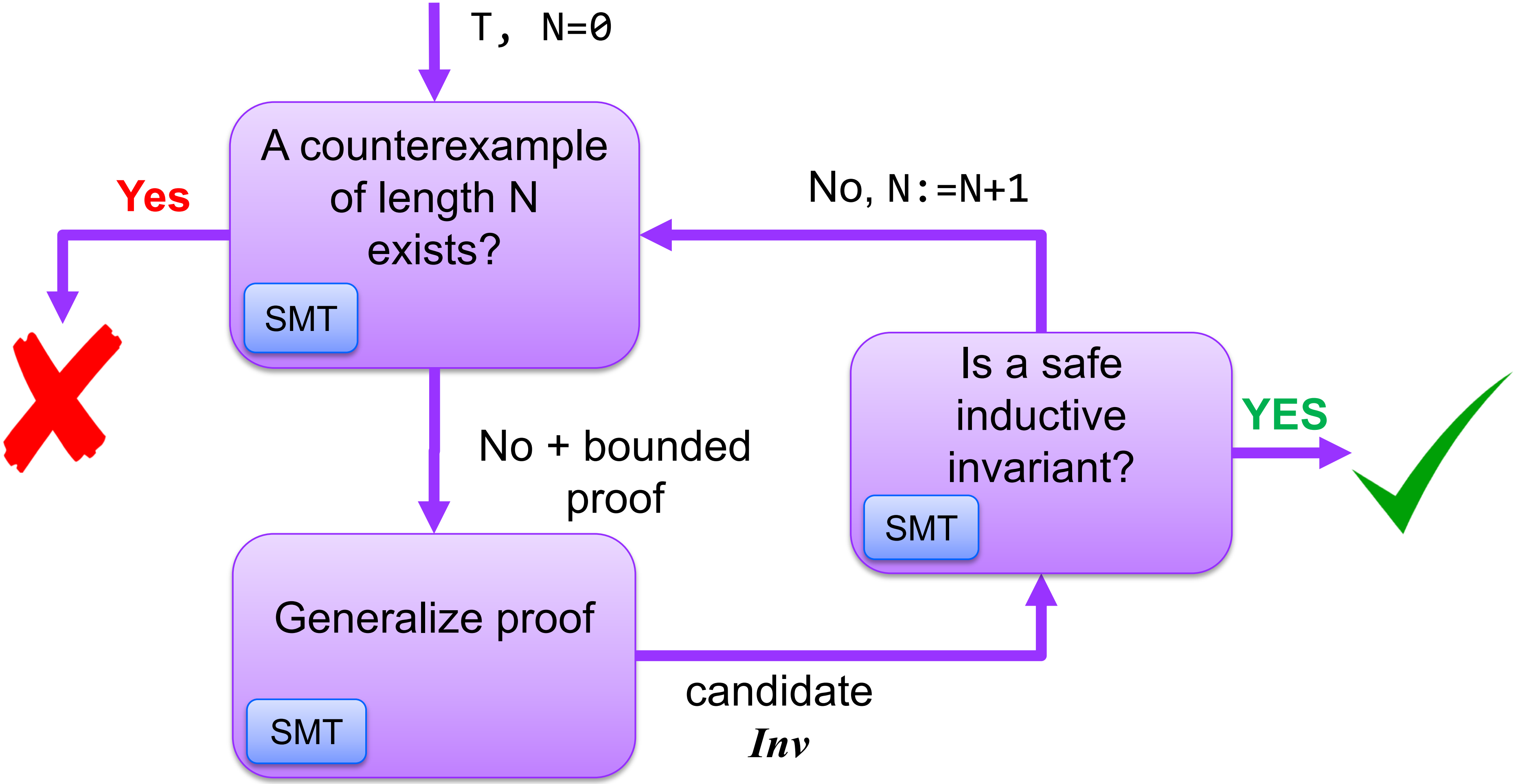
- simulating Numeric Abstract Interpretation with PDR
- N. Bjørner and A. Gurfinkel: Property Directed Polyhedral Abstraction. VMCAI 2015

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

SMT-based Model Checking

Generalizing from bounded proofs



IC3/PDR/Spacer Overview

Input: Safety problem $\langle Init(X), Tr(X, X'), Bad(X)\rangle$

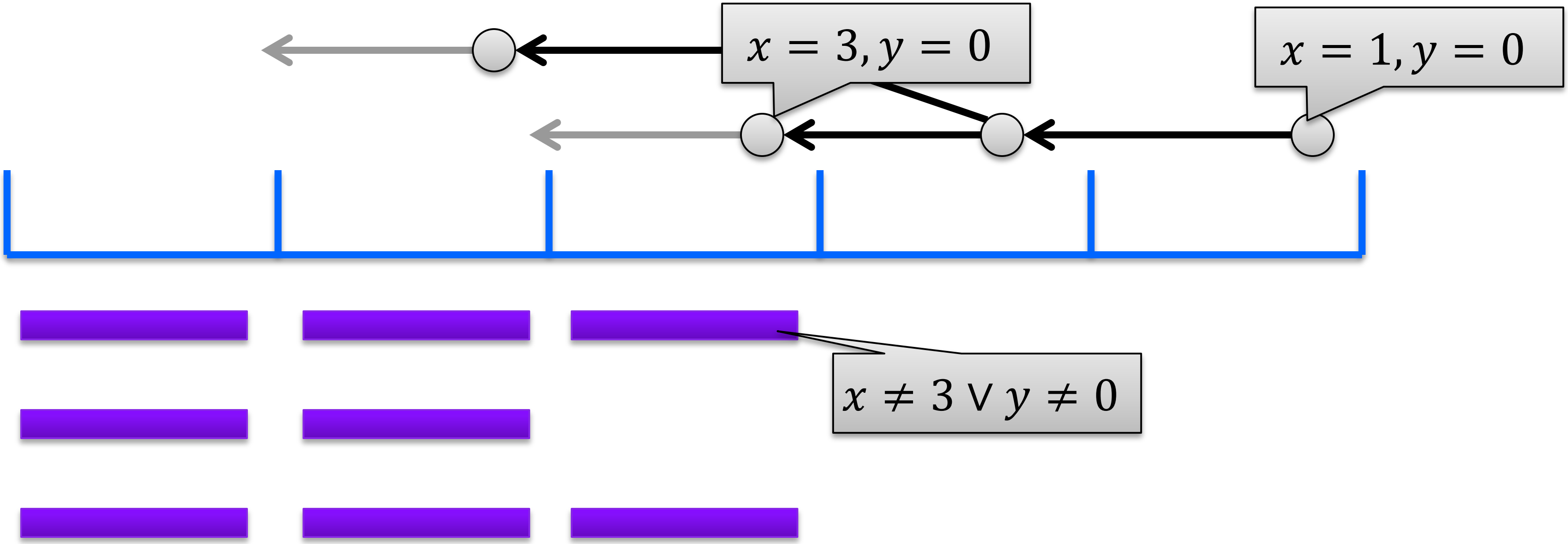
```
$F_0 \leftarrow Init$ ; $N \leftarrow 0$ **repeat**
    $\mathbf{G} \leftarrow \textsc{PdrMkSafe}([F_0, \ldots, F_N], Bad)$
    **if** $\mathbf{G} = [\,]$ **then return** $Reachable$;
    $\forall 0 \leq i \leq N \cdot F_i \leftarrow \mathbf{G}[i]$
    $F_0, \ldots, F_N \leftarrow \textsc{PdrPush}([F_0, \ldots, F_N])$
    **if** $\exists 0 \leq i < N \cdot F_i = F_{i+1}$ **then return** $Unreachable$;
    $N \leftarrow N + 1$ ; $F_N \leftarrow \emptyset$
**until** $\infty$;
```

bounded safety

strengthen result

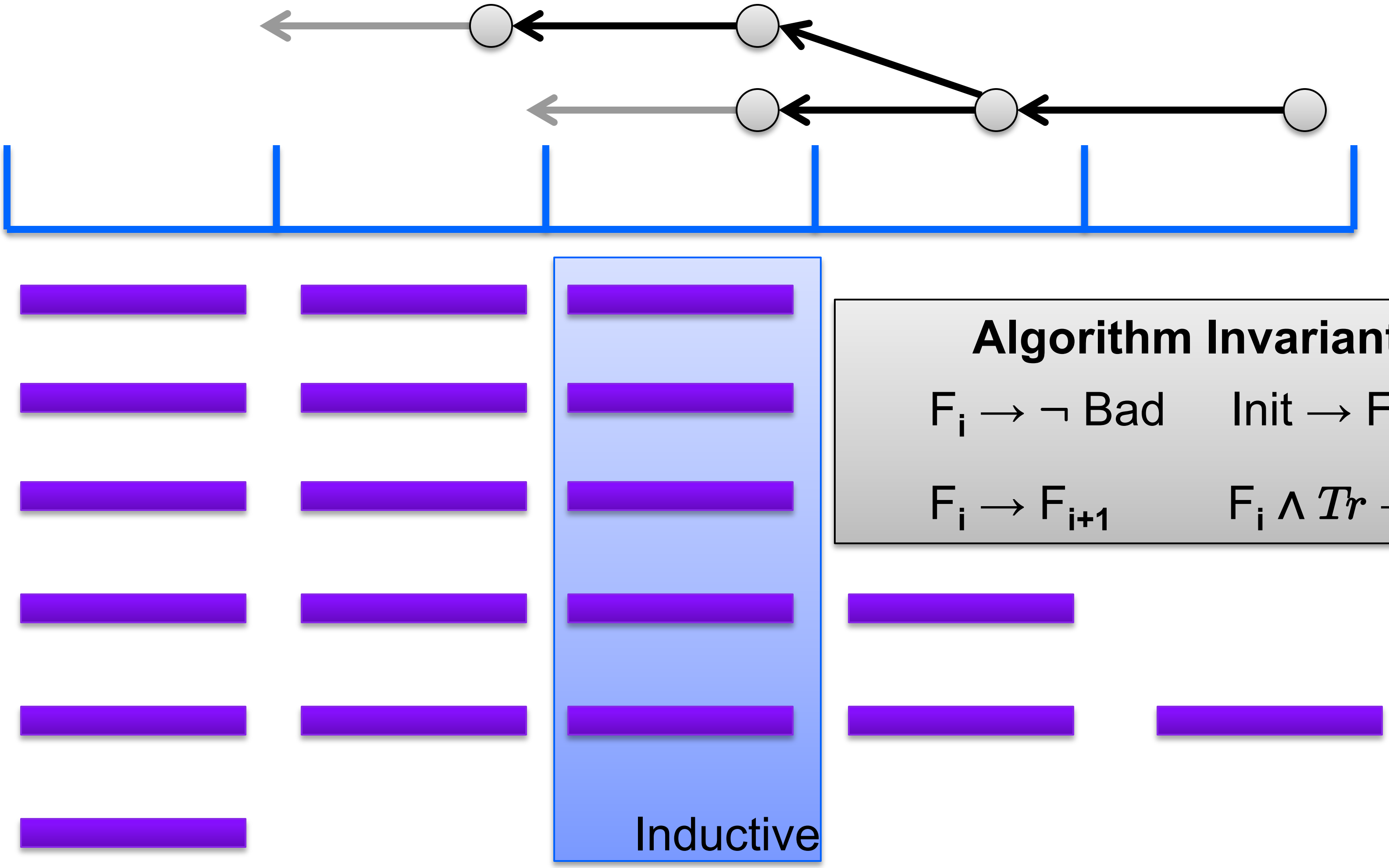
IC3/PDR/Spacer In Pictures: MkSafe

MkSafe



IC3/PDR in Pictures: Push

Push



Algorithm Invariants

$F_i \rightarrow \neg \text{Bad}$ $\quad$ $\text{Init} \rightarrow F_i$

$F_i \rightarrow F_{i+1}$ $\quad$ $F_i \wedge \mathcal{Tr} \rightarrow F_{i+1}$

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

Decide Rule: Generalizing Predecessors

Decide If $\langle m, i+1\rangle \in Q$ and there are m_0 and m_1 s.t. $m_1 \rightarrow m$, $m_0 \wedge m_1'$ is satisfiable, and $m_0 \wedge m_1' \rightarrow F_i \wedge Tr \wedge m'$, then add $\langle m_0, i\rangle$ to Q.

Decide rule chooses a (generalized) predecessor m_0 of *m* that is consistent with the current frame

Simplest implementation is to extract a predecessor m_o from a satisfying assignment of $M \vDash F_i \wedge Tr \wedge m'$

- m_0 cab be further generalized using ternary simulation by dropping literals and checking that m’ remains forced

An alternative is to let m_0 be an implicant (not necessarily prime) of $F_i \wedge \exists X'.(Tr \wedge m')$

- finding a prime implicant is difficult because of the existential quantification
- we settle for an arbitrary implicant. The side conditions ensure it is not trivial

Conflict Rule: Inductive Generalization

Conflict For $0 \leq i < N$: given a candidate model $\langle m, i+1 \rangle \in Q$ and clause φ, such that $\varphi \to \neg m$, if $\mathit{Init} \to \varphi$, and $\varphi \wedge F_i \wedge \mathit{Tr} \to \varphi'$, then add φ to F_j, for $j \leq i+1$.

A clause φ is inductive relative to F iff

- Init → φ (Initialization) and φ ∧ F ∧ Tr → φ (Inductiveness)

Implemented by first letting φ = ¬m and generalizing φ by iteratively dropping literals while checking the inductiveness condition

Theorem: Let F_0, F_1, …, F_N be a valid IC3 trace. If φ is inductive relative to F_i, 0 · i < N, then, for all j · i, φ is inductive relative to F_j.

- Follows from the monotonicity of the trace
  - if j < i then $F_j \to F_i$
  - if $F_j \to F_i$ then $(\varphi \wedge F_i \wedge Tr \to \varphi) \to (\varphi \wedge F_j \wedge Tr \to \varphi')$

From Propositional PDR to Solving CHC

Theories with Infinite Models

- infinitely many satisfying assignments
- can’t simply enumerate (in decide)
- can’t block one assignment at a time (in conflict)

Non-Linear Horn Clauses

- multiple predecessors (in decide)

The problem is undecidable in general, but we want an algorithm that makes progress

- doesn’t get stuck in a decidable sub-problem

CHC OVER LINEAR ARITHMETIC

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

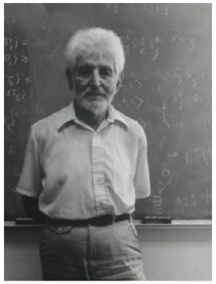
$$((F_i \wedge Tr) \vee Init') \Rightarrow \varphi'$$

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

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

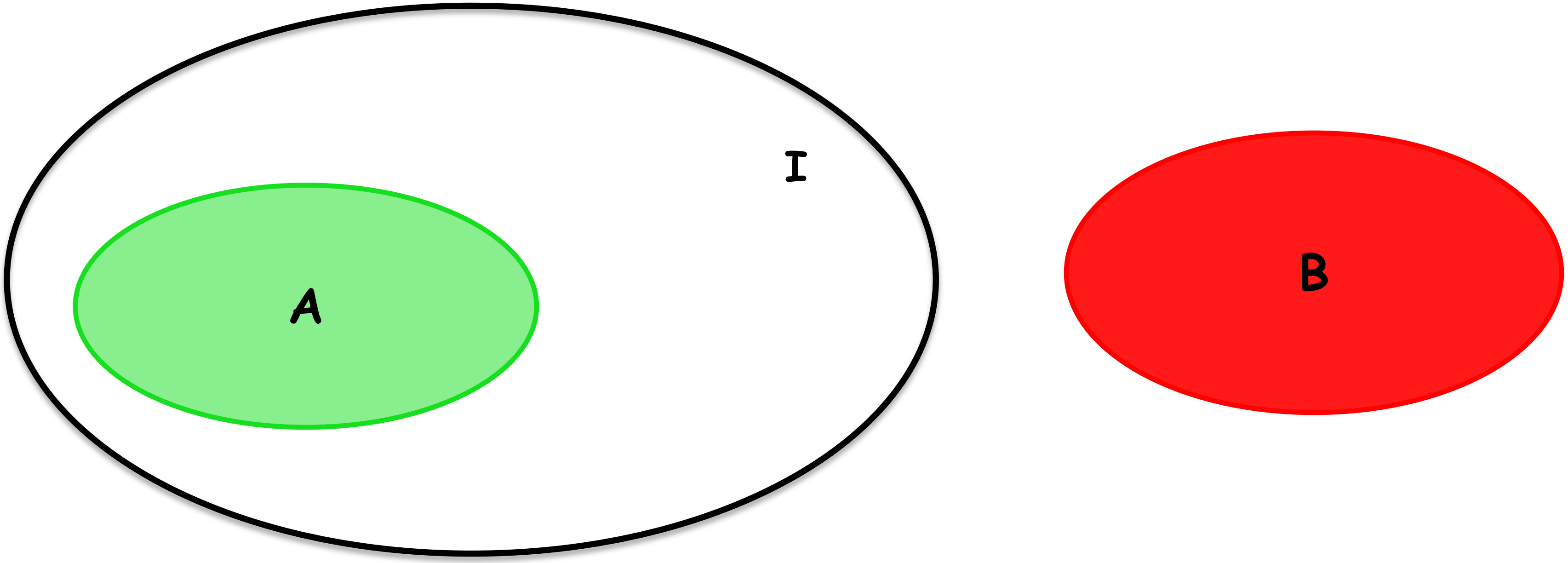
$$A \Rightarrow I \qquad I \Rightarrow \neg B$$

$$\mathit{atoms}(I) \in \mathit{atoms}(A) \cap \mathit{atoms}(B)$$

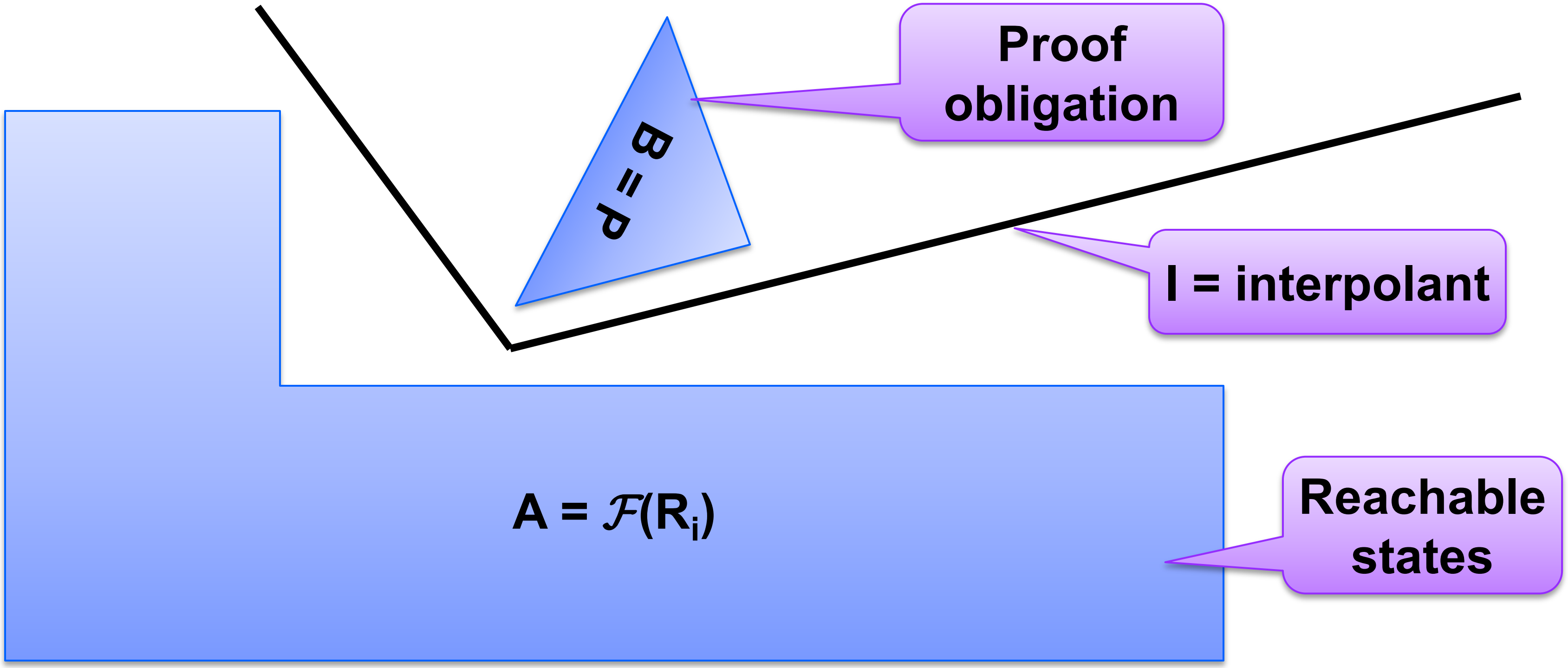
A Craig interpolant ITP(A, B) can be effectively constructed from a resolution proof of unsatisfiability of A∧B

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

Craig Interpolant



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 ∧ 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

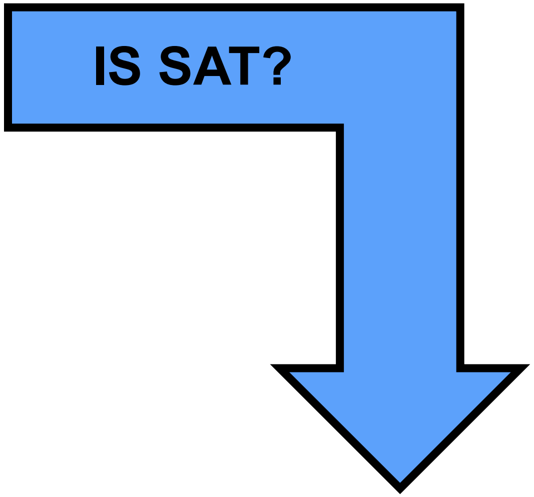
- every theory-lemma that mixes B-pure literals with other literals is interpolated to produce a single literal in the final solution
- interpolation is restricted to clauses of the form $(\wedge B_i \Rightarrow \vee A_j)$

Interpolating (UNSAT) Cores *(ongoing work with Bernhard Gleiss)*

- 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

Back to addition example…

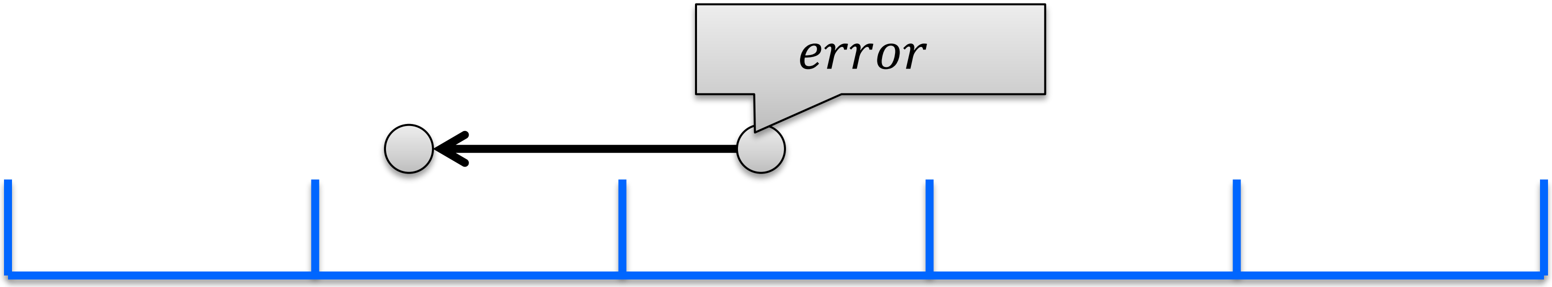
```
z = x; i = 0;
assume (y > 0);
while (i < y)
  z = z + 1;
assert(z == x + y);
```



```
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
```

MkSafe

Lemma Generation Example



Transition Relation

Pob

x = x0 & z = z0+1 & i=i0+1 & y > i0

i >= y & x + y > z

Farkas explanation for unsat

$$\dfrac{\dfrac{x0 + y0 \le z0,\ \ x \le x0,\ z0 < z,\ i \le i0 + 1}{x + i \le z} \qquad \dfrac{i \ge y,\ x+y > z}{x + i > z}}{\textbf{false}}$$

Learn lemma: x + i <= z

$$s \subseteq pre(c)$$

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

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

ARITHMETIC DECIDE

Model Based Projection

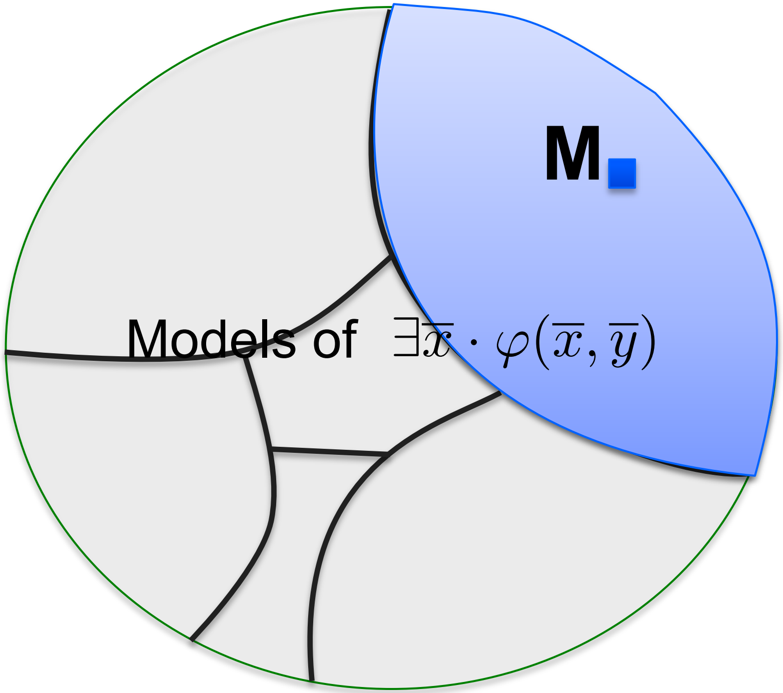
Definition: Let φ be a formula, U a set of variables, and M a model of φ. Then ψ = MBP (U, M, φ) is a Model Based Projection of U, M and φ iff

1. ψ is a monomial
2. Vars(ψ) $\subseteq$ Vars(φ) \ U
3. M ⊨ ψ
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 $\psi(\overline{y}) \equiv \exists \overline{x} \cdot \varphi(\overline{x}, \overline{y})$


M
Models of $\exists \overline{x} \cdot \varphi(\overline{x}, \overline{y})$

1. Find model M of φ (x,y)

2. Compute a partition containing M

Quantifier Elimination for Linear Real Arithmetic

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

Quadratic increase in the formula size

Quantifier Elimination with an order side-cond

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

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 order on terms?!

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

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

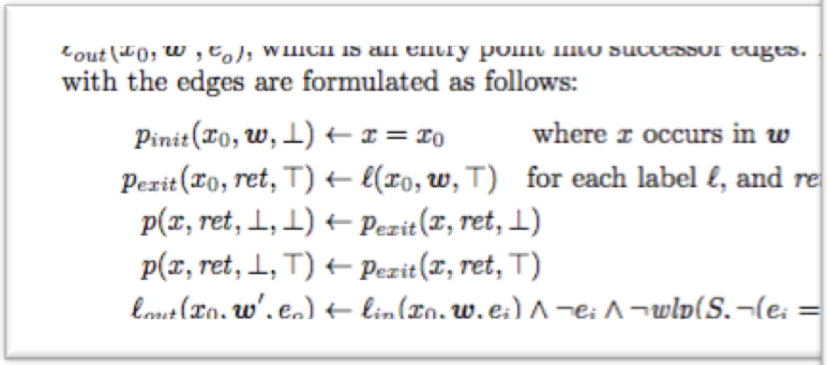
- finitely many possible predecessors when all other arguments are fixed

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)

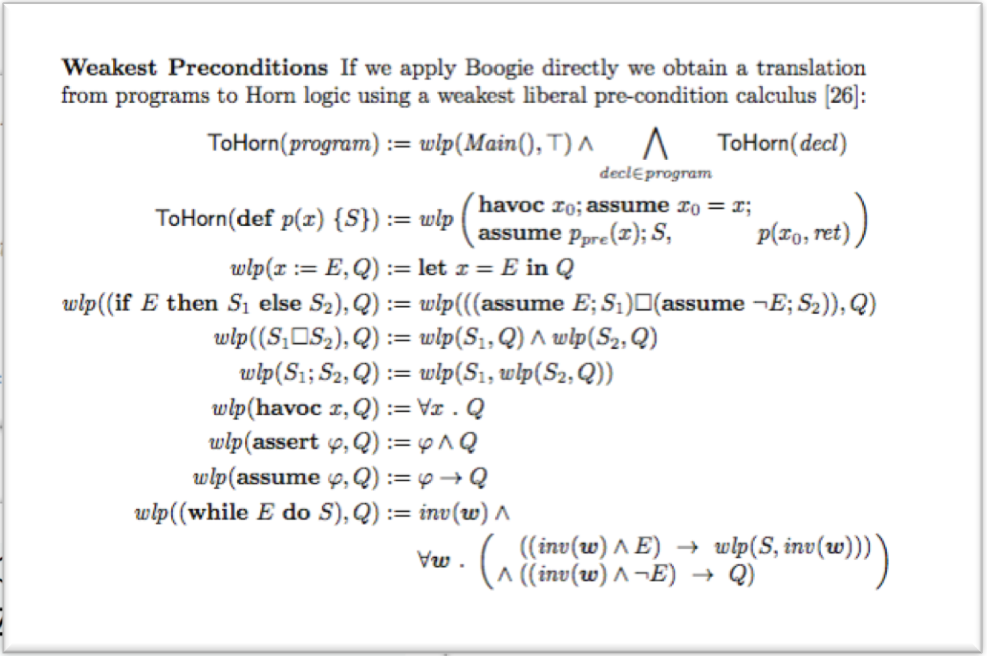
SOLVING NON-LINEAR CHC

Horn Clauses for Program Verification



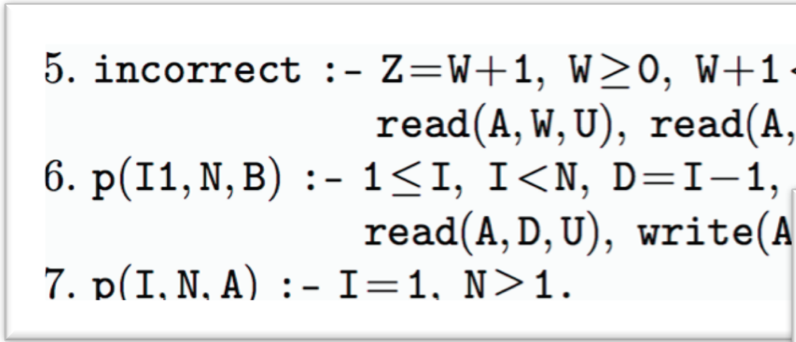
$\ell_{out}(x_0, \boldsymbol{w}', e_o)$, which is an entry point into successor edges.
with the edges are formulated as follows:

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



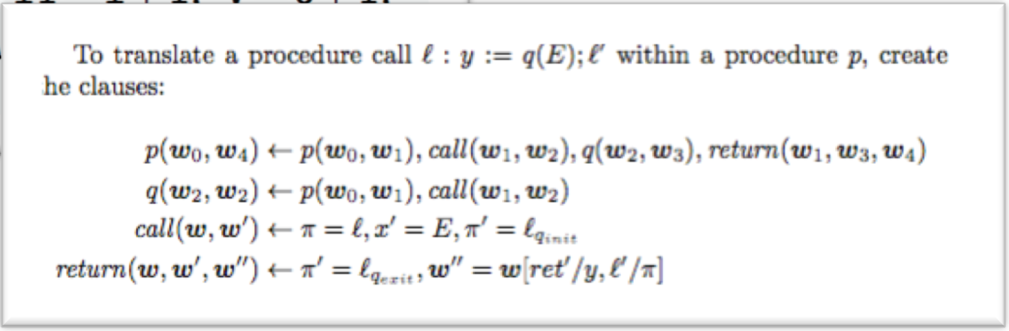
Weakest Preconditions If we apply Boogie directly we obtain a translation from programs to Horn logic using a weakest liberal pre-condition calculus [26]:

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



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

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



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

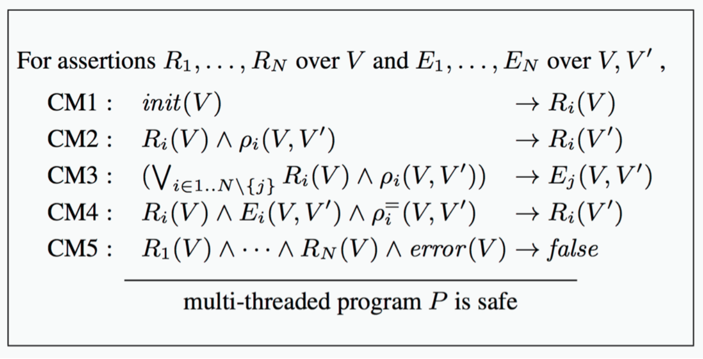
$$p(\boldsymbol{w}_0, \boldsymbol{w}_4) \leftarrow p(\boldsymbol{w}_0, \boldsymbol{w}_1), call(\boldsymbol{w}_1, \boldsymbol{w}_2), q(\boldsymbol{w}_2, \boldsymbol{w}_3), return(\boldsymbol{w}_1, \boldsymbol{w}_3, \boldsymbol{w}_4)$$
$$q(\boldsymbol{w}_2, \boldsymbol{w}_2) \leftarrow p(\boldsymbol{w}_0,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

Bjørner, Gurfinkel, McMillan, and Rybalchenko:

Horn Clause Solvers for Program Verification

UNIVERSITY OF WATERLOO

Horn Clauses for Concurrent / Distributed / Parameterized Systems

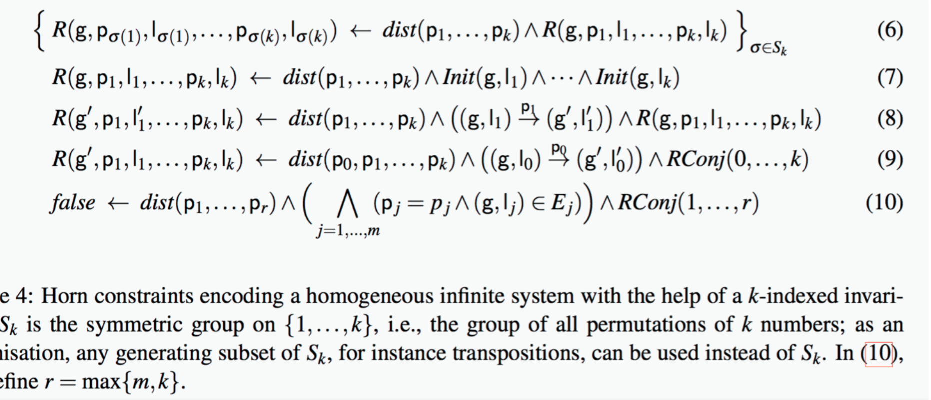


For assertions $R_1, \ldots, R_N$ over V and $E_1, \ldots, E_N$ over V, V',

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

multi-threaded program P is safe

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



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

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

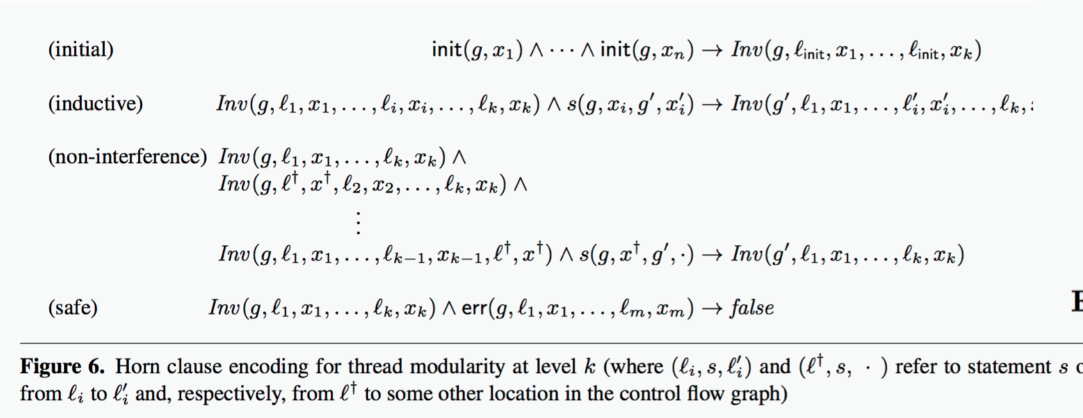
$$R(\mathsf{g}', \mathsf{p}_1, \mathsf{l}'_1, \ldots, \mathsf{p}_k, \mathsf{l}_k) \leftarrow dist(\mathsf{p}_1, \ldots, \mathsf{p}_k) \wedge \big((\mathsf{g}, \mathsf{l}_1) \xrightarrow{\mathsf{p}_1} (\mathsf{g}', \mathsf{l}'_1)\big) \wedge R(\mathsf{g}, \mathsf{p}_1, \mathsf{l}_1, \ldots, \mathsf{p}_k,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

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

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

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



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

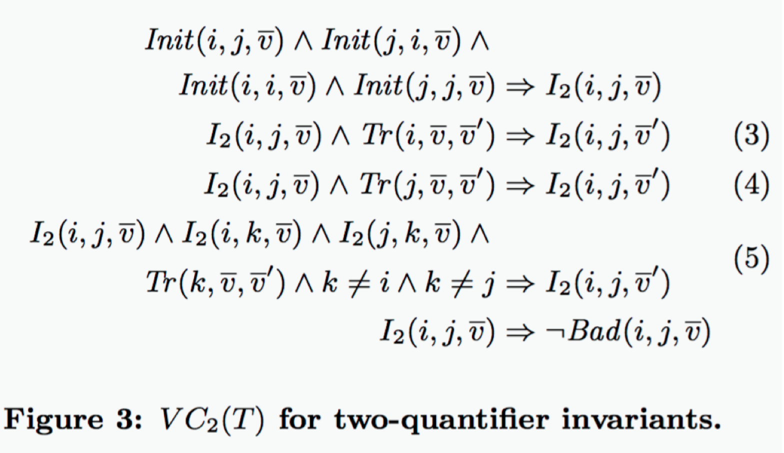
(inductive) $Inv(g, \ell_1, x_1, \ldots, \ell_i, x_i, \ldots, \ell_k, x_k) \wedge s(g, x_i, g', x'_i) \rightarrow Inv(g', \ell_1, x_1, \ldots, \ell'_i, x'_i, \ldots, \ell_k,$

(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 ar
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



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

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

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

Non-Linear CHC Satisfiability

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

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

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

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

where, X' = {x' | x $\in$ X}, X^o = {x^o | 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 $\varphi$ to $F_j$, for all $j \leq i + 1$.

until $\infty$;
```

counterexample is a tree

two predecessors

theory-aware **Conflict**

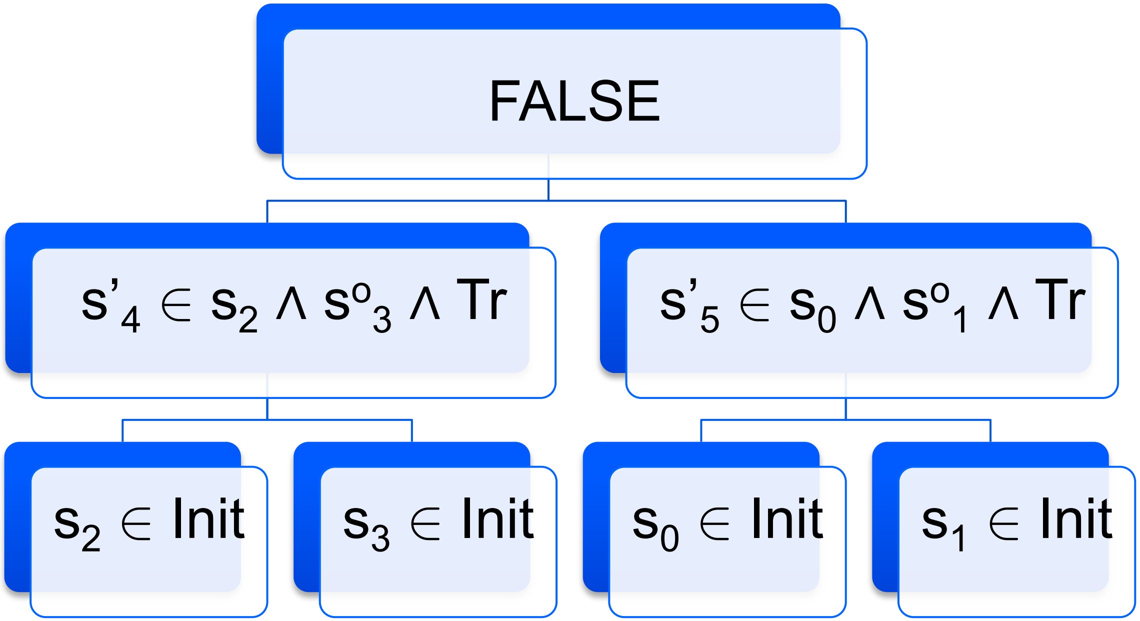
Counterexamples to non-linear CHC

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



FALSE

$s'_4 \in s_2 \wedge s^{o}_3 \wedge Tr$

$s'_5 \in s_0 \wedge s^{o}_1 \wedge Tr$

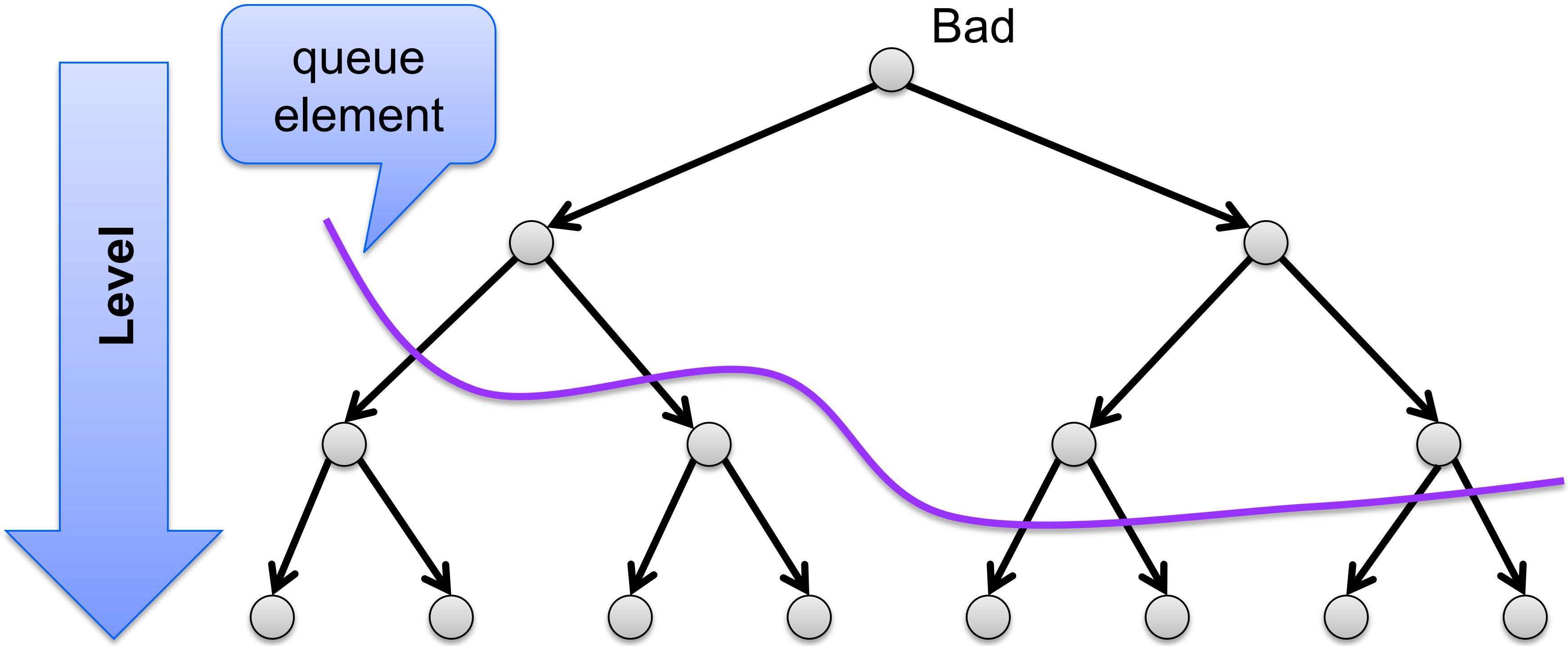
$s_2 \in Init$

$s_3 \in Init$

$s_0 \in Init$

$s_1 \in Init$

GPDR Search Space



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

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

Can explore both predecessors in parallel

- e.g., BFS or DFS exploration order

Number of predecessors is unbounded

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

No caching/summarization of previous decisions

- worst-case exponential for Boolean Push-Down Systems

Spacer

Same queue as in IC3/PDR

Cache Reachable states

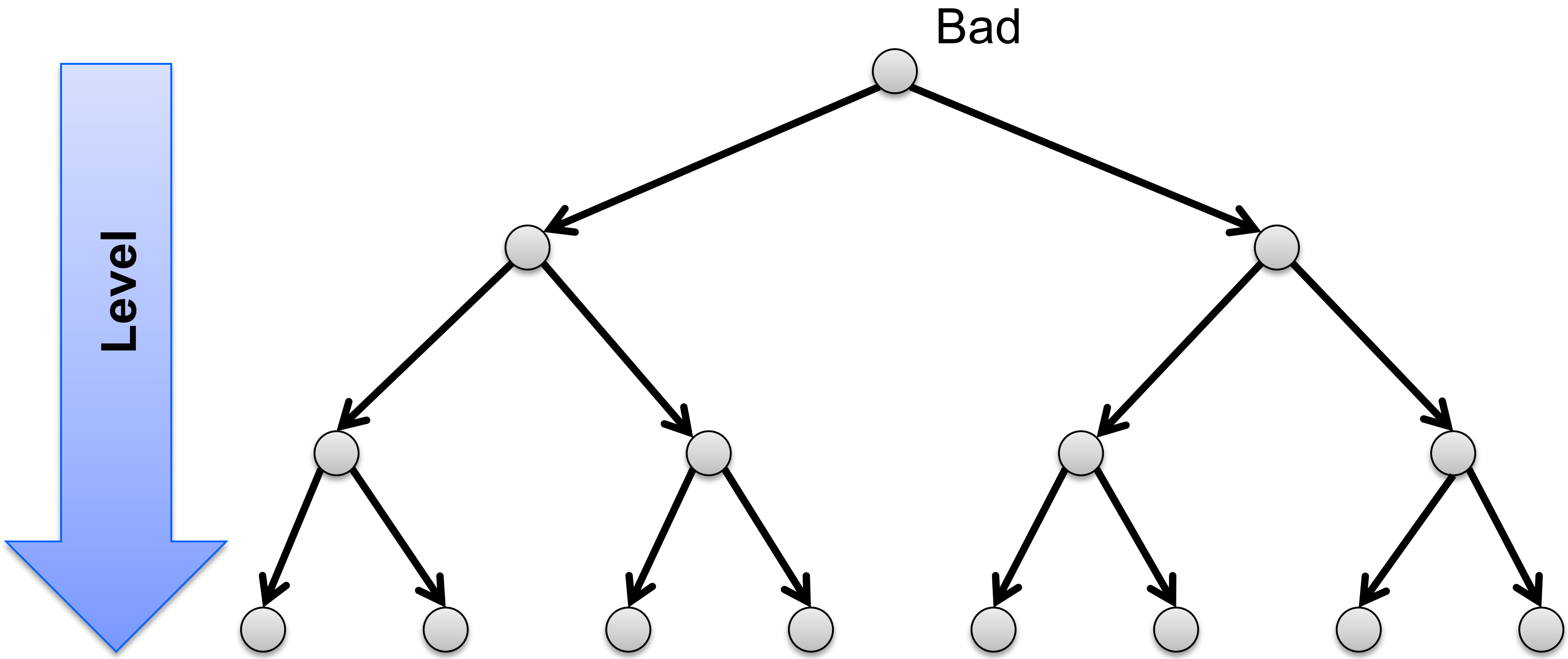
Three variants of **Decide**

Same **Conflict** as in APDR/GPDR

```
Input: A safety problem $\langle Init(X), Tr(X, X^o, X'), Bad(X)\rangle$.
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: $Init \rightarrow \neg Bad$
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 $F_N \rightarrow \neg Bad$, 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
        $\psi = \mathcal{F}(\vee\textsc{Reach}) \wedge m'$. Then, add $s$ to $\textsc{Reach}$, where
        $s' \in \text{MBP}(\{X, X^o\}, \psi)$.

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

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

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

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

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

until $\infty$;
```

UNIVERSITY OF WATERLOO

SPACER Search Space



Unfold the derivation tree in a fixed depth-first order

- use MBP to decide on counterexamples

Learn new facts (reachable states) on the way up

- use MBP to propagate facts bottom up

UNIVERSITY OF WATERLOO

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

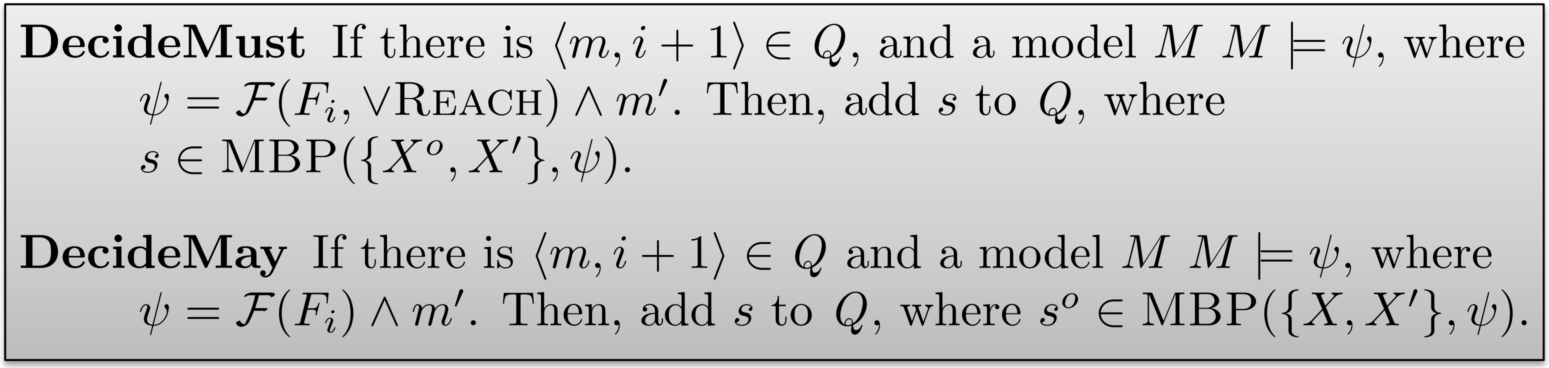
Second use of MBP

- orthogonal to the use of MBP in Decide
- REACH can contain auxiliary variables, but might get too large

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

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

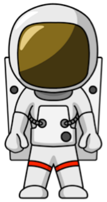
DecideMust

- use computed summary 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**)

Conclusion and Future Work



Spacer: an SMT-based procedure for deciding CHC modulo theories

- extends IC3/PDR from SAT to SMT
- interpolation to over-approximate a possible model
- model-based projection to summarize derivations

The curse of interpolation

- interpolation is fantastic at quickly discovering good lemmas
- BUT it is highly unstable: small changes to input (or code) drastically change what is discovered
- what is easy today might be difficult tomorrow ☹

Many open problems

- Parallel solving (see FMCAD’17)
- Supporting extra theories: bit-vectors, uninterpreted functions, EPR
- Stability – reduce reliance on interpolation
- Exploration strategies, transformations, heuristics, …

CHC-COMP: CHC Solving Competition

First edition on July 13, 2018 at HVCS@FLOC

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

?

?

?

?

?

?

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

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

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
- $B = t_{i+1} \geq b_i \wedge \ldots \wedge t_n \geq b_n$

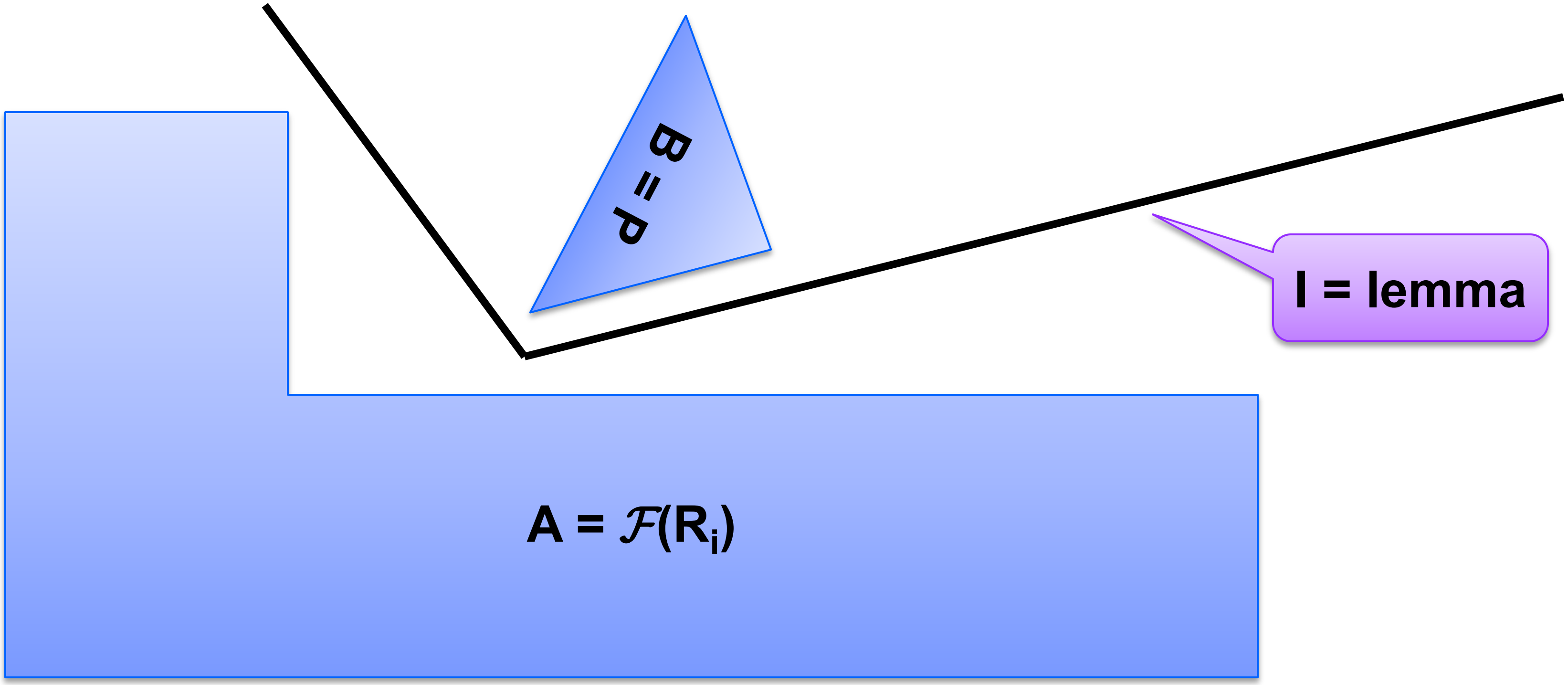
Let $g_1, \ldots, g_n$ be the Farkas coefficients witnessing UNSAT

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

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