The SeaHorn Verification Framework

Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA 15213

Arie Gurfinkel

with Teme Kahsai, Jorge A. Navas, and
Anvesh Komuravelli
April 11th, 2015

Automated Software Analysis

Program

Automate
d
Analysis

Correc
t

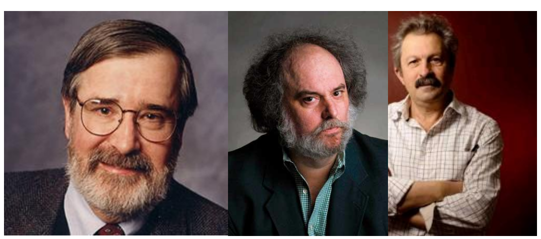
Incorre
ct

Software Model Checking with Predicate Abstraction

e.g., Microsoft's SDV

Abstract Interpretation with Numeric Abstraction

e.g., ASTREE, Polyspace



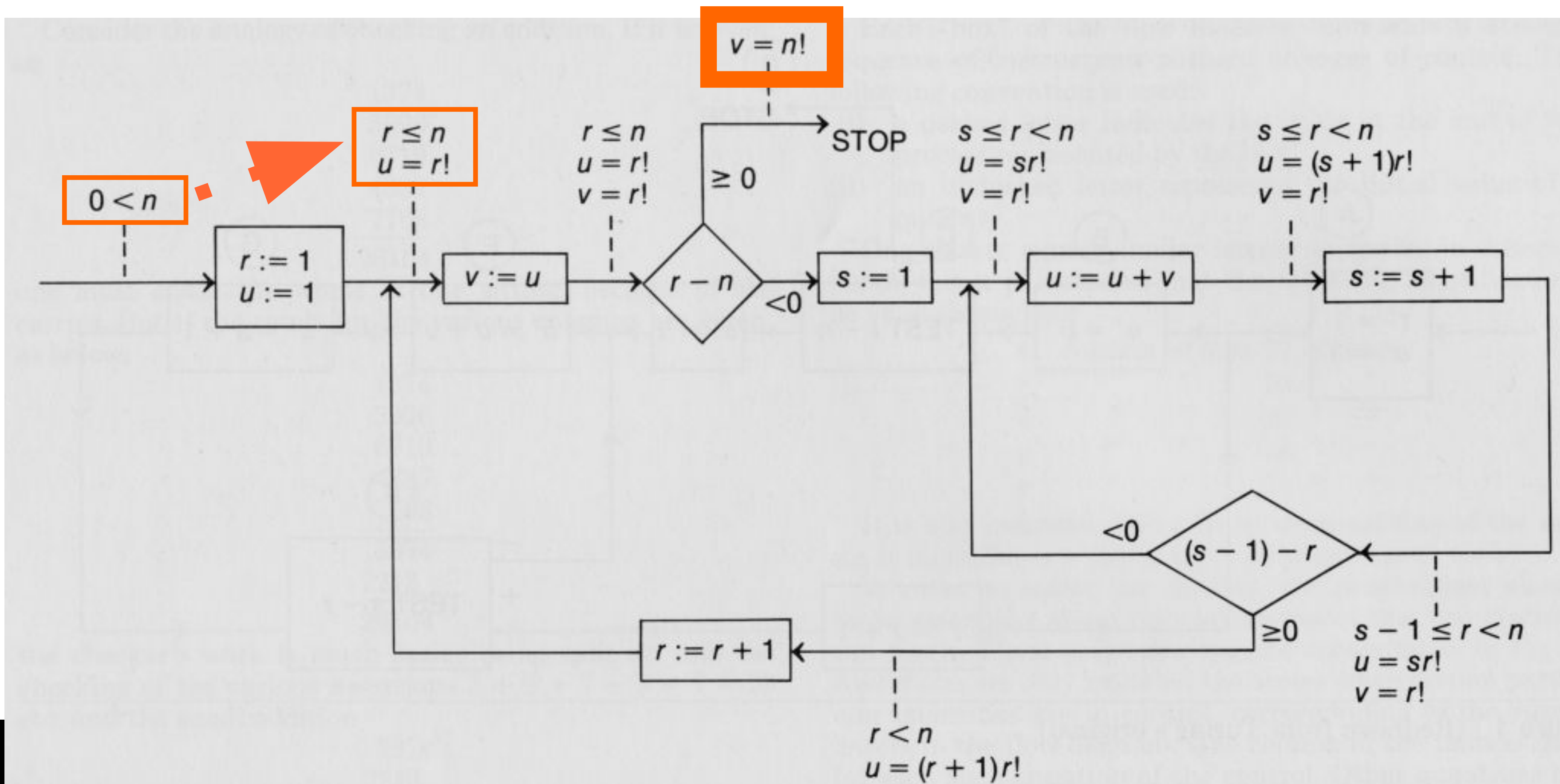
Turing, 1936:
“undecidable”

Turing, 1949

Alan M. Turing. “Checking a large routine”, 1949

How can one check a routine in the sense of making sure that it is right?

programmer should make a number of definite assertions which can be checked individually, and from which the correctness of the whole programme easily follows.

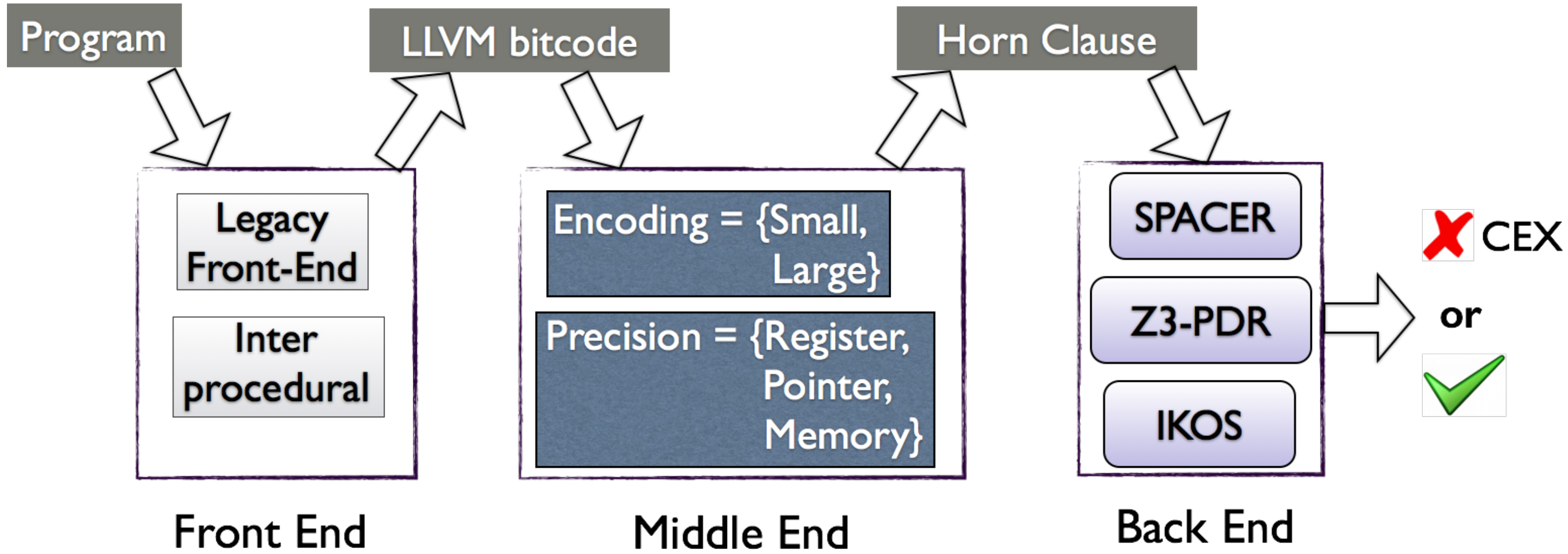


SeaHorn

A fully automated verification framework for LLVM-based languages.

http://seahorn.github.io

SeaHorn Verification Framework



Distinguishing Features

- LLVM front-end(s)
- Constrained Horn Clauses to represent Verification Conditions
- Comparable to state-of-the-art tools at SV-COMP'15

Goals

- be a state-of-the-art Software Model Checker
- be a framework for experimenting and developing CHC-based verification

Related Tools

CPAChecker

- Custom front-end for C
- Abstract Interpretation-inspired verification engine
- Predicate abstraction, invariant generation, BMC, k-induction

SMACK / Corral

- LLVM-based front-end
- Reduces C verification to Boogie
- Corral / Q verification back-end based on Bounded Model Checking with SMT

SeaHorn Usage

`> sea pf FILE.c`

Outputs `sat` for unsafe (has counterexample); `unsat` for safe

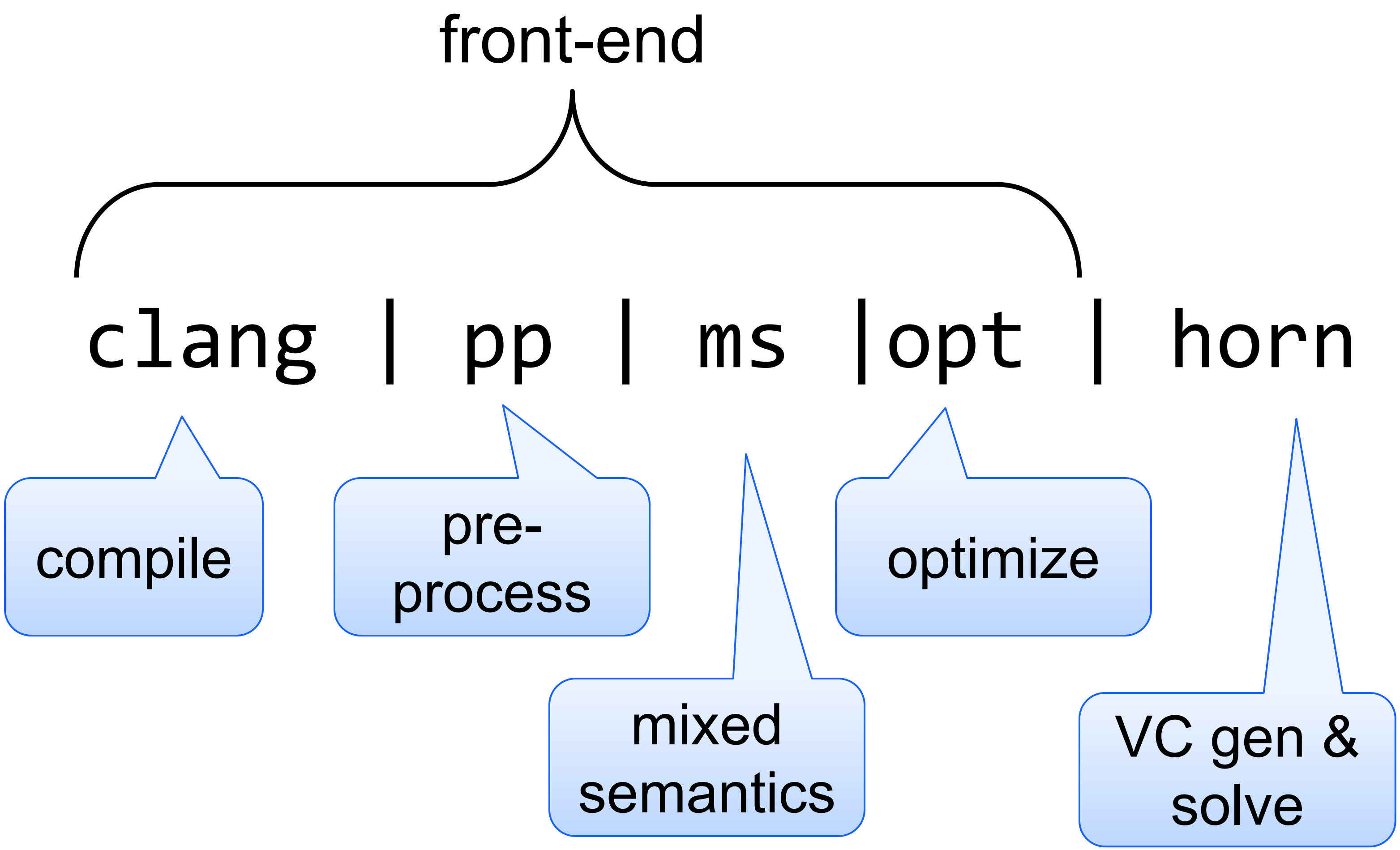
Additional options

- `--cex=trace.xml` outputs a counter-example in SV-COMP'15 format
- `--track={reg,ptr,mem}` track registers, pointers, memory content
- `--step={large,small}` verification condition step-semantics
  - *small* == basic block, *large* == loop-free control flow block
- `--inline` inline all functions in the front-end passes

Additional commands

- `sea smt` – generates CHC in extension of SMT-LIB2 format
- `sea clp` -- generates CHC in CLP format (under development)
- `sea lfe-smt` – generates CHC in SMT-LIB2 format using legacy front-end

Verification Pipeline



Constrained Horn Clauses (CHC)

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

8 V . (Á Æ $p_1[X_1]$ Æ…Æ $p_n[X_n]$ → h[X]), where

- Á is a constrained in a background theory A (e.g., arithmetic, arrays, SMT)
- p_1, …, p_n, h are n-ary predicates
- $p_i[X]$ is an application of a predicate to first-order terms

We write clauses as rules, with all variables implicitly quantified

h[X] Ã $p_1[X_1]$,…, $p_n[X_n]$, Á.

A model of a set of clauses ¦ is an interpretation of each predicate p_i that makes all clauses in ¦ valid

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

A model is A-definable, it each p_i is definable by a formula $Ã_i$ in A

FROM PROGRAMS TO CLAUSES

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 Æ Q

wlp (**assume**(E), Q) = E → Q

wlp (**while** E **do** S, Q) = I(w) Æ

8w . ((I(w) Æ E) → wlp (S, I(w))) Æ ((I(w) Æ :E) → Q))

wlp (y = P(E), Q) = p_{pre}(E) Æ (8 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) Æ 8{P 2 Prog} . ToHorn (P)

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, $p_{en}(x_0,x,w)$ for entry, p_{ex} (x_0,r) for exit

The verification condition is a conjunction of clauses:

$p_{en}(x_0,x)$ Ã $x_0=x$

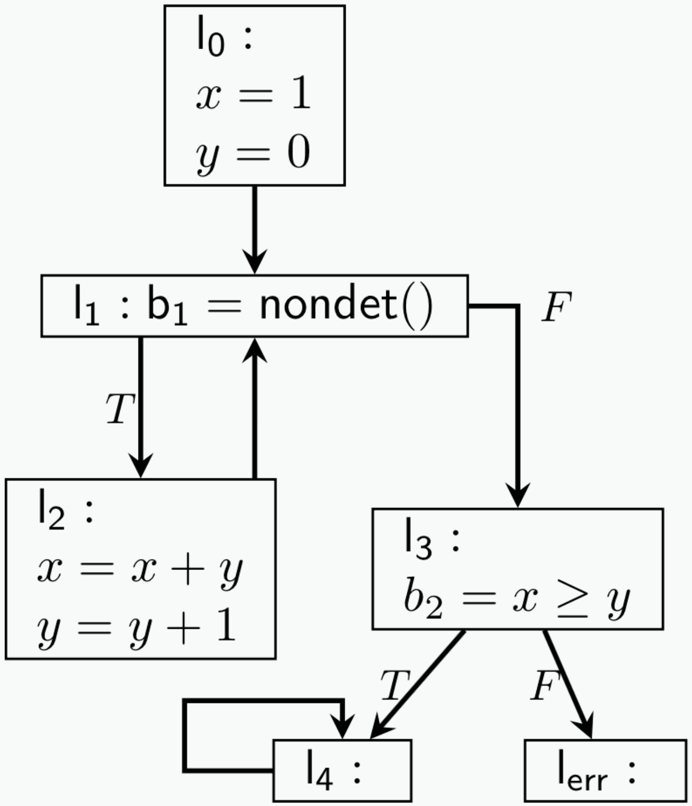
$l_i(x_0,w')$ Ã $l_j(x_0,w)$ Æ :wlp (S, :(w=w')), for each statement l_i: S; goto l_j

p (x_0,r) Ã $p_{ex}(x_0,r)$

false Ã $Main_{ex}$(x, ret)

Example Horn Encoding

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



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

Large Step Encoding: Single Static Assignment

```
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:
```

Example: Large Step Encoding

```
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
```

Example: Large Step Encoding

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

```
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: Large Step Encoding

$x_1 = x_0 + y_0$

$x_2 = x_0 - y_0$

$y_1 = -1 * y_0$

$B_2 \rightarrow x_0 < N$

$B_3 \rightarrow B_2 \wedge y_0 > 0$

$B_4 \rightarrow B_2 \wedge y_0 \leq 0$

$B_5 \rightarrow (B_3 \wedge x_3=x_1) \vee (B_4 \wedge x_3=x_2)$

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

$p_1(x'_0,y'_0)$ Ã $p_1(x_0, y_0)$, Á.

```
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
```

Mixed Semantics

PROGRAM TRANSFORMATION

[GWC’08,LQ’14]

Mixed Semantics

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]
  - (¾, ¾`) 2 ||f|| iff the execution of f on input state ¾ terminates and results in state ¾’
- 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,

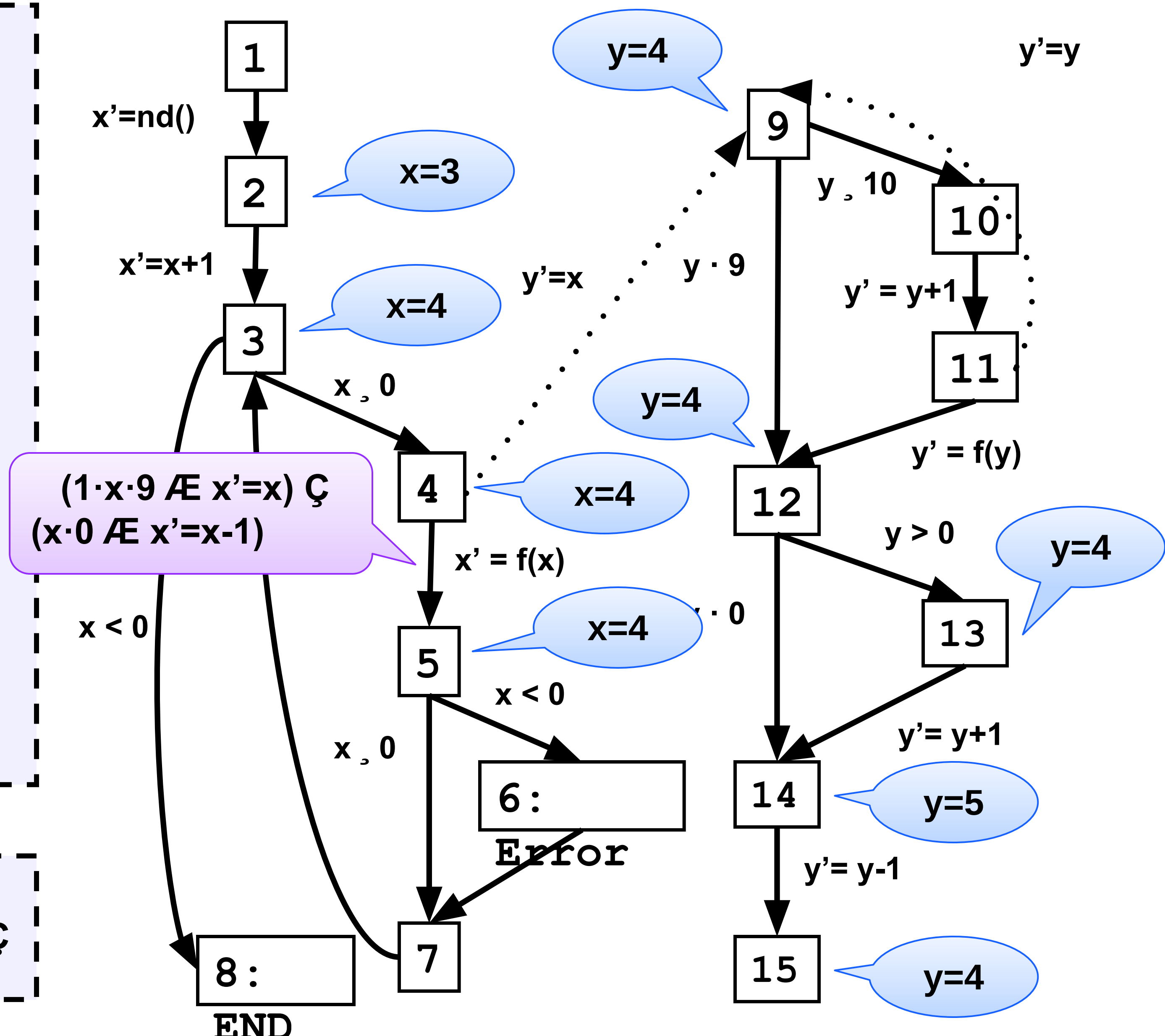
K ² EF (pc=L) , K^m ² EF (pc=L) and K ² EG (pc≠L) , K^m ² EG (pc≠L)

```
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
```

Summary of f(y)

(1·y·9 Æ y'=y) Ç
(y·0 Æ y'=y-1)



Mixed Semantics as Program Transformation

```
main ()
    p1 (); p1 ();
    assert (c1);
p1 ()
    p2 ();
    assert (c2);
p2 ()
    assert (c3);
```

Mixed Semantics

```
main_new ()
    if (*) goto p1_entry;
    else p1_new ();
    if (*) goto p1_entry;
    else p1_new ();
    if (¬c1) goto error;
    assume (false);
p1_entry :
    if (*) goto p2_entry;
    else p2_new ();
    if (¬c2) goto error;
p2_entry :
    if (¬c3) goto error;
    assume (false);
error : assert (false);
p1_new ()
    p2_new ();
    assume (c2);
p2_new ()
    assume (c3);
```

SOLVING CHC WITH SMT

Programs, Cexs, Invariants

A program *P = (V, Init, ½, Bad)*

- Notation: F(X) = 9 ***u*** . (X Æ ½) Ç Init

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

$$Init(v_0) \wedge \left(\bigwedge_{i=0}^{N-1} \rho(v_i, v_{i+1}) \right) \wedge Bad(v_N) \not\Rightarrow \perp$$

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

$$\left.\begin{aligned} Init(u) &\Rightarrow Inv(u) \\ Inv(u) \wedge \rho(u, v) &\Rightarrow Inv(v) \end{aligned}\right\} \quad \text{Inductive}$$

$$Inv(u) \Rightarrow \neg Bad(u) \quad \text{Safe}$$

IC3/PDR Algorithm Overview

Input: Transition system $T = (\mathit{Init}, \mathit{Tr}, \mathit{Bad})$

$F_0 \leftarrow \mathit{Init}\,;\, N \leftarrow 0$

repeat

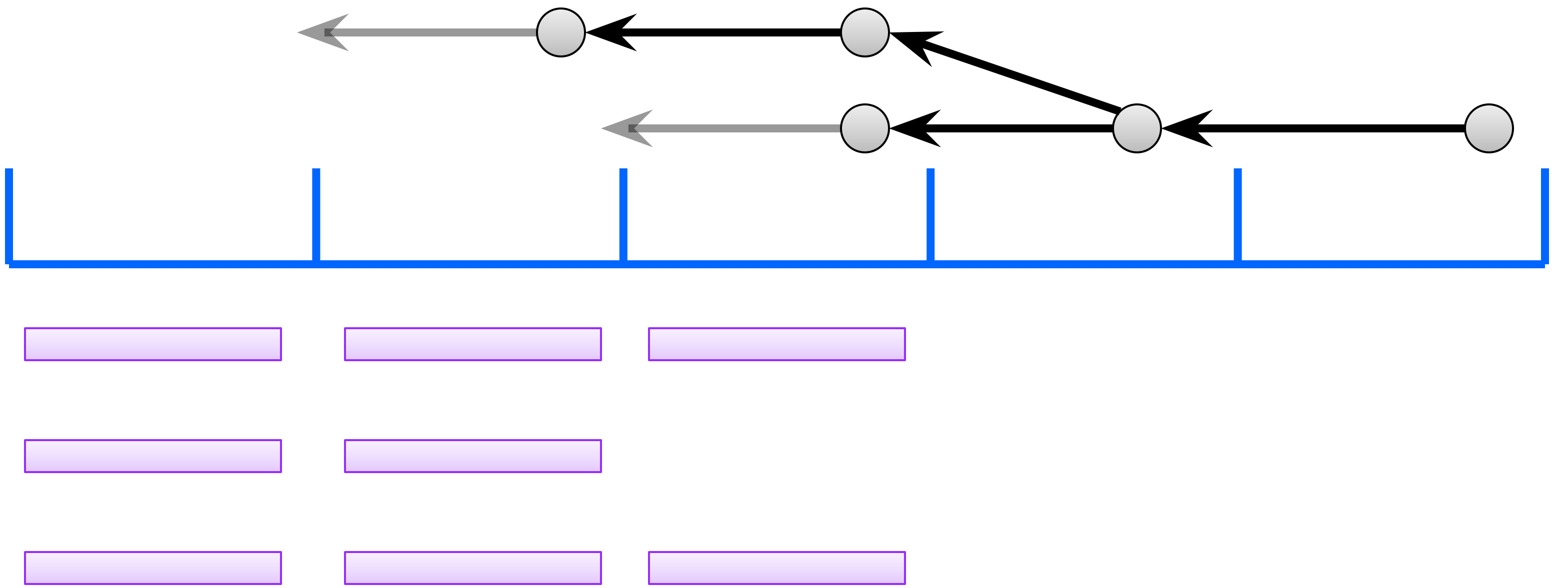
- $\boldsymbol{G} \leftarrow \textsc{PdrMkSafe}([F_0, \ldots, F_N], \mathit{Bad})$
- **if** $\boldsymbol{G} = [\,]$ **then return** UNSAFE;
- $\forall 0 \leq i \leq N \cdot F_i \leftarrow \boldsymbol{G}[i]$
- $F_0, \ldots, F_N \leftarrow \textsc{PdrPush}([F_0, \ldots, F_N])$
- `//` $F_0, \ldots, F_N$ `is a safe` δ`-trace`
- **if** $\exists 0 \leq i \leq N \cdot F_i = \emptyset$ **then return** SAFE;
- $N \leftarrow N + 1\,;\, F_N \leftarrow \emptyset$

until ∞;

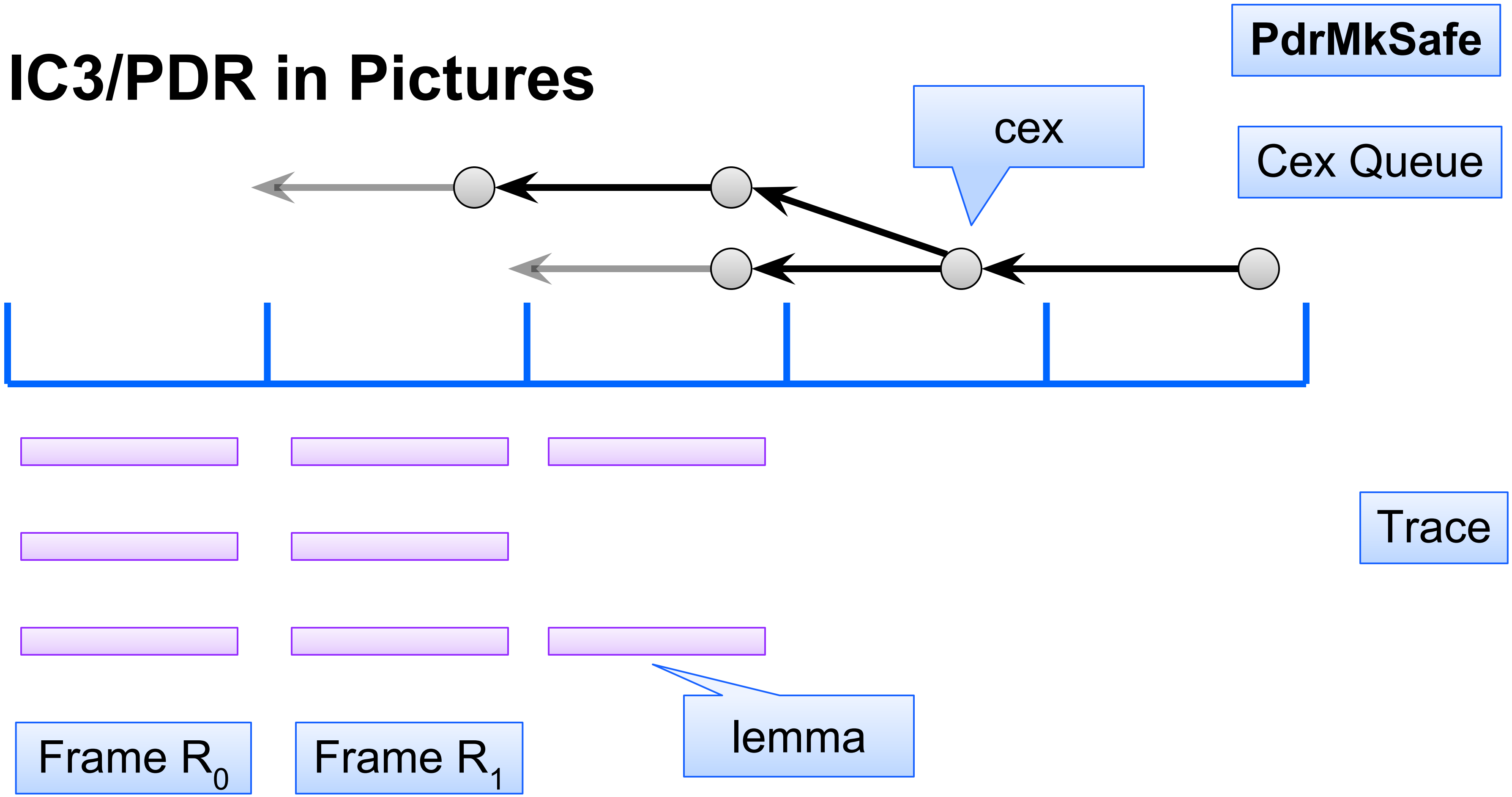
Aaron R. Bradley:SAT-Based Model Checking without Unrolling. VMCAI 2011: 70-87

IC3/PDR in Pictures

PdrMkSafe

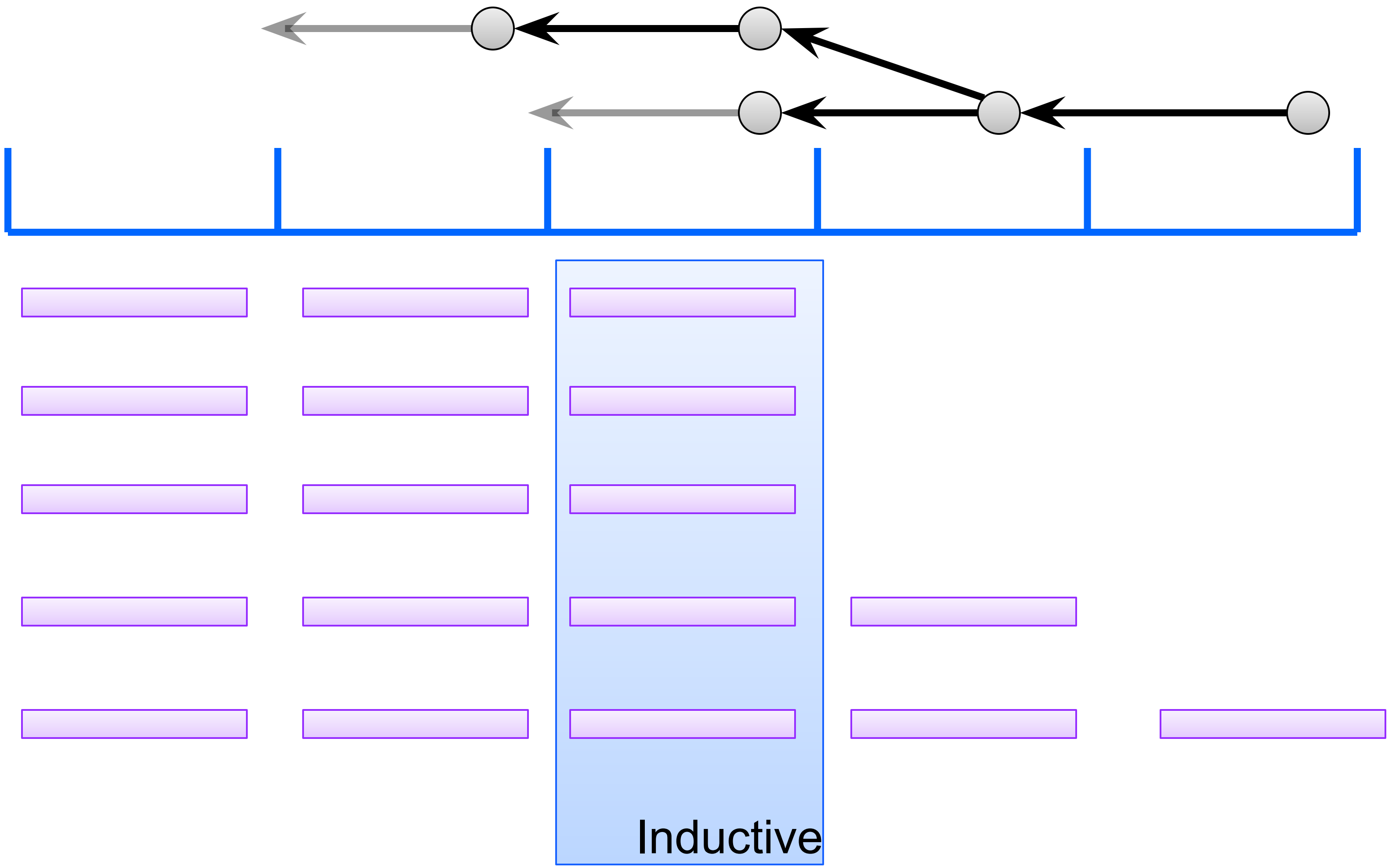


IC3/PDR in Pictures



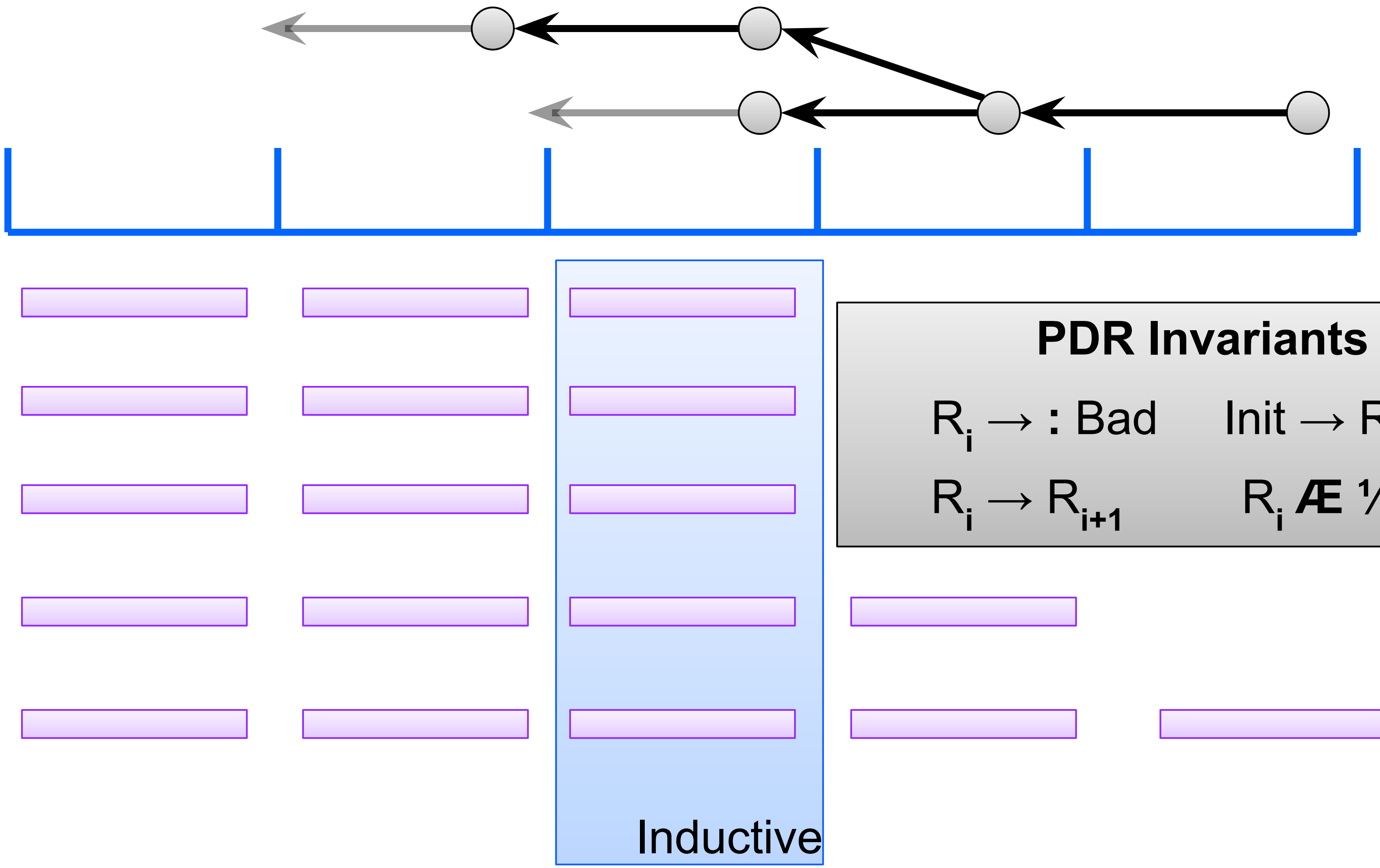
IC3/PDR in Pictures

PdrPush



PdrPush

IC3/PDR in Pictures



PDR Invariants

$R_i \rightarrow$: Bad   Init $\rightarrow R_i$

$R_i \rightarrow R_{i+1}$ R_i Æ ½ $\rightarrow R_{i+1}$

IC3/PDR

Data: Q a queue of counter-examples. Initially, $Q = \emptyset$.
Data: N a level indication. Initially, $N = 0$.
Data: $R_0, R_1, \ldots, R_N$ is a trace. Initially, $R_0 = Init$.
repeat

- **Unreachable** If there is an $i < N$ s.t. $R_{i+1} \rightarrow R_i$, return *Unreachable*.
- **Reachable** If there is an m s.t. $\langle m, 0 \rangle \in Q$ return *Reachable*.
- **Unfold** If $R_N \rightarrow \neg Bad$, then set $N \leftarrow N + 1$, $R_N \leftarrow \top$.
- **Candidate** If for some m, $m \rightarrow R_N \wedge Bad$, then add $\langle m, N \rangle$ to Q.
- **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 $m_0 \wedge m_1' \rightarrow \mathcal{F}(R_i) \wedge m'$, then add $\langle m_0, i \rangle$ to Q.
- **Conflict** For $0 \leq i < N$: given a candidate model $\langle m, i+1 \rangle \in Q$ and clause φ, such that $\neg\varphi \subseteq m$, if $\mathcal{F}(R_i \wedge \varphi) \rightarrow \varphi$, then add φ to R_j, for $j \leq i + 1$.
- **Leaf** If $\langle m, i \rangle \in Q$, $0 < i < N$ and $\mathcal{F}(R_{i-1}) \wedge m'$ is unsatisfiable, then add $\langle m, i+1 \rangle$ to Q.
- **Induction** For $0 \leq i < N$, a clause $(\varphi \vee \psi) \in R_i$, $\varphi \notin R_{i+1}$, if $\mathcal{F}(R_i \wedge \varphi) \rightarrow \varphi$, then add φ to R_j, for each $j \leq i + 1$.

until ∞;

IC3/PDR

Data: Q a queue of counter-examples. Initially, $Q = \emptyset$.
Data: N a level indication. Initially, $N = 0$.
Data: $R_0, R_1, \ldots, R_N$ is a trace. Initially, $R_0 = Init$.
repeat

Unreachable If there is an $i < N$ s.t. $R_{i+1} \rightarrow R_i$, return *Unreachable*.

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 $m_0 \wedge m_1' \rightarrow \mathcal{F}(R_i) \wedge m'$, then add $\langle m_0, i \rangle$ to Q.

Conflict For $0 \leq i < N$: given a candidate model $\langle m, i+1 \rangle \in Q$ and clause φ, such that $\neg\varphi \subseteq m$, if $\mathcal{F}(R_i \wedge \varphi) \rightarrow \varphi$, then add φ to R_j, for $j \leq i+1$.

Induction For $0 \leq i < N$, a clause $(\varphi \vee \psi) \in R_i$, $\varphi \notin R_{i+1}$, if $\mathcal{F}(R_i \wedge \varphi) \rightarrow \varphi$, then add φ to R_j, for each $j \leq i+1$.

until ∞;

Extending PDR to Arithmetic: APDR

Decide$^{\mathcal{A}}$ If $\langle P, i+1\rangle \in Q$ and there is a model $m(\mathbf{v}, \mathbf{v}')$ s.t. $m \models \mathcal{F}(R_i) \wedge P'$, add $\langle P_{\downarrow}, i\rangle$ to Q, where $P_{\downarrow} \in \textsc{Mbp}(\mathbf{v}', m, \mathcal{F}(R_i) \wedge P')$.

Conflict$^{\mathcal{A}}$ For $0 \leq i < N$, given a counterexample $\langle P, i+1\rangle \in Q$ s.t. $\mathcal{F}(R_i) \wedge P'$ is unsatisfiable, add $P^{\uparrow} = \textsc{Itp}(\mathcal{F}(R_i)(\mathbf{v}_0, \mathbf{v}), P)$ to R_j for $j \leq i+1$.

Model Based Projection: MBP(***v***, m, F) [KGC'14]

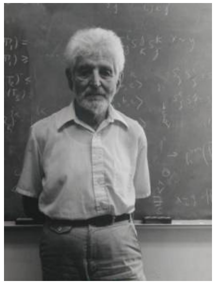
- generates an implicant of 9 ***v*** . F that contains the model m

Counter-examples are monomials (conjunction of inequalities)

Lemmas are clauses (disjunction of inequalities)

APDR computes an (possibly non-convex) QFLRA invariant in CNF

Craig Interpolation Theorem



Theorem (Craig 1957)

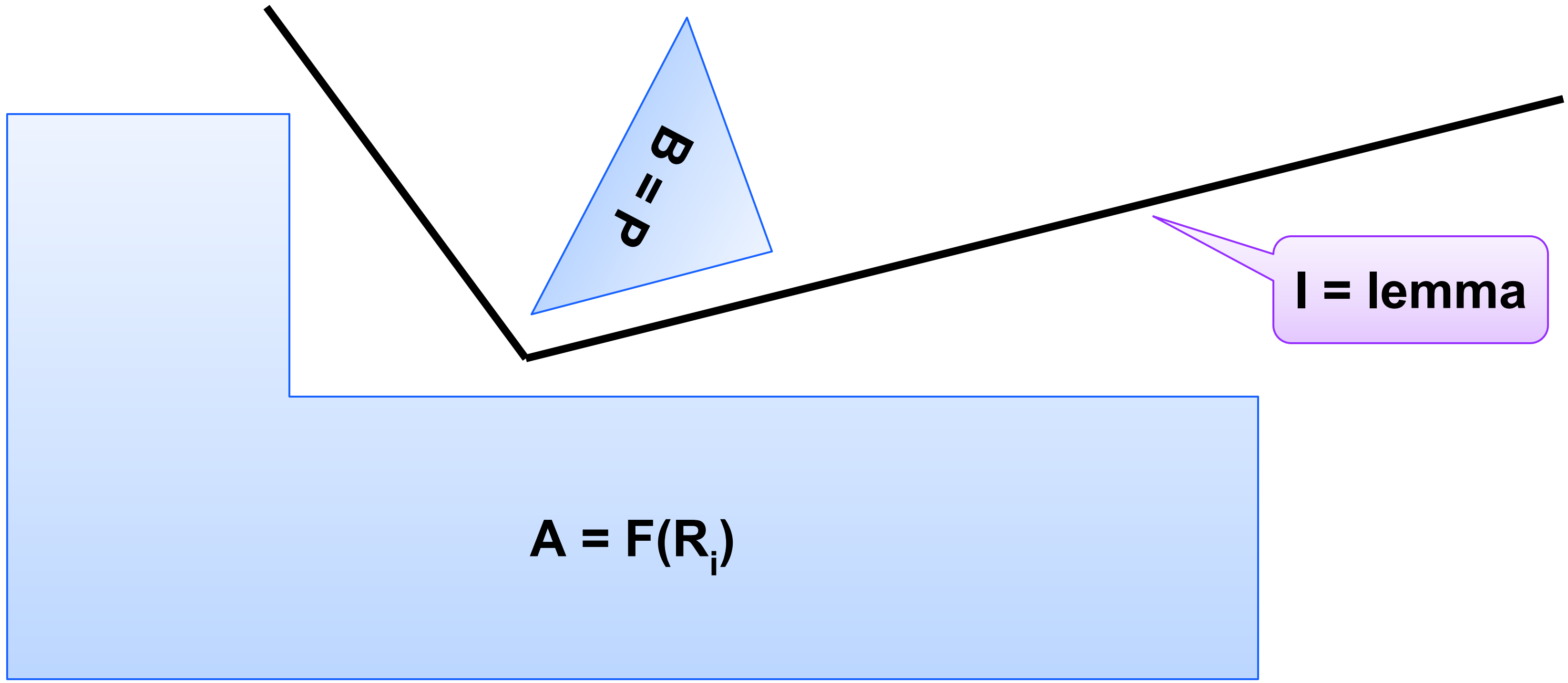
Let A and B be two First Order (FO) formulae such that A) :B, then there exists a FO formula I, denoted ITP(A, B), such that

A) I   I) :B   *atoms*(I) 2 *atoms*(A) Å *atoms*(B)

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

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

Craig Interpolation for Linear Arithmetic

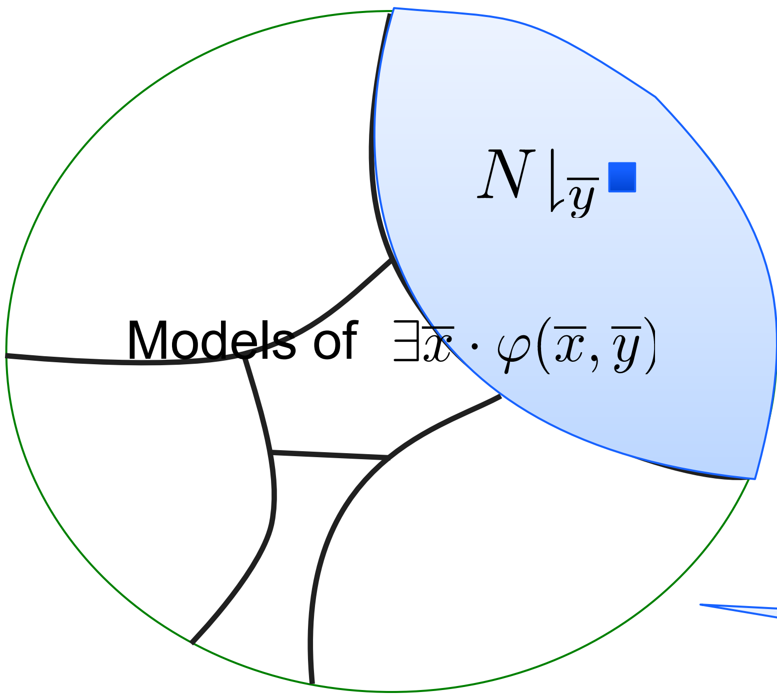


Useful properties of existing interpolation algorithms [CGS10] [HB12]

- I 2 ITP (A, B) then :I 2 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

Model Based Projection

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



1. **find** $N \models \varphi(\overline{x}, \overline{y})$

   (e.g. specific pre-post pair that needs to be generalized)

2. **choose disjunct "covering" *N***

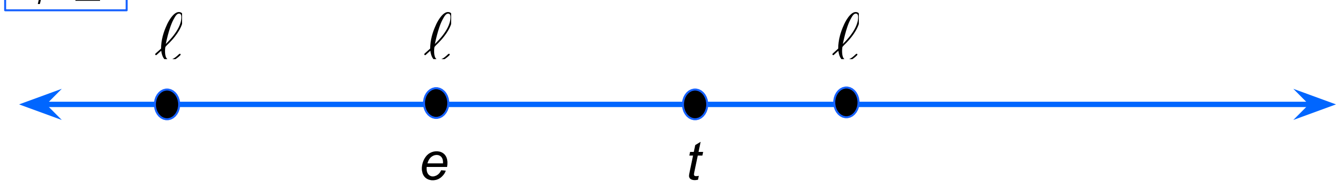
   using virtual substitution

Lazy Quantifier Elimination!

MBP for Linear Rational Arithmetic

$$\exists \ell \cdot (\boxed{\ell = e} \wedge \phi_1) \vee (\boxed{t < \ell} \wedge \ell < u) \vee (\ell < u \wedge \phi_2)$$

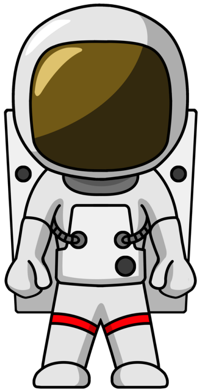
$$\begin{aligned} \equiv \quad & \boxed{(\phi_1 \vee (t < e \wedge e < u) \vee (e < u \wedge \phi_2))} \\ \vee \quad & \boxed{(t < u \vee (t < u \wedge \phi_2))} \\ \vee \quad & \boxed{\phi_2} \end{aligned}$$



pick a disjunct that covers a given model

[1] Cooper, *Theorem Proving in Arithmetic without Multiplication*, 1972
[2] Loos and Weispfenning, *Applying Linear Quantifier Elimination*, 1993
[3] Bjorner, *Linear Quantifier Elimination as an Abstract Decision Procedure*, 2010

Spacer: Solving CHC in Z3



Spacer: solver for SMT-constrained Horn Clauses

- stand-alone implementation in a fork of Z3
- http://bitbucket.org/spacer/code

Support for Non-Linear CHC

- model procedure summaries in inter-procedural verification conditions
- model assume-guarantee reasoning
- uses MBP to under-approximate models for finite unfoldings of predicates
- uses MAX-SAT to decide on an unfolding strategy

Supported SMT-Theories

- Best-effort support for arbitrary SMT-theories
  - data-structures, bit-vectors, non-linear arithmetic
- Full support for Linear arithmetic (rational and integer)
- Quantifier-free theory of arrays
  - only quantifier free models with limited applications of array equality

RESULTS

SV-COMP 2015

http://sv-comp.sosy-lab.org/2015/

4th Competition on Software Verification held (here!) at TACAS 2015

Goals

- Provide a snapshot of the state-of-the-art in software verification to the community.
- Increase the visibility and credits that tool developers receive.
- Establish a set of benchmarks for software verification in the community.

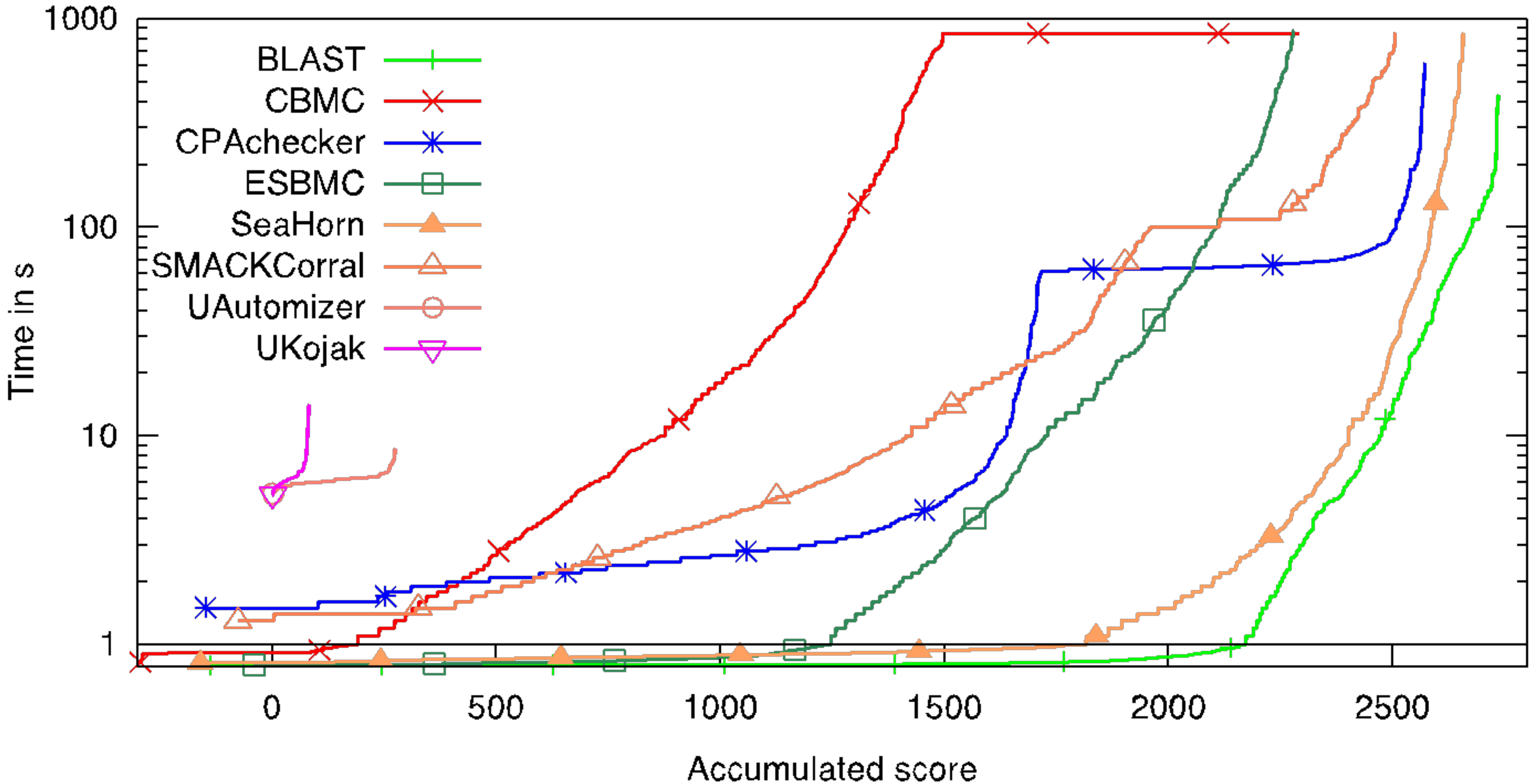
Participants:

- Over 22 participants, including most popular Software Model Checkers and Bounded Model Checkers

Benchmarks:

- C programs with error location (programs include pointers, structures, etc.)
- Over 6,000 files, each 2K – 100K LOC
- Linux Device Drivers, Product Lines, Regressions/Tricky examples
- http://sv-comp.sosy-lab.org/2015/benchmarks.php

Results for DeviceDriver category



Conclusion

SeaHorn (http://seahorn.github.io)

- a state-of-the-art Software Model Checker
- LLVM-based front-end
- CHC-based verification engine
- a framework for research in logic-based verification

The future

- making SeaHorn useful to users of verification technology
  – counterexamples, build integration, property specification, proofs, etc.
- targeting many existing CHC engines
  – specialize encoding and transformations to specific engines
  – communicate results between engines
- richer properties
  – termination, liveness, synthesis

Contact Information

Arie Gurfinkel, Ph. D.
Sr. Researcher
CSC/SSD
Telephone: +1 412-268-5800
Email: info@sei.cmu.edu

Web
www.sei.cmu.edu
www.sei.cmu.edu/contact.cfm

U.S. Mail
Software Engineering Institute
Customer Relations
4500 Fifth Avenue
Pittsburgh, PA 15213-2612
USA

Customer Relations
Email: info@sei.cmu.edu
Telephone: +1 412-268-5800
SEI Phone: +1 412-268-5800
SEI Fax: +1 412-268-6257