Vinta: Verification with INTerpolation and Abstract iterpretation

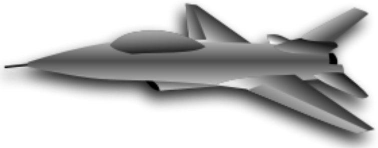
Arie Gurfinkel
Software Engineering Institute
Carnegie Mellon University

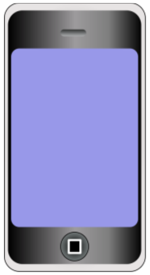
joint work with
Aws Albarghouthi, Yi Li, and Marsha Chechik
University of Toronto

Sagar Chaki, SEI

This work was created in the performance of Federal Government Contract Number FA8721-05-C-0003 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center. The Government of the United States has a royalty-free government-purpose license to use, duplicate, or disclose the work, in whole or in part and in any manner, and to have or permit others to do so, for government purposes pursuant to the copyright license under the clause at 252.227-7013.

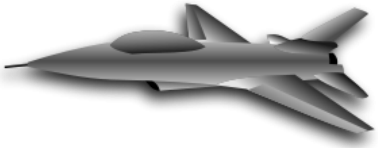
Software is Everywhere

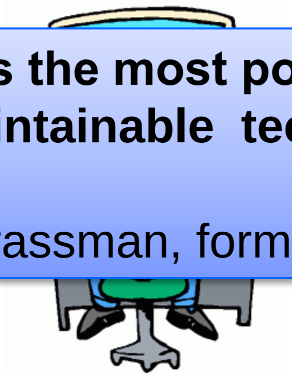








Software is Everywhere





"Software easily rates as the most poorly constructed, unreliable, and least maintainable technological artifacts invented by man"

Paul Strassman, former CIO of Xerox

Recent Software Disasters

In July 2010, The Food and Drug Administration ordered Baxter International to recall all of its Colleague infusion pumps in use and provide a refund or no-cost replacement to United States customers. It has been working with Baxter since 1999 to correct numerous device flaws. Some of the issues were caused by simple buffer overflow.

In January 2011, two German researchers have shown that most “feature” mobile phones can be “killed” by sending a simple SMS message (**SMS of Death**). The attack exploits many bugs in the implementation of SMS protocol in the phones. It can potentially bring down all mobile communication…

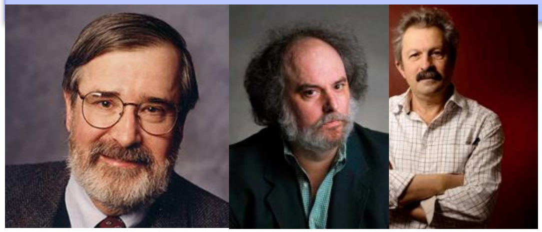
On August 1, 2012, Knight Capital's bugs in high-frequency trading algorithm caused a pre-tax loss of $440m. The nature of the bug was described as a "technology breakdown".

Automated Software Analysis

Program → **Automated Analysis** → Correct / Incorrect

Software Model Checking with Predicate Abstraction

e.g., Microsoft's SDV



Abstract Interpretation with Numeric Abstraction

e.g., ASTREE, Polyspace

Motivation

Abstract Interpretation is one of the most scalable approaches for program verification

But, in practice, AI suffers from many false positives due to

- imprecise operations: join, widen
- imprecise semantics of operations: abstract post
- in-expressivity of abstract domains: weakly relational facts, …

No CounterExamples and No Refinement

Goal: Enhance Abstract Interpretation with Interpolation-based refinement strategy

Outline (of the rest of the talk)

Numeric Abstract Interpretation

Vinta illustrated

- Abstract Interpretation with Unfoldings
- Abstract-Interpretation guided DAG-Interpolation Refinement

Implementation

Results of Software Verification Competition

Secret Sauce

Conclusions and Future Directions

Numeric Abstract Interpretation

Analysis is restricted to a fixed Abstract Domain

Abstract Domain ≡ "a (possibly infinite) set of predicates from a fixed theory" + efficient (*abstract*) operations

Common Numeric Abstract Domains

Abstract Domain	Abstract Elements
Sign	$0 < x, \quad x = 0, \quad x > 0$
Box (or Interval)	$c_1 \le x \le c_2$
Octagon	$\pm x \pm y \le c$
Polyhedra	$a_1x_1 + a_2x_2 + a_3x_3 + a_4 \le 0$

Legend	
x,y	program variables
c, c_i, a_i	numeric constants

Abstract Interpretation w/ Box Domain (1)

Program

```
if (3 <= y1 <= 4) {
  x1 := y1-2;
  x2 := y1+2;
}
else if (3 <= y2 <= 4) {
  x1 := y2-2;
  x2 := y2+2;
}
else return;

assert (5 <= x1 + x2 <= 10);
```

3 <= y1 <= 4

3 <= y1 <= 4
1 <= x1 <= 2
5 <= x2 <= 6

3 <= y2 <= 4

3 <= y2 <= 4
1 <= x1 <= 2
5 <= x2 <= 6

1<=x1<=2
5<=x2<=6

Steps: 1 2 3 4 5

Abstract Interpretation w/ Box Domain (2)

Program

```
x := 0

while (x < 1000) {

  x := x + 1;

}

assert (x == 1000);
```

0<= x <=2

x = 0

0<= x <=2

x = 0

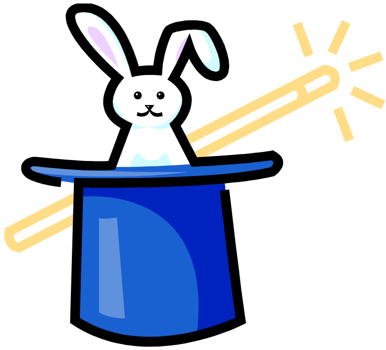
1<= x <=3

0<= x <=1000

0<= x < 1000

1<= x <= 1000

x = 1000



widening

Steps: 1 2 3 4 5 6 7 8 9 10 11 12 13 14

Abstract Domain as an Interface

interface AbstractDomain(V) :

- V – set of variables
- A – abstract elements
- E – expressions
- S – statements

abstract

concretize

α : E → A	**γ** : A → E	**meet** : A × A → A
isTop : A → bool	**isBot** : A → bool	**join** : A × A → A
leq : A × A → bool	**αPost** : S → (A → A)	**widen** : A × A → A

order

abstract transformer

All operations are over-approximations, e.g.,

$$\boldsymbol{\gamma}\,(a)\ ||\ \boldsymbol{\gamma}\,(b) \Rightarrow \boldsymbol{\gamma}\,(\ \textbf{join}\ (a, b)\ )$$

$$\boldsymbol{\gamma}\,(a)\ \ \&\&\ \boldsymbol{\gamma}\,(b) \Rightarrow \boldsymbol{\gamma}\,(\textbf{meet}\ (a,b)\ )$$

Example: Box Abstract Domain

$1 \le x \le 10$ $\xrightarrow[\text{abstract}]{\alpha}$ $(1, 10)$ $\xrightarrow[\text{concretize}]{\gamma}$ $1 \le x \le 10$

Definition of Operations	Examples
(a, b) **meet** (c, d) = (max(a,c), min(b,d))	(1, 10) **meet** (2, 12) = (2,10)
(a, b) **join** (c, d) = (min(a,c),max(b,d))	(1, 3) **join** (7, 12) = (1,12)
αPost (x := x + 1) ((a, b)) = (a+1, b+1)	(1, 10) + 1 = (2, 11)

over-approximation

Abstract Interpretation w/ Box Domain (3)

Program

```
assume (i=1 || i=2)
if (i = 1)
  x1 := i;
else if (i = 2)
  x2 := -4;

if (i = 1)
  assert (x1 > 0);
else if (i = 2)
  assert (x2 < 0);
```

`1 <= i <= 2`

`i=1`

`i=1 && x1=1`

`i=2`

`i=2 && x2=-4`

`1 <= i <= 2`

`i=1`

`i=2`

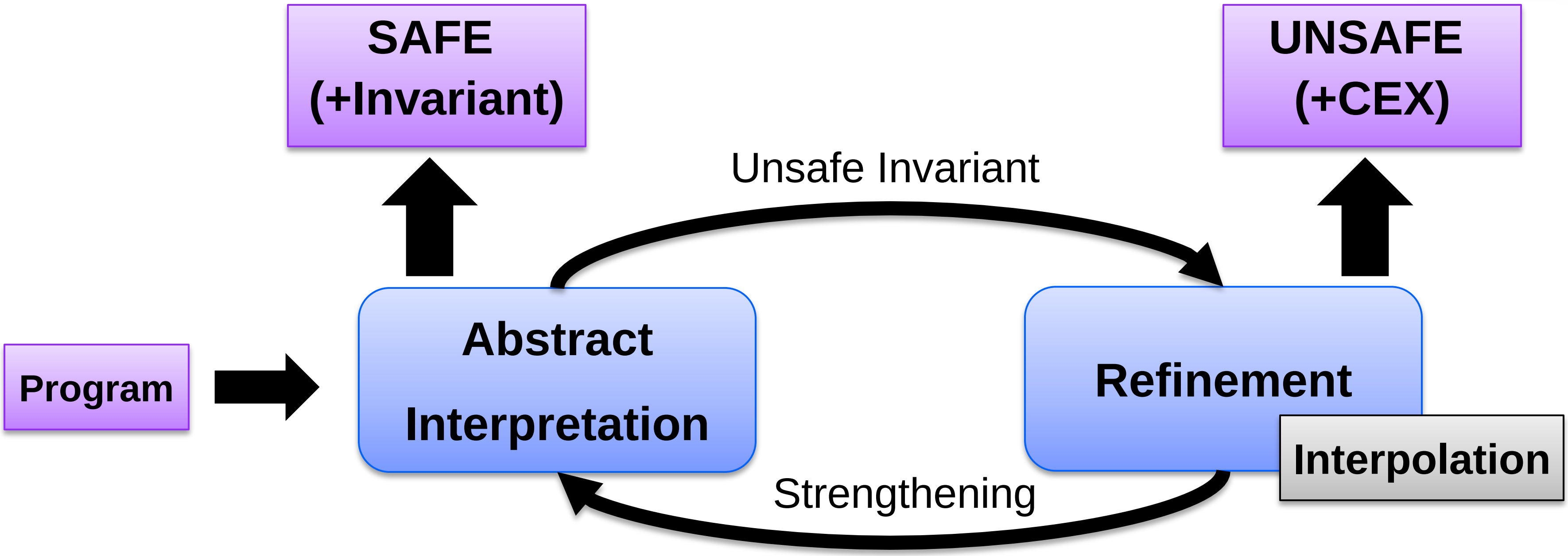
Loss of precision due to join

False Positive

Steps: 1 2 3 4 5 6 7 8

Vinta: Verification with INTERP and AI

వింత



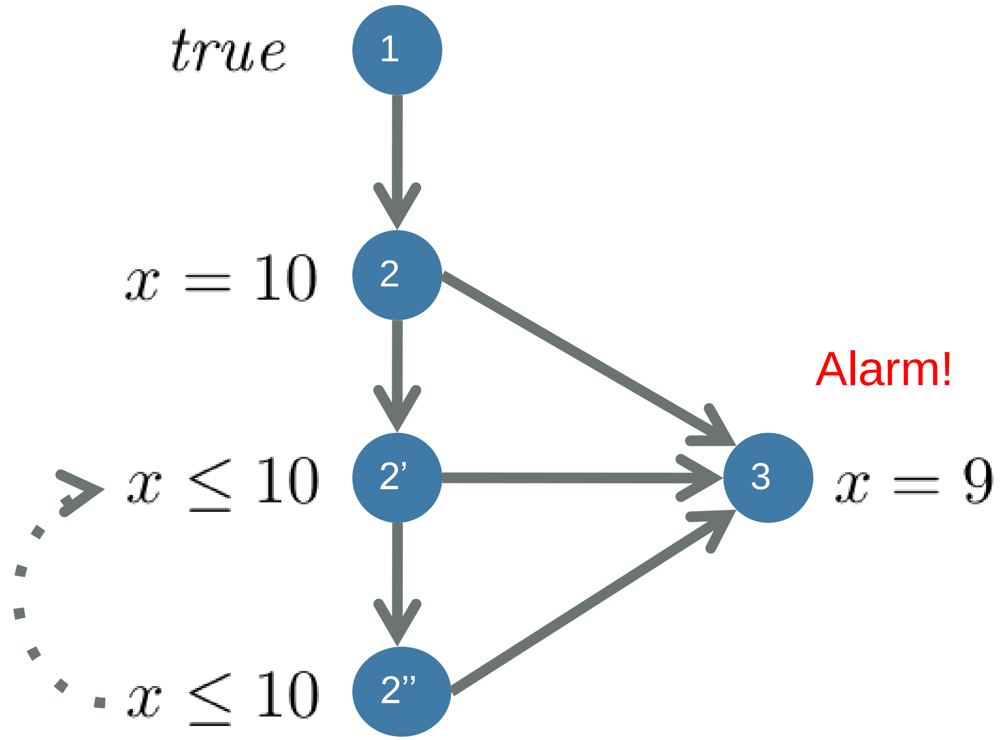
- uses Cutpoint Graph (CPG)
- maintains an unrolling of CPG
- computes disjunctive invariants
- uses novel powerset widening

- uses SMT to check for CEX
- DAG Interpolation for Refinement
- Guided by AI-computed Invs
- Fills in “gaps” in AI

Example: AI phase

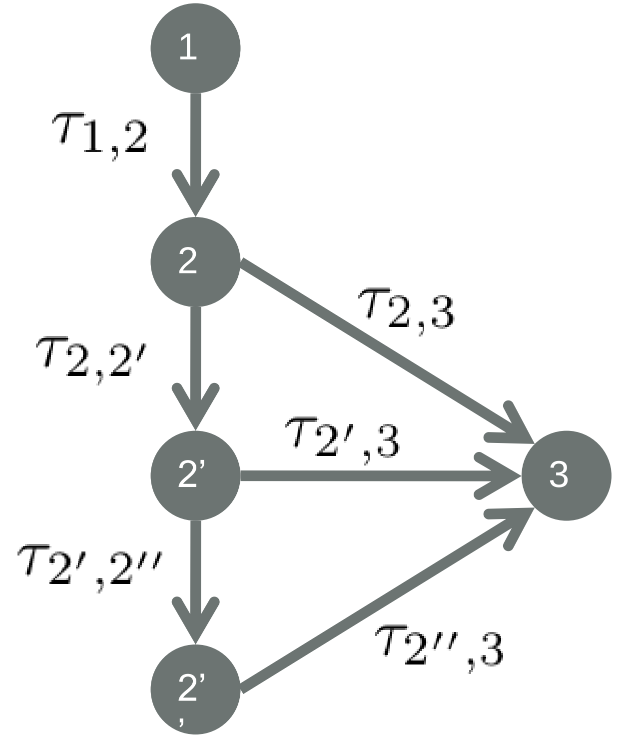
- *Exploration:* WTO
- *Abstract Domain:* Intervals
- *Side effect:* Labelled CFG unrolling

```
1: x = 10;

2: while (*)
        x = x - 2;

   if (x == 9)
3:   error();
```



Verification Conditions

```
1: x = 10;

2: while (*)
      x = x - 2;

   if (x == 9)
3:   error();
```



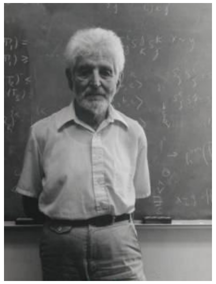
Instruction encoding

$$\tau_{1,2} \equiv x_0 = 0$$
$$\tau_{2,2'} \equiv x_1 = x_0 + 1$$
$$\ldots$$

Control-flow encoding

$$v_1$$
$$v_1 \Rightarrow \tau_{1,2} \wedge v_2$$
$$v_2 \Rightarrow (\tau_{2,2'} \wedge v_{2'}) \vee (\tau_{2,3} \wedge v_3)$$
$$\ldots$$

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

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

Theorem (McMillan 2003)

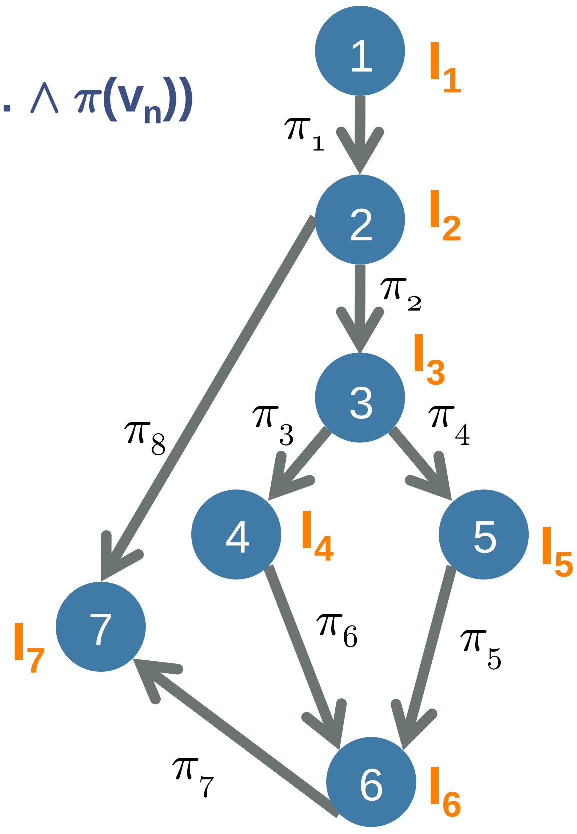
A Craig interpolant ITP(A, B) can be effectively constructed from a resolution proof of unsatisfiability of $A \wedge B$

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

DAG Interpolants [TACAS'12]

Given a DAG G = (V, E) and a labeling of edges π:E$\rightarrow$Expr. A ***DAG Interpolant*** (if it exists) is a labeling I:V$\rightarrow$Expr such that

- for any path $v_0, \ldots, v_n$, and $0 < k < n$,
  $\mathbf{I(v_k) = ITP\ (\pi(v_0) \wedge \ldots \wedge \pi\ (v_{k-1}),\quad \pi(v_k) \wedge \ldots \wedge \pi(v_n))}$
- $\forall$ **(u, v)** $\in$ **E . (I(u)** $\wedge$ π **(u, v))** $\Rightarrow$ **I(v)**



$I_2 = ITP\ (\pi_1,\quad \pi_8)$
$I_2 = ITP\ (\pi_1,\quad \pi_2 \wedge \pi_3 \wedge \pi_6 \wedge \pi_7)$
…

$(I_1 \wedge \pi_1) \Rightarrow I_2$
$(I_2 \wedge \pi_8) \Rightarrow I_7$
$(I_2 \wedge \pi_2) \Rightarrow I_3$
…

DAG Interpolation Algorithm [TACAS'12]

Reduce DAG Interpolation to Sequence Interpolation!

```
DagItp ((V, E), π)
{
   (A0, …, An) = Encode(V, E, π)

   (I1, …, In-1) = SeqItp(A0, …, An)

   for i in [1, n-1] do Ji = Clean(Ii)

   return (J1, …, Jn-1)
}
```

Encode input DAG by a set of constraints. One constraint per vertex.

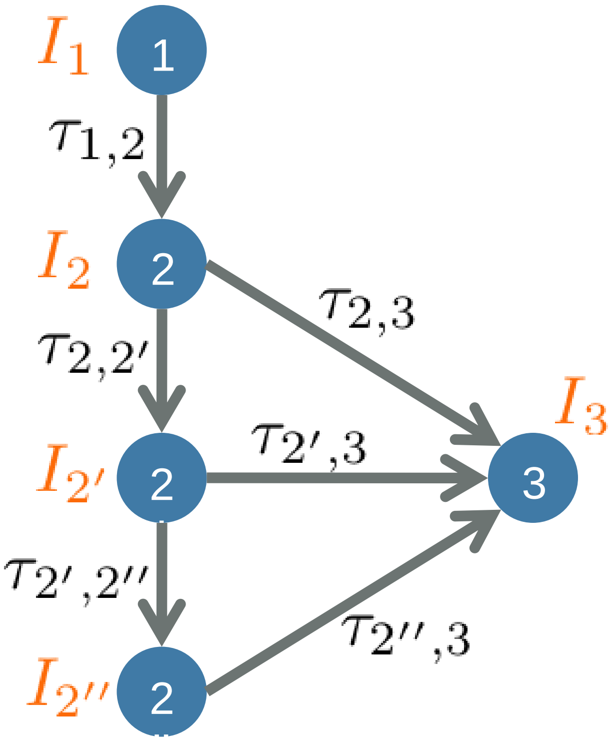
Compute interpolant sequence. One interpolant per vertex.

Remove out-of-scope variables

In our running example…

$$I_1 \equiv true$$
$$I_3 \equiv false$$

For any edge (i, j)
$$I_i \wedge \tau_{i,j} \Rightarrow I_j$$



How to use the results of AI here?

Restricted DAG Interpolants

$$I_1 \equiv true$$

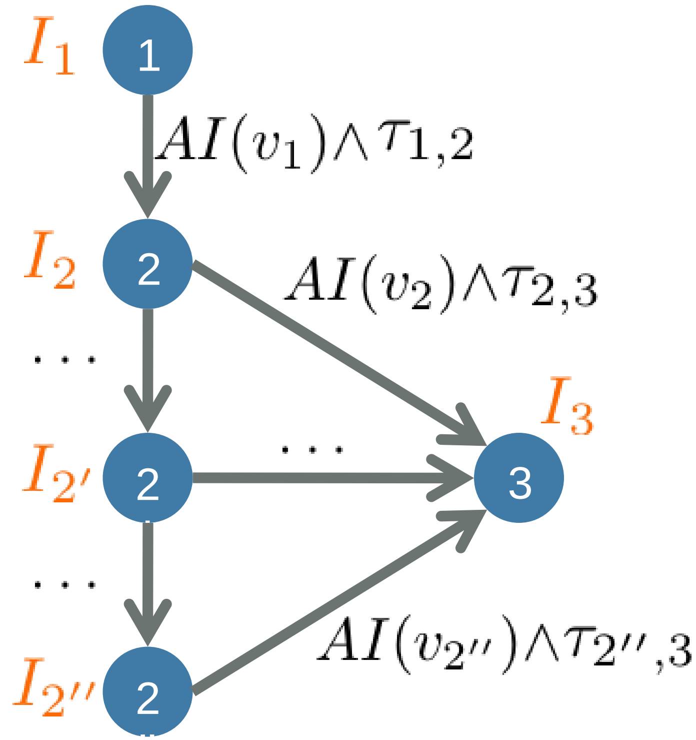
$$I_3 \triangleq \boxed{AI(v_3)} \Rightarrow false$$

For any edge (i, j)

$$I_i \wedge \boxed{AI(v_i)} \wedge \tau_{i,j} \Rightarrow I_j$$

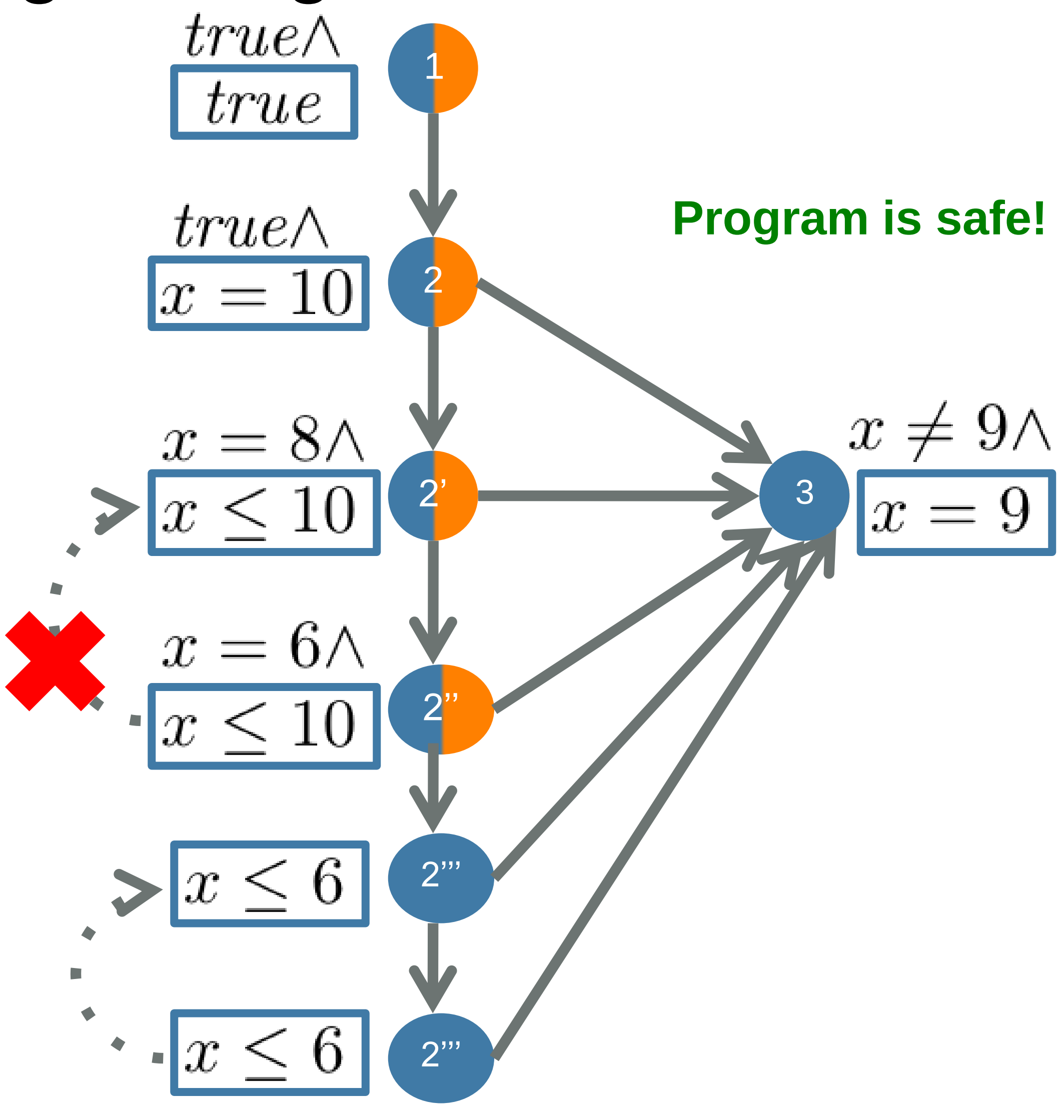
Vertex labels from AI

$$AI : V \to Expr$$



Refinement: Strengthening

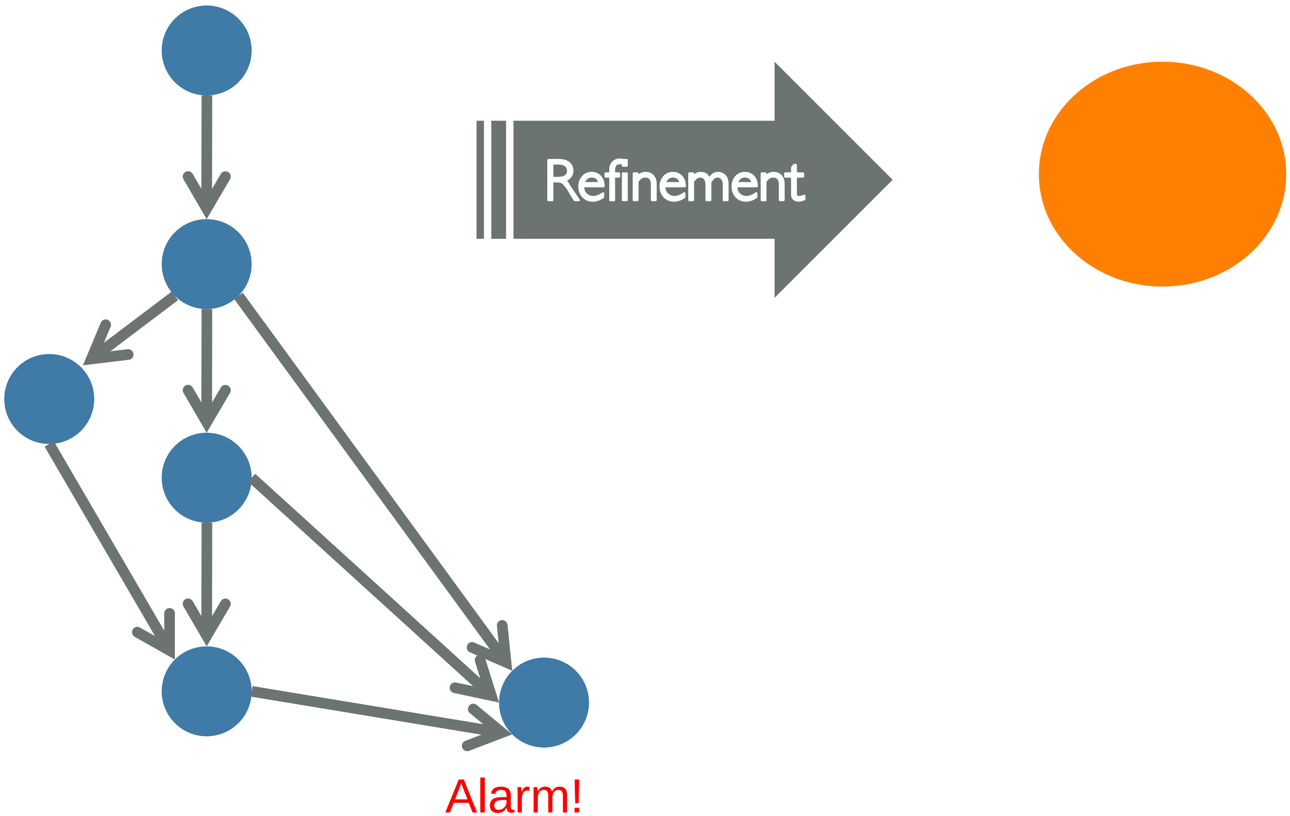
```
1: x = 10;

2: while (*)
       x = x - 2;

   if (x == 9)
3:   error();
```



VINTA from 30,000 ft

Abstract Interpretation

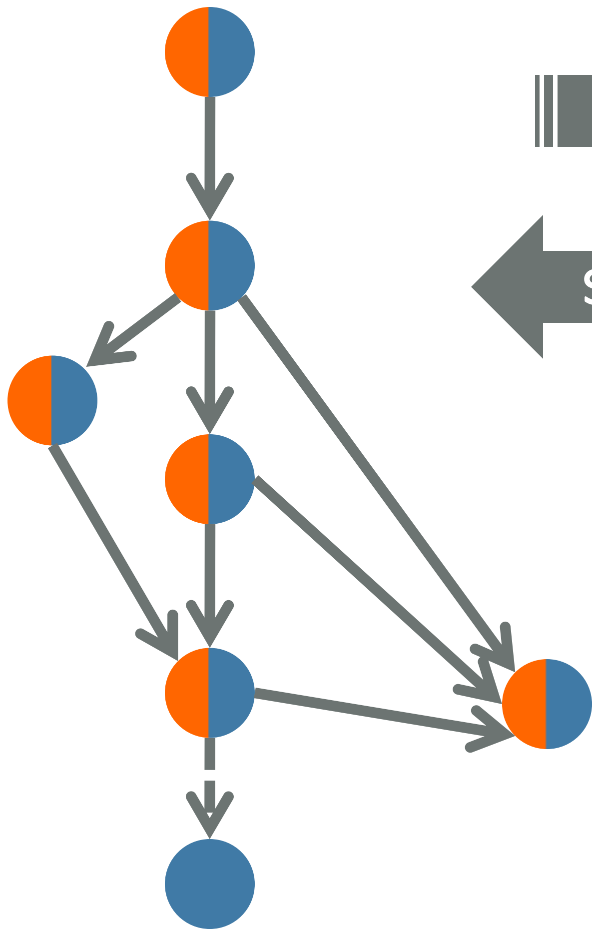
Refinement w/ DAG Interpolants

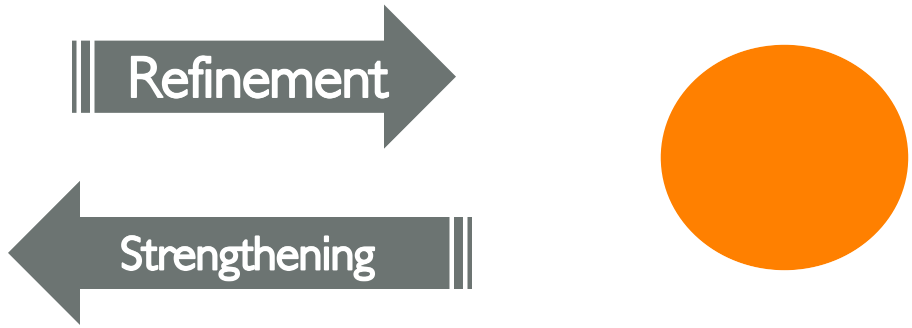


VINTA from 30,000 ft

Abstract Interpretation

Refinement w/ DAG Interpolants





Refinement recovers imprecision in:

- Join, Widening
- Abstract Transformer
- Inexpressive Abstract Domain

Vinta is part of UFO

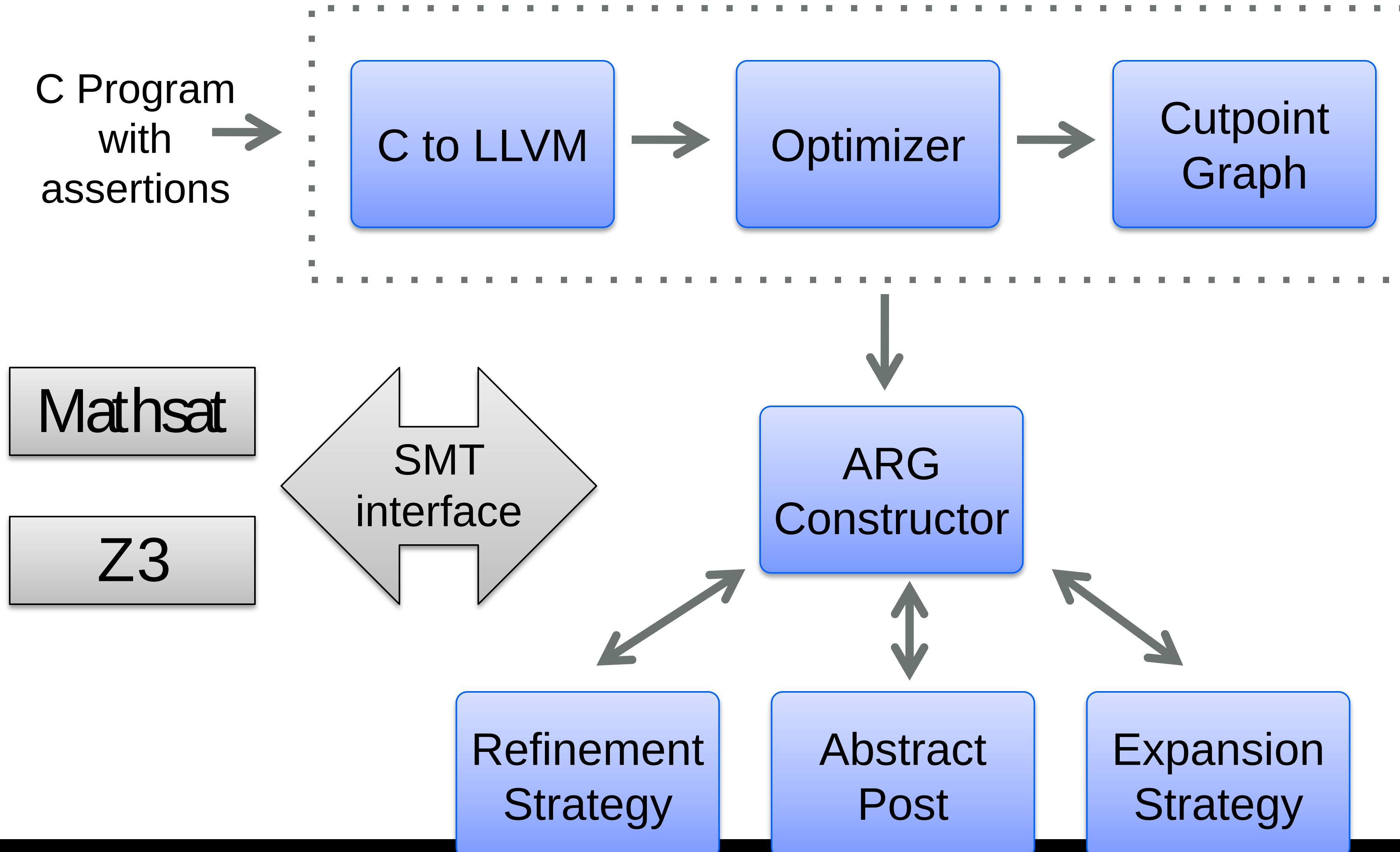
- A *framework* and a *tool* for software verification
- Tightly integrates *interpolation-* and *abstraction*-based techniques

Check it out at:
http://bitbucket.org/arieg/ufo

References:

[SAS12] Craig Interpretation
[CAV12] UFO: A Framework for Abstraction- and Interpolation-based Software Verification
[TACAS12] From Under-approximations to Over-approximations and Back
[VMCAI12] Whale: An Interpolation-based Algorithm for Interprocedural Verification

Implementation in UFO Framework



Software Verification Competitoion (SV-COMP 2013)

SV-COMP 2013

http://sv-comp.sosy-lab.org/2013/

2nd Software Verification Competition held at TACAS 2013

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:

- **BLAST**, **CPAChecker-Explicit**, **CPAChecker-SeqCom**, CSeq, **ESBMC**, **LLBMC**, Predator, **Symbiotic**, Threader, **UFO**, **Ultimate**

Benchmarks:

- C programs with ERROR label (programs include pointers, structures, etc.)
- Over 2,000 files, each 2K – 100K LOC
- Linux Device Drivers, SystemC, “Old” BLAST, Product Lines
- http://sv-comp.sosy-lab.org/2013/benchmarks.php

SV-COMP 2013: Scoring Scheme

Points	Reported Result	Description
0	UNKNOWN	Failure to compute verification result, out of resources, program crash.
+1	FALSE/UNSAFE correct	The error in the program was found and an error path was reported.
-4	FALSE/UNSAFE wrong	An error is reported for a program that fulfills the property (false alarm, incomplete analysis).
+2	TRUE/SAFE correct	The program was analyzed to be free of errors.
-8	TRUE/SAFE wrong	The program had an error but the competition candidate did not find it (missed bug, unsound analysis).

Ties are broken by run-time

UFO/VINTA Results

VINTA was the main reasoning engine used by UFO at SV-COMP

UFO won in 4 categories

- Control Flow Integers (perfect score)
- Product Lines (perfect score)
- Device Drivers
- SystemC

VINTA with Box domain was most competitive for bug-discovery

VINTA with Boxes domain was most competitive for proving safety

http://sv-comp.sosy-lab.org/2013/results/index.php

The Secret Sauce

UFO Front-End

Boxes Abstract Domain

Parallel Verification Strategy

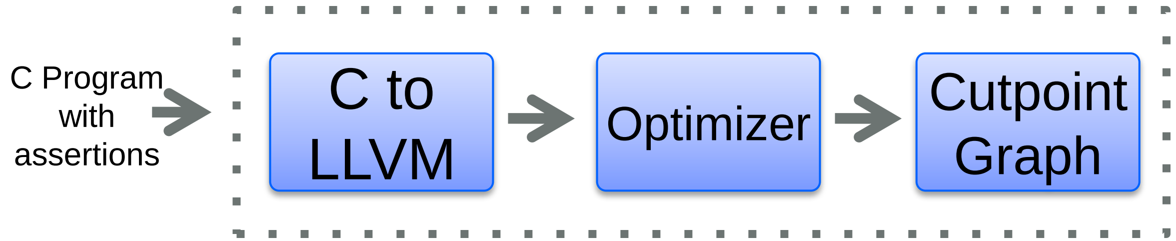
UFO Front End

In principle simple, but in practice very messy

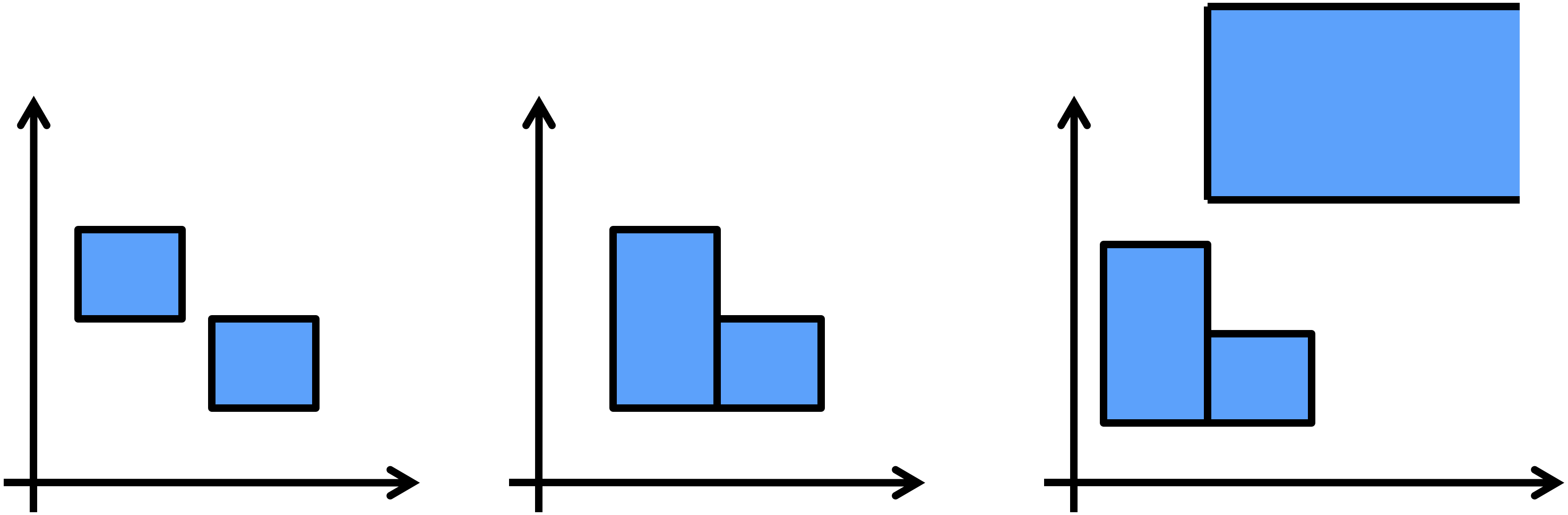
- CIL passes to normalize the code (library functions, uninitialized vars, etc.)
- `llvm-gcc` (without optimization) to compile C to LLVM bitcode
- `llvm opt` with many standard, custom, and modified optimizations
  - lower pointers, structures, unions, arrays, etc. to registers
  - constant propagation + many local optimizations
  - difficult to preserve *indented* semantics of the benchmarks
  - based on very old LLVM 2.6 (newer version of LLVM are “too smart”)

Many benchmarks discharged by front-end alone

- 1,321 SAFE (out of 1,592) and 19 UNSAFE (out of 380)



Boxes Abstract Domain: Semantic View



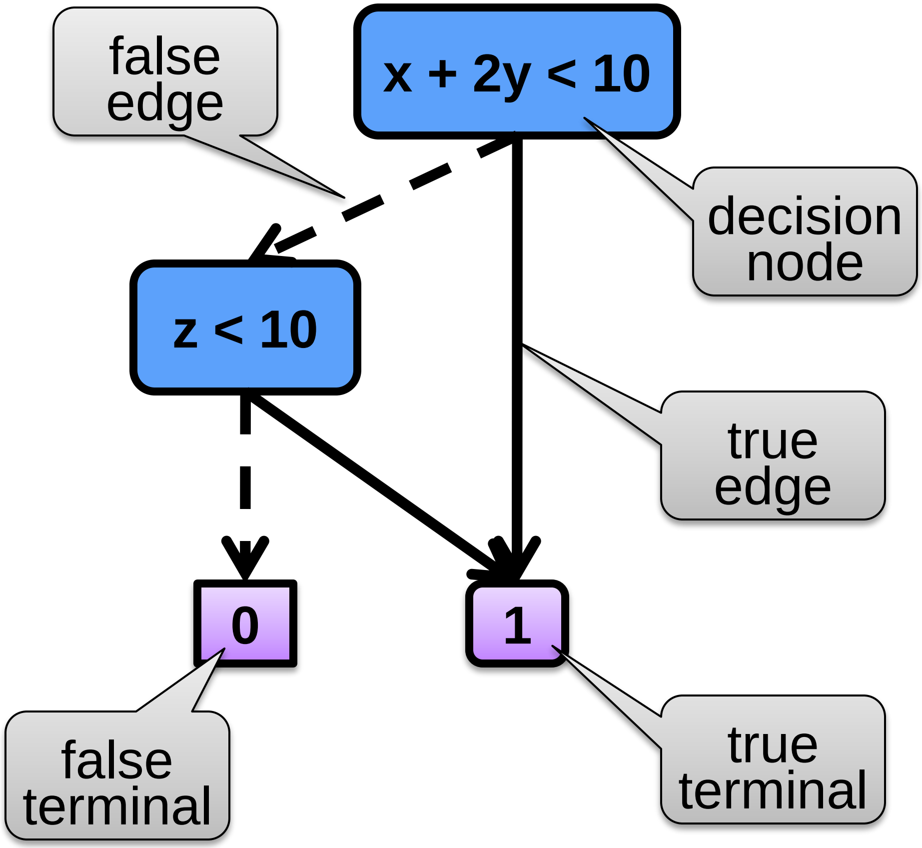
Boxes are “finite union of box values”

(alternatively)

Boxes are “Boolean formulas over interval constraints”

Linear Decision Diagrams in a Nutshell*

Linear Decision Diagram



Linear Arithmetic Formula

(x + 2y < 10) **OR**
(x + 2y ≥ 10 **AND** z < 10)

Compact Representation

- Sharing sub-expressions
- Local numeric reductions
- Dynamic node reordering

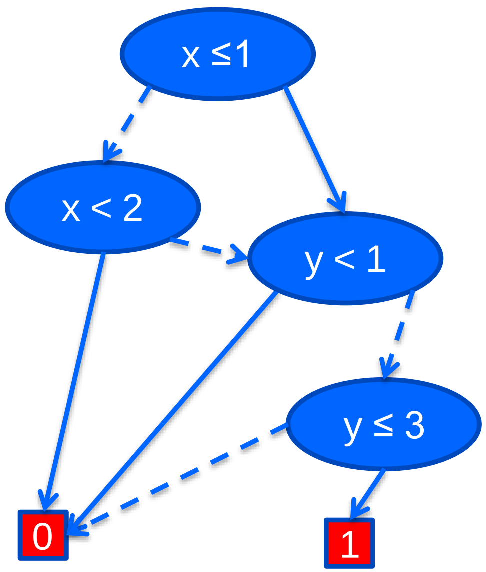
Operations

- Propositional (AND, OR, NOT)
- Existential Quantification

*joint work w/ Ofer Strichman

Boxes: Representation

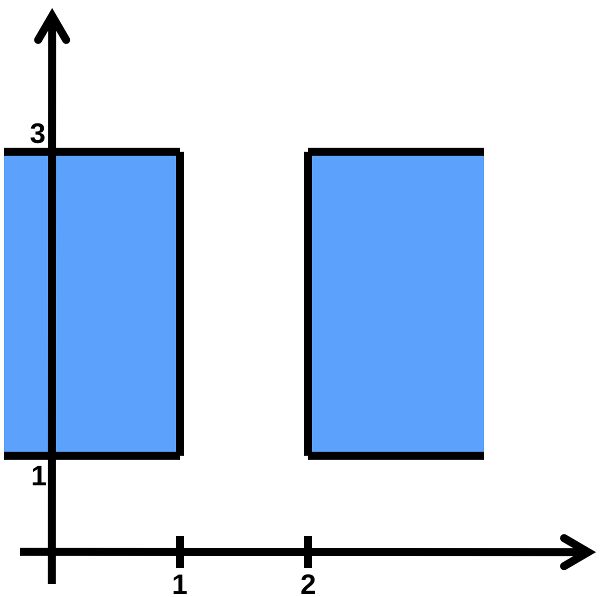
LDD



Syntax

$(x \leq 1 \,||\, x \geq 2)$
$\&\&$
$1 \leq y \leq 3$

Semantics

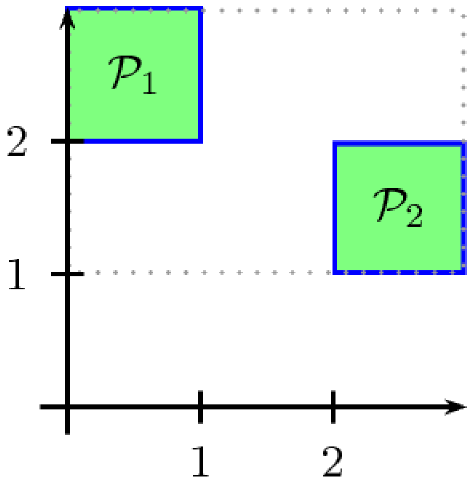


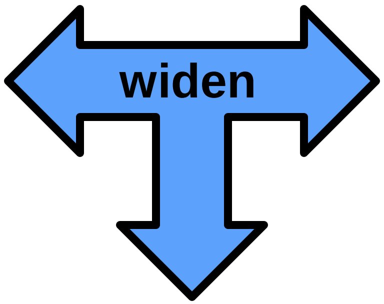
Represented by (Interval) Linear Decision Diagrams (LDD)

- BDDs + non-terminal nodes are labeled by interval constraints + extra rules
- retain complexity of BDD operations
- canonical representation for Boxes Abstract Domain
- available at http://lindd.sf.net

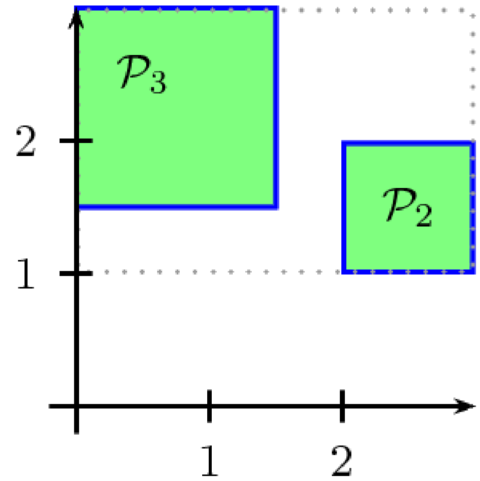
Widening: The Problem

$(x \leq 1 \wedge 2 \leq y \leq 3) \vee$
$(2 \leq x \leq 3 \wedge 1 \leq y \leq 2)$





$(x \leq 1.5 \wedge 1.5 \leq y \leq 3) \vee$
$(2 \leq x \leq 3 \wedge 1 \leq y \leq 2)$

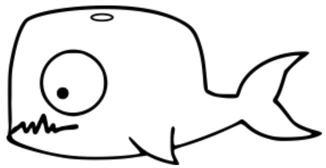


Parallel Verification Strategy

Run 7 verification strategies in parallel until a solution is found

- **cpredO3**
  - all LLVM optimizations + Cartesian Predicate Abstraction
- **bpredO3**
  - all LLVM optimizations + Boolean PA + 20s TO
- **bigwO3**
  - all LLVM optimizations + BOXES + non-aggressive widening + 10s TO
- **boxesO3**
  - all LLVM optimizations + BOXES + aggressive widening
- **boxO3**
  - all LLVM optimizations + BOX + aggressive widening + 20s TO
- **boxesO0**
  - minimal LLVM optimizations + BOXES + aggressive widening
- **boxbpredO3**
  - all LLVM opts + BOX + Boolean PA + aggressive widening + 60s TO

Vinta Family



Whale [VMCAI12]

- Interpolation-based interprocedural analysis
- Interpolants as procedure summaries
- State/transition interpolation
  - a.k.a. Tree Interpolants

UFO [TACAS12]

- Refinement with *DAG interpolants*
- Tight integration of interpolation-based verification with predicate abstraction

వింత

Vinta [SAS12]

- Refinement of Abstract Interpretation (AI)
- AI-guided DAG Interpolation

Future Work

Symbolic Abstraction

- An abstract domain based on SMT-formulae

DAG Interpolation via (Non-Recursive) Horn Clause Solving

- DAG Interpolation is an instance of Horn Clause Satisfiability Problem
- Need to better understand how to combine Interpolation and Inductive Generalization-based solutions

Tighter integration of existing engines and passes

- our current solution is “embarrassingly parallel”
- there are many other strategies with better defined communication between components and “failed” attempts

Concurrency

Contact Information

Presenter
Arie Gurfinkel
RTSS
Telephone: +1 412-268-7788
Email: arie@cmu.edu

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

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

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

THE END