UFO: From Underapproximations to Overapproximations and Back!

Arie Gurfinkel (SEI/CMU)

with Aws Albarghouthi and Marsha Chechik (U. of Toronto)

and Sagar Chaki (SEI/CMU), and Yi Li (U. of Toronto)

Automated Software Analysis

Program

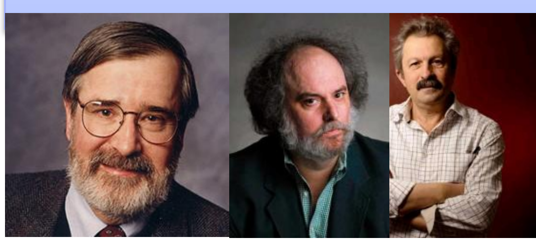
Automated Analysis

Correct

Incorrect

Software Model Checking with Predicate Abstraction

e.g., Microsoft's SDV



Abstract Interpretation with Numeric Abstraction

e.g., ASTREE, Polyspace

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

Outline

Over- and Under-approximation Driven Approaches

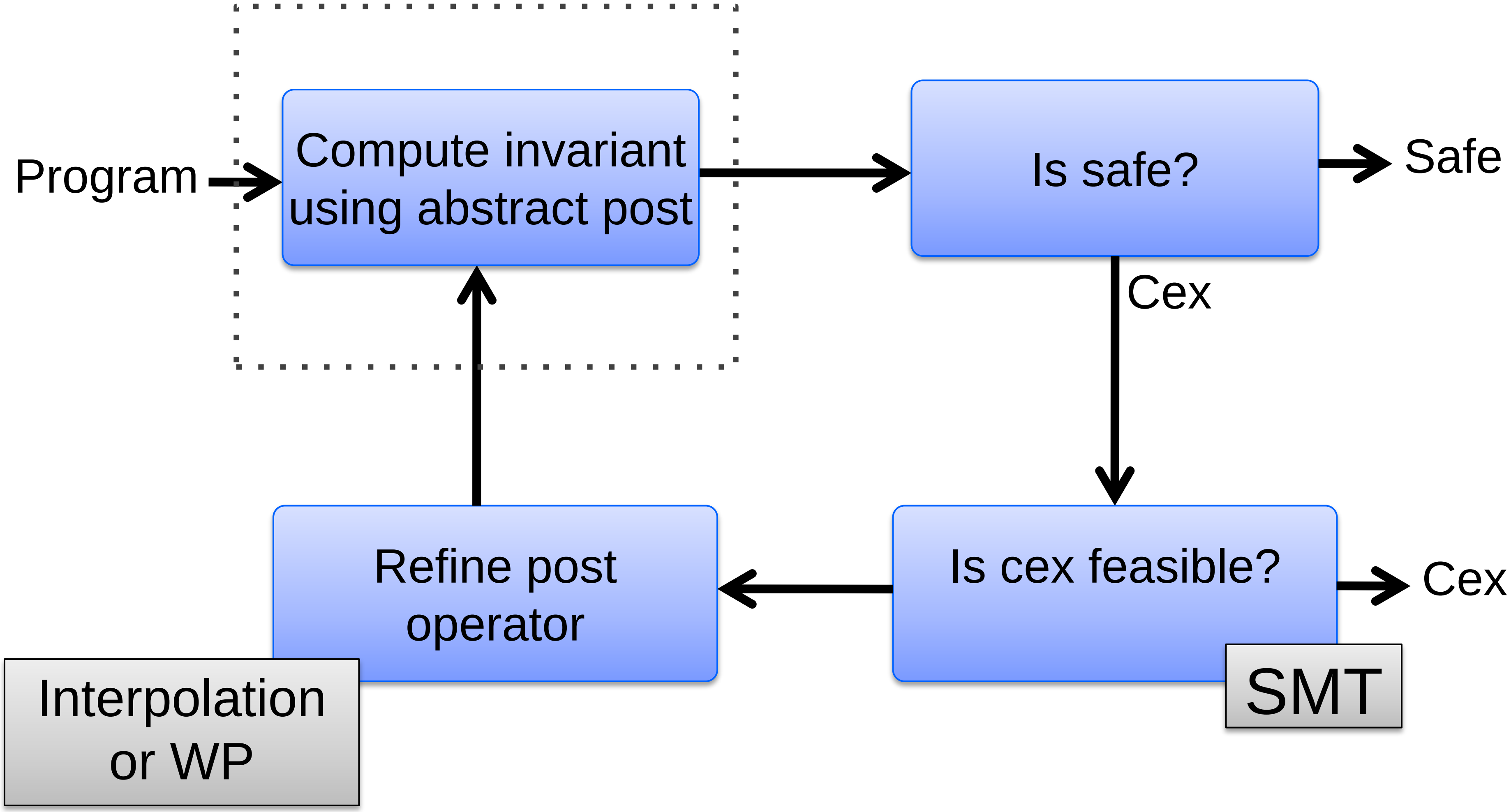
UFO: From Under- to Over- and Back!

- Exploration Strategy
- Refinement Strategy

Software Verification Competition (SV-COMP'13)

Conclusion

Overapproximation-driven Approach (CEGAR)



e.g., BLAST, SLAM, CPAChecker, YaSM, SATAbs, etc.

Is ERROR Reachable?

Program

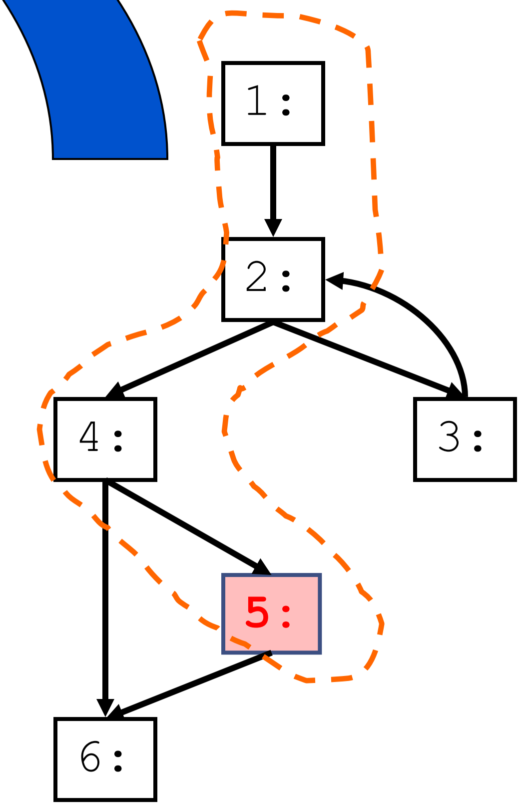
```
1: int x = 2;
   int y = 2;
2: while (y <= 2)
3:       y = y - 1;
4: if (x == 2)
5:       ERROR:;
6:
```

Need This!

Abstraction

```
1: ;

2: while (*)
3:       ;
4: if (*)
5:       ERROR:;
6:
```

Over-Approximation



CEGAR steps

Abstract → Translate → Check → Validate →

Over-Driven: Is ERROR Reachable?

Program

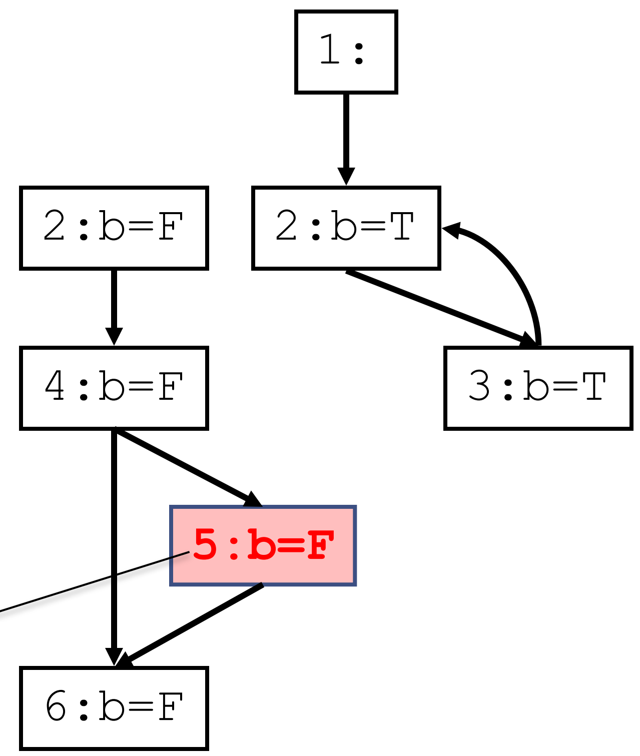
```
1: int x = 2;
   int y = 2;
2: while (y <= 2)
3:   y = y - 1;
4: if (x == 2)
5:     ERROR:;
6:
```

Abstraction

(with `y<=2`)

```
bool b is (y <= 2)
1: b = T;

2: while (b)
3:   b = b ? T : *;
4: if (*)
5:     ERROR:;
6:
```

Over-Approximation

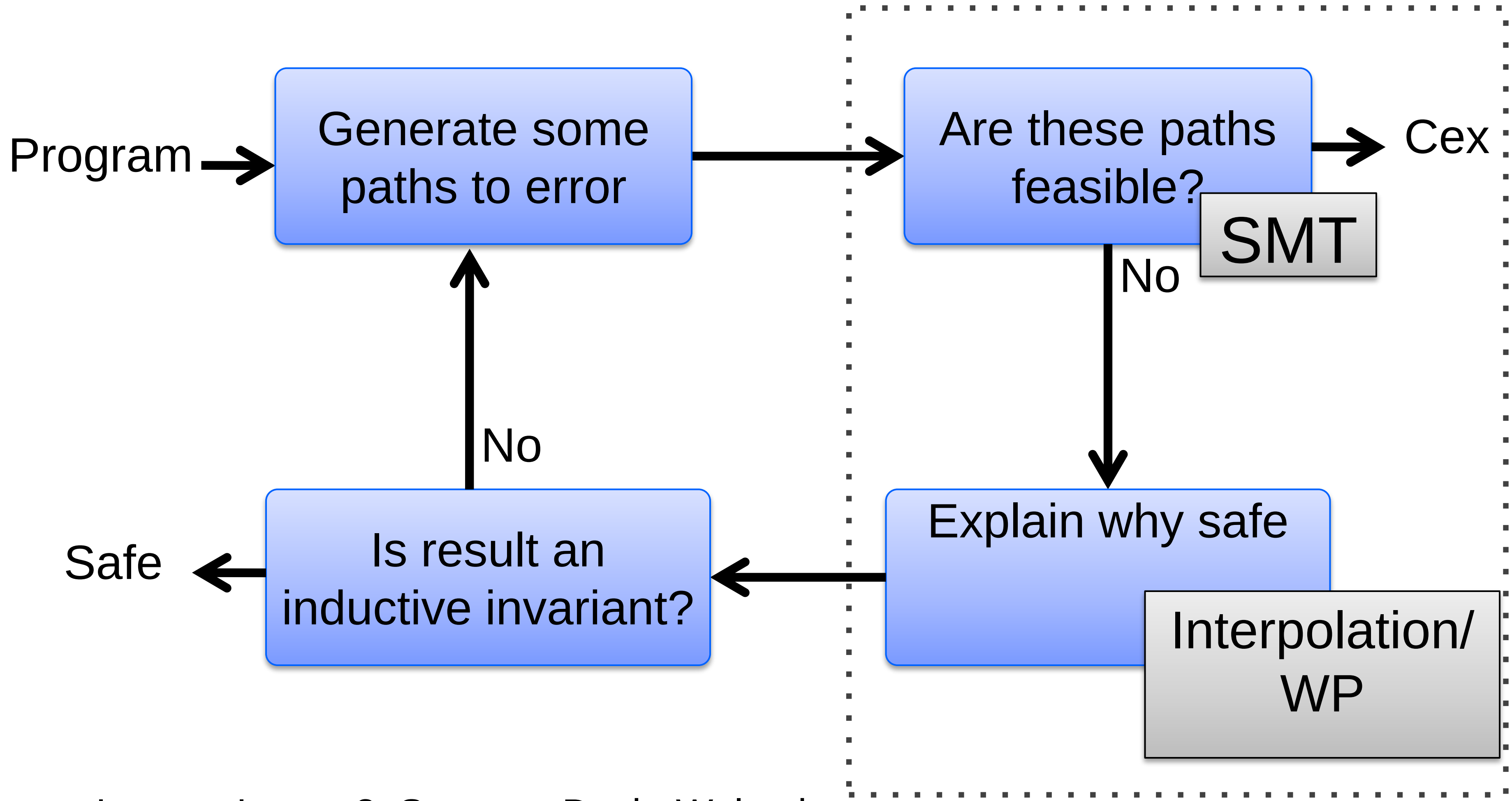


UNREACHABLE

CEGAR steps

Abstract → Translate → Check → NO ERROR

Underapproximation-driven Approach (LAWI)

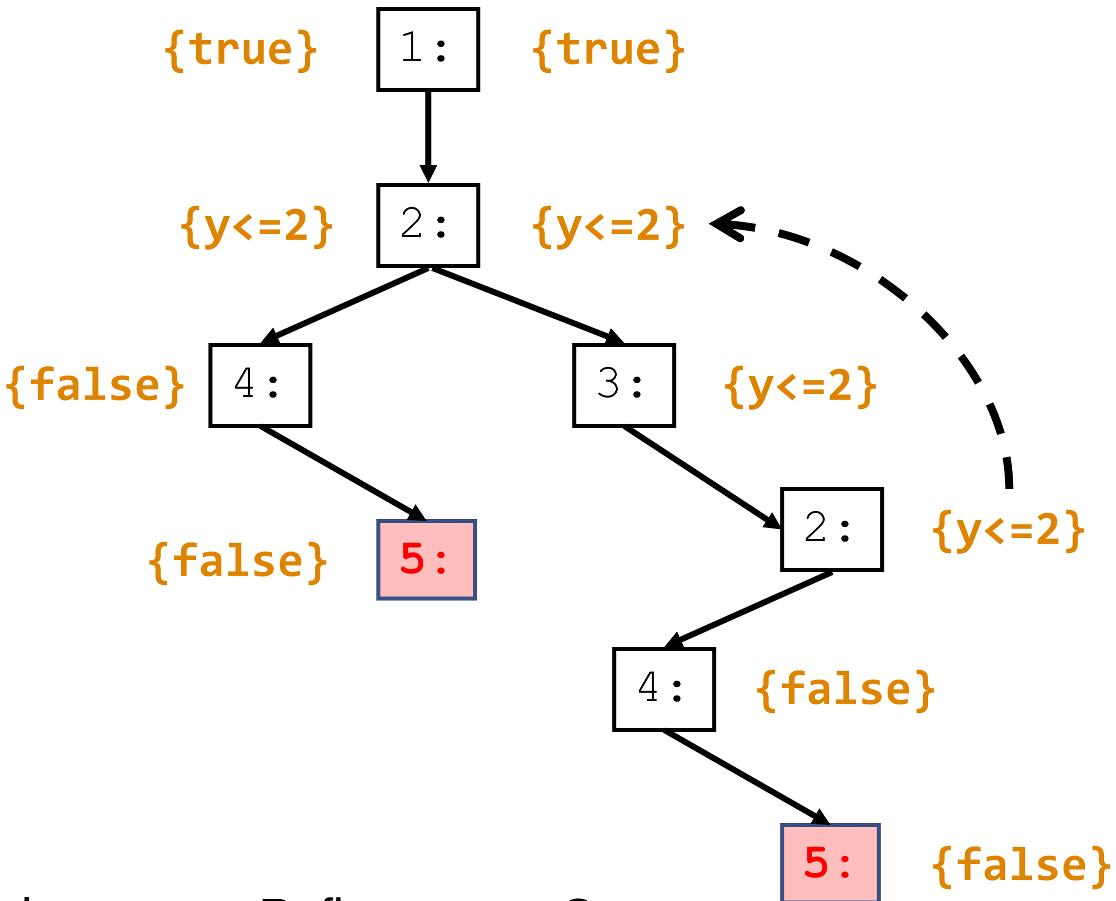


e.g., Impact, Impact2, Synergy, Dash, Wolverine

Under- Driven: Is ERROR Reachable?

Program

```
1: int x = 2;
   int y = 2;
2: while (y <= 2)
3:   y = y - 1;
4: if (x == 2)
5:     ERROR:;
6:
```



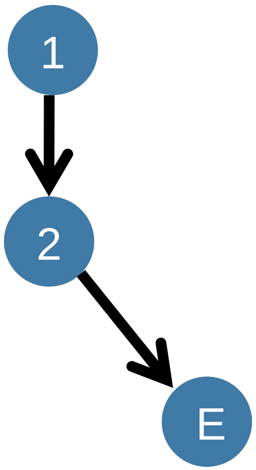
IMPACT steps

Explore → Refine → Explore → Refine → Cover

Over- Driven v.s. Under- Driven in a Nutshell

OD

UD




```
int main(){
1 …
2 while (…){
     …
  }
E: ERROR
}
```

- Unlabeled
- Pred. abs. label
- Interpolant label

Explore

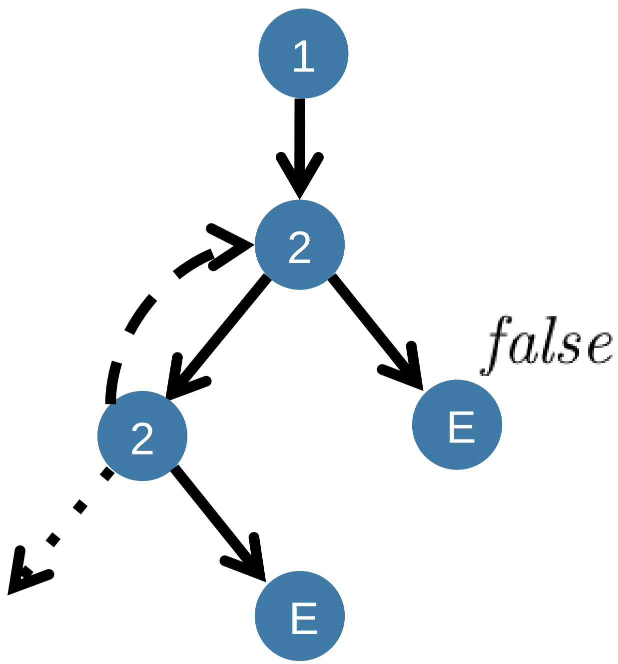
Refine

Explore

Over- Driven v.s. Under- Driven in a Nutshell

OD

UD



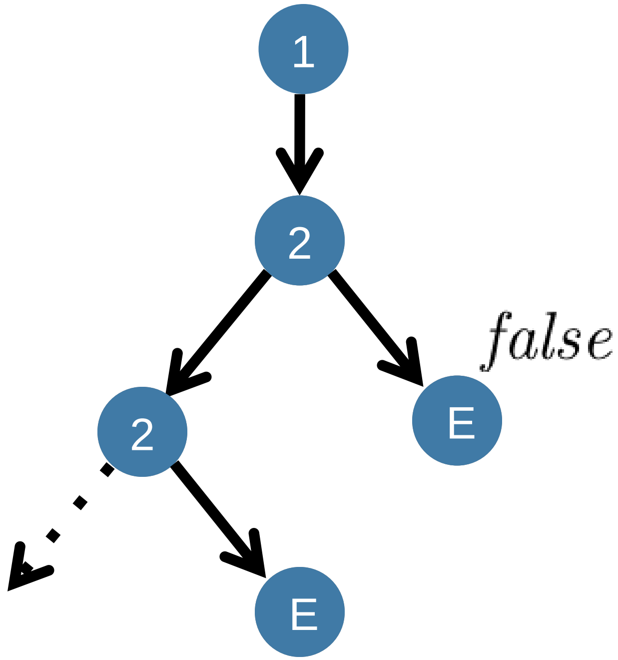

```
int main(){
1 …
2 while (…){
     …
   }
E: ERROR
}
```

- Unlabeled
- Pred. abs. label
- Interpolant label

Explore
Refine
Explore

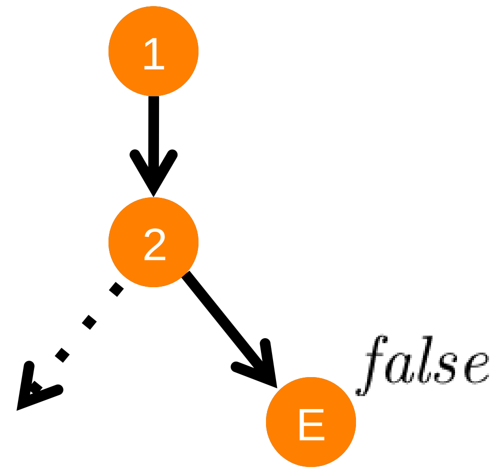
Over- Driven v.s. Under- Driven in a Nutshell

OD




```
int main(){
1 …
2 while (…){
     …
   }
E: ERROR
}
```

UD

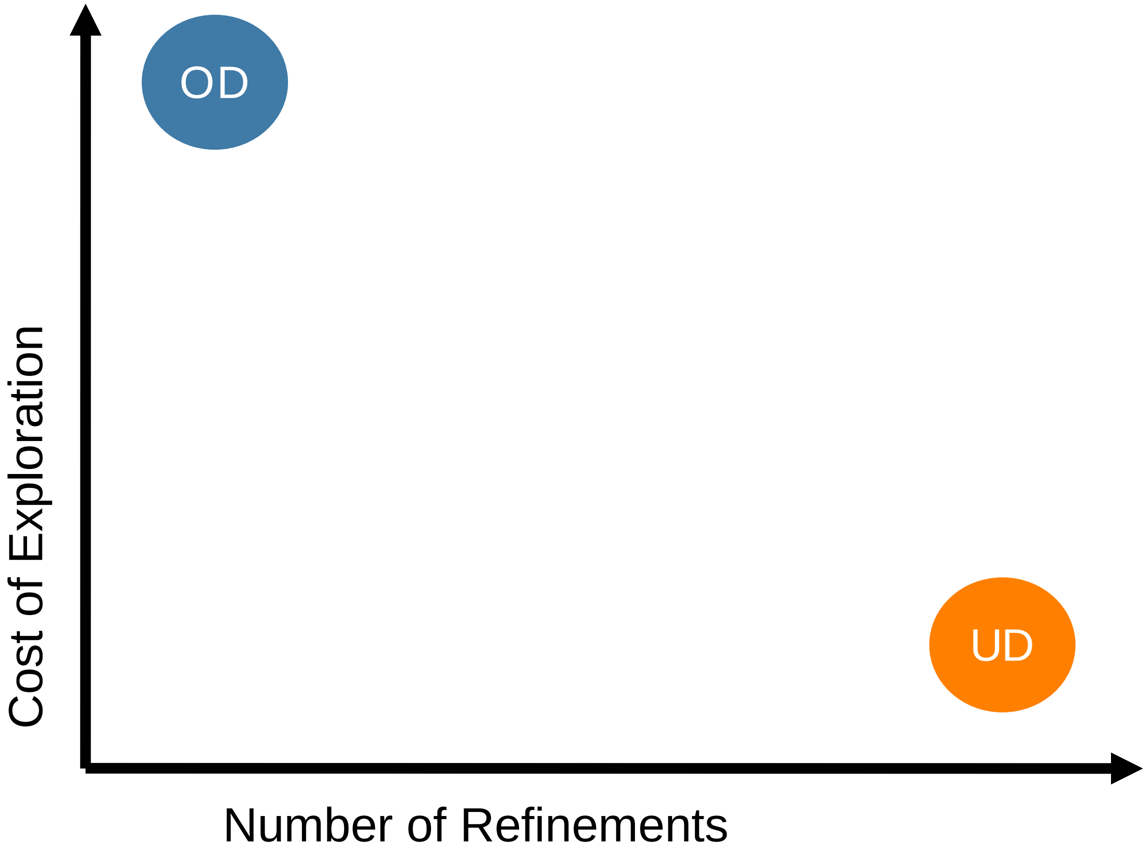


- Unlabeled
- Pred. abs. label
- Interpolant label

Explore
Refine
Explore

Explore
Refine
Explore

OD vs. UD Approaches



Our Algorithm: UFO

UD algorithm

Interpolation-based

OD algorithm

Predicate abstraction based

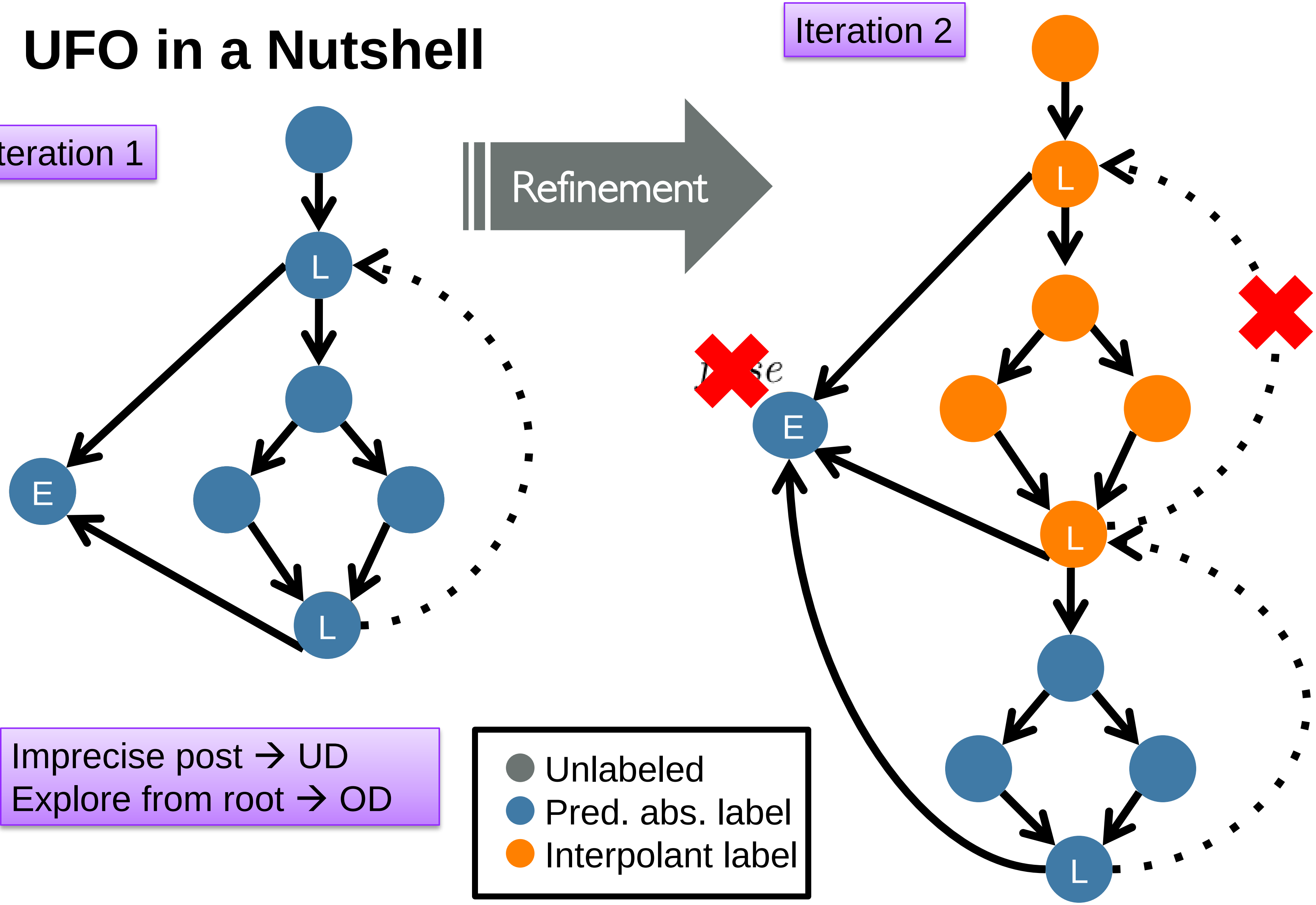
Combination of UD and OD

+

A novel interpolation-based refinement

Multiple paths checked and refined with a single SMT call

UFO in a Nutshell



Imprecise post → UD
Explore from root → OD

The UFO Algorithm

func UfoMain (Program P) :
  create node v_{en}
  $\psi(v_{\mathsf{en}}) \leftarrow \mathit{true}, \nu(v_{\mathsf{en}}) \leftarrow$
  $\mathit{marked}(v_{\mathsf{en}}) \leftarrow \mathit{true}$
  $\mathit{labels} \leftarrow \emptyset$
  while *true* **do**
    ExpandArg()
    if $\psi(v_{\mathsf{err}})$ is UNSAT **then**
      return SAFE
    $\mathit{labels} \leftarrow$ Refine()
    if $\mathit{labels} = \emptyset$ **then**
      return UNSAFE
    clear AH and FN

Explore

Refine

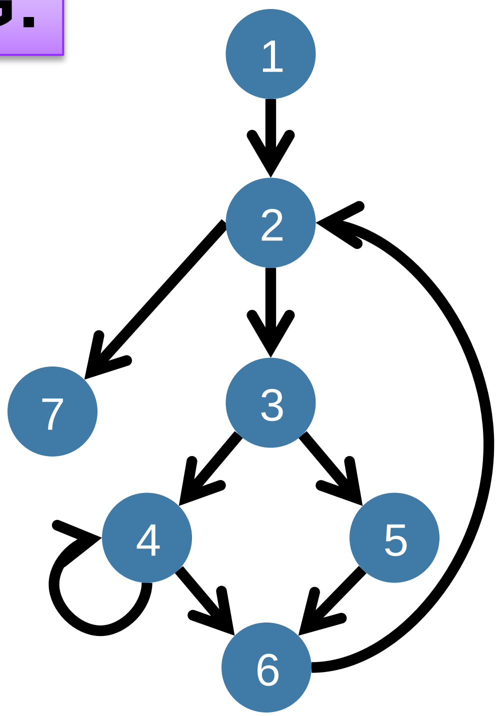
func GetFutureNode ($\ell \in \mathcal{L}$) :
  if FN(ℓ) exists **then**
    return FN(ℓ)
  create node v
  $\psi(v) \leftarrow \mathit{true}; \nu(v) \leftarrow \ell$
  FN(l) $\leftarrow v$
  return v

func ExpandNode ($v \in V$) :
  if v has children **then**
    for all $(v, w) \in E$ **do**
      FN($\nu(w)$) $\leftarrow w$
  else
    for all $(\nu(v), T, \ell) \in \Delta$ **do**
      $w \leftarrow$ GetFutureNode(ℓ)
      $E \leftarrow E \cup \{(v, w)\}; \tau(v, w) \leftarrow T$

func ExpandArg () :
  $v \leftarrow v_{\mathsf{en}}$
  while *true* **do**
    ExpandNode(v)
    if $\mathit{marked}(v)$ **then**
      $\mathit{marked}(v) \leftarrow \mathit{false}$
      $\psi(v) \leftarrow \bigvee_{(u,v) \in E} \textsc{Post}(u, v)$
      for all $(v, w) \in E$ **do** $\mathit{marked}(w) \leftarrow \mathit{true}$
    else if $\mathit{labels}(v)$ bound **then**
      $\psi(v) \leftarrow \mathit{labels}(v)$
      for all $\{(v, w) \in E \mid \mathit{labels}(w)$ unbound$\}$ **do**
        $\mathit{marked}(w) \leftarrow \mathit{true}$
    if $v = v_{\mathsf{err}}$ **then** break
    if $\nu(v)$ is head of a component **then**
      if $\psi(v) \Rightarrow \bigvee_{u \in \mathsf{AH}(\nu(v))} \psi(u)$ **then**
        erase AH($\nu(v)$) and FN($\nu(v)$)
        $l \leftarrow$ WtoExit($\nu(v)$)
        $v \leftarrow$ FN(l); erase FN(l)
        for all $\{(v, w) \in E \mid \nexists u \neq v \cdot (u, w) \in E\}$ **do**
          erase FN($\nu(w)$)
        continue
      add v to AH($\nu(v)$)
    $l \leftarrow$ WtoNext($\nu(v)$)
    $v \leftarrow$ FN(l); erase FN(l)

Weak Topological Ordering

DAG:



Definition (WTO):

A weak topological order (WTO) of a DAG G = (V, E) is a well-parenthesised total-order $\preceq$ of V without two consecutive '(' such that for every edge (u, v) $\in$ E:

$$(u \prec v \land v \notin \omega(u)) \lor (u \preceq u \land v \in \omega(u))$$

Elements between two matching paren. are called *components*

First element of a component is called *head*

ω(u) is the set of heads of components containing u

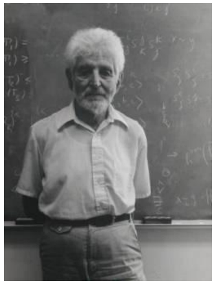
WTO:

(1 (2 3 (4) 5 6) 7)

Refinement

DAG Interpolation

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

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

Theorem (McMillan 2003)

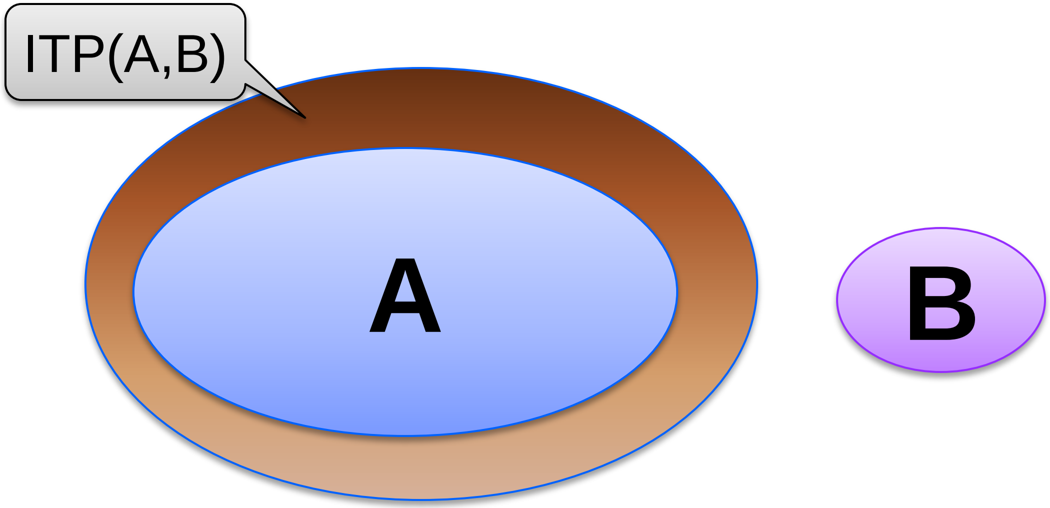
A Craig interpolant ITP(A, B) can be effectively constructed from a resolution proof of unsatisfiability of $A \wedge B$

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

Craig Interpolation in Model Checking

Over-Approximating Reachable States

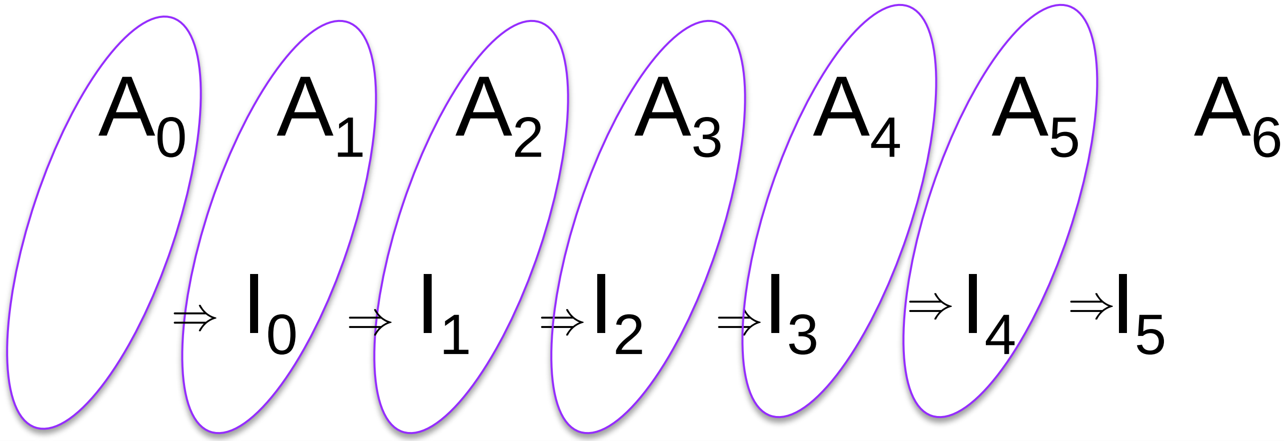
- Let $\mathbf{R^i}$ be the i^{th} step of the transition relation
- Assume: $\mathbf{= Init \wedge R^0 \wedge \ldots \wedge R^n \wedge Bad}$ is UNSAT (no **Bad** in n steps)
- Let $\mathbf{A = Init \wedge R^0 \wedge \ldots \wedge R^n}$ and $\mathbf{B = Bad}$
- **ITP (A, B)** (if exists) is an over-approx of states reachable in n-steps that does not contain any Bad states



Interpolation Sequence, a.k.a. Path Interpolants

Given a sequence of formulas $\mathbf{A = \{A_i\}_{i=0}^{n}}$, an ***interpolation sequence*** $\mathbf{ItpSeq(A) = \{I_1, \ldots, I_{n-1}\}}$ is a sequence of formulas such that

- $\mathbf{I_k}$ is an $\mathbf{ITP\ (A_0} \land \ldots \land \mathbf{A_{k-1},\quad A_k} \land \ldots \land \mathbf{A_n)}$, and
- $\forall\ \mathbf{k<n\ .\ I_k} \land \mathbf{A_{k_{+1}}} \Rightarrow \mathbf{I_{k+1}}$



If A_i is a transition relation of step i, then the interpolation sequence is a proof why a program trace is safe.

DAG Interpolants: Solving the Refinement Prob.

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

$I_2 = ITP\ (\pi_1,\quad \pi_8)$

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

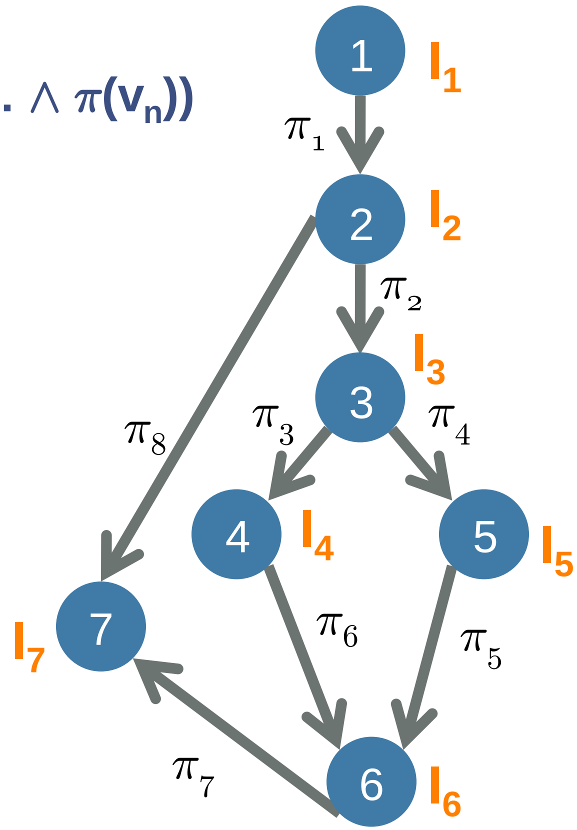
…

$(I_1 \wedge \pi_1) \Rightarrow I_2$

$(I_2 \wedge \pi_8) \Rightarrow I_7$

$(I_2 \wedge \pi_2) \Rightarrow I_3$

…



DAG Interpolation Algorithm

Reduce DAG Interpolation to Sequence Interpolation!

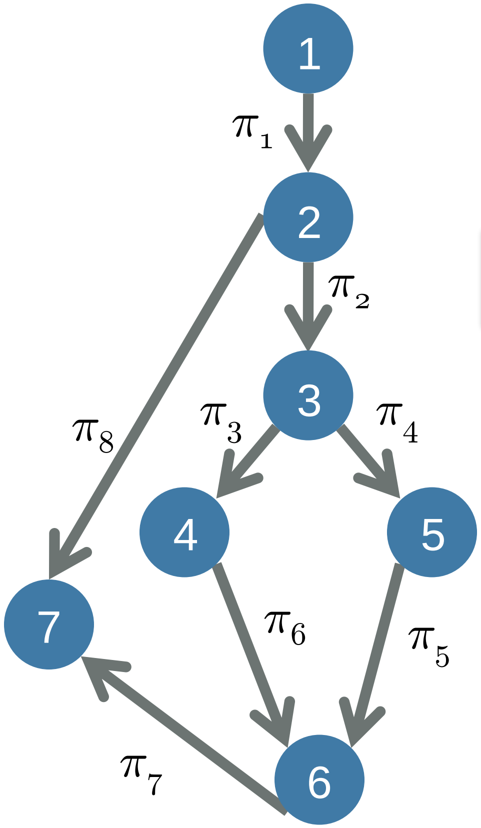
```
DagItp ((V, E), π)
{
   (A_0, …, A_n) = Encode(V, E, π)

   (I_1, …, I_n-1) = SeqItp(A_0, …, A_n)

   for i in [1, n-1] do J_i = Clean(I_i)

   return (J_1, …, J_n-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

DagItp: Encode



Encode

A_1: v_1

$v_1 \Rightarrow v_2 \wedge \pi_1$

A_2: $v_2 \Rightarrow (v_3 \wedge \pi_2) \vee (v_7 \wedge \pi_8)$

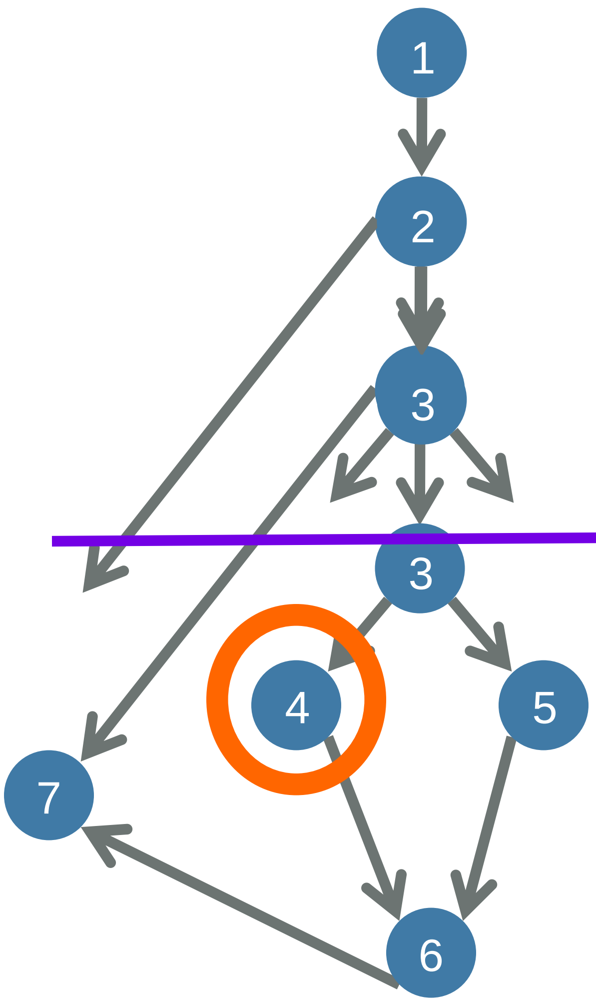
A_3: $v_3 \Rightarrow (v_4 \wedge \pi_3) \vee (v_5 \wedge \pi_4)$

A_4: $v_4 \Rightarrow v_6 \wedge \pi_6$

A_5: $v_5 \Rightarrow v_6 \wedge \pi_5$

A_6: $v_6 \Rightarrow v_7 \wedge \pi_7$

DagItp: Sequence Interpolate



A_1: v_1
$v_1 \Rightarrow v_2 \wedge \pi_1$

A_2: $v_2 \Rightarrow (v_3 \wedge \pi_2) \vee (v_7 \wedge \pi_8)$

A_3: $v_3 \Rightarrow (v_4 \wedge \pi_3) \vee (v_5 \wedge \pi_4)$

I_4

A_4: $v_4 \Rightarrow v_6 \wedge \pi_6$

A_5: $v_5 \Rightarrow v_6 \wedge \pi_5$

A_6: $v_6 \Rightarrow v_7 \wedge \pi_7$

DagItp: Clean

```
Clean(Iᵢ) =
```

$$\forall\{x \mid x \in \mathrm{var}(I_i) \wedge \neg \mathrm{inScope}(x, v_i)\} \cdot I[v_i \leftarrow \top][v_j \leftarrow \bot \mid j \neq i]$$

The universal quantification is a major bottleneck in practice. We use many heuristics to limit its application. In the worst case, we use quantifier elimination by Loos and Weispfenning as implemented in Z3.

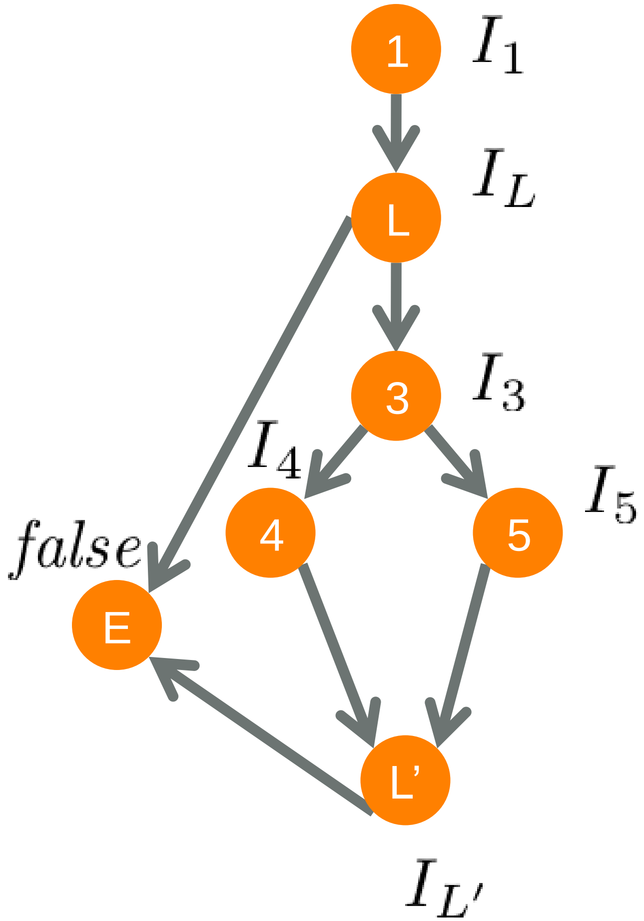
We are exploring several approaches that do not require quantifier elimination at all.

UFO Refinement

1. Construct DAG of current unfolding
2. Use DagItp to find new labels

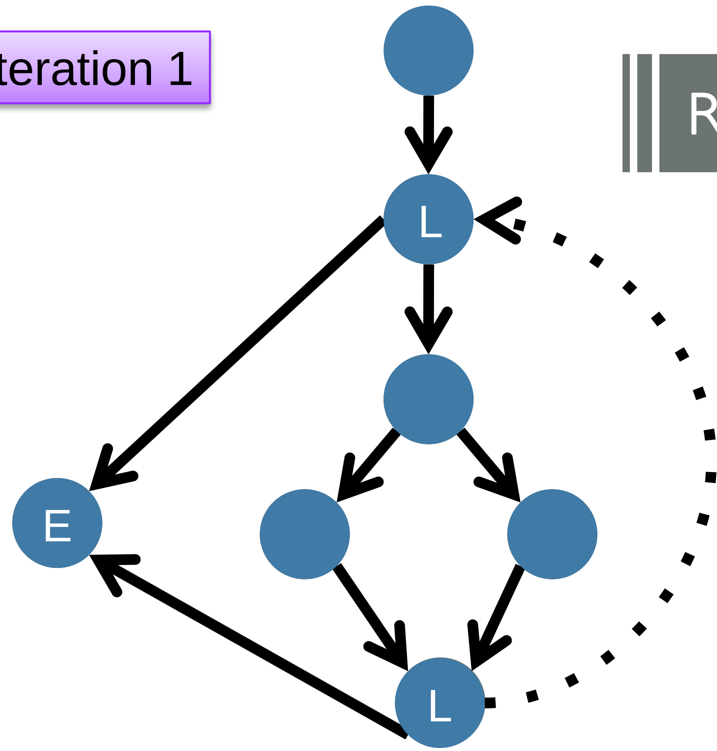
Refinement is done with a ***single*** SMT call

Cleaning the labels with quantifier elimination is a major bottleneck

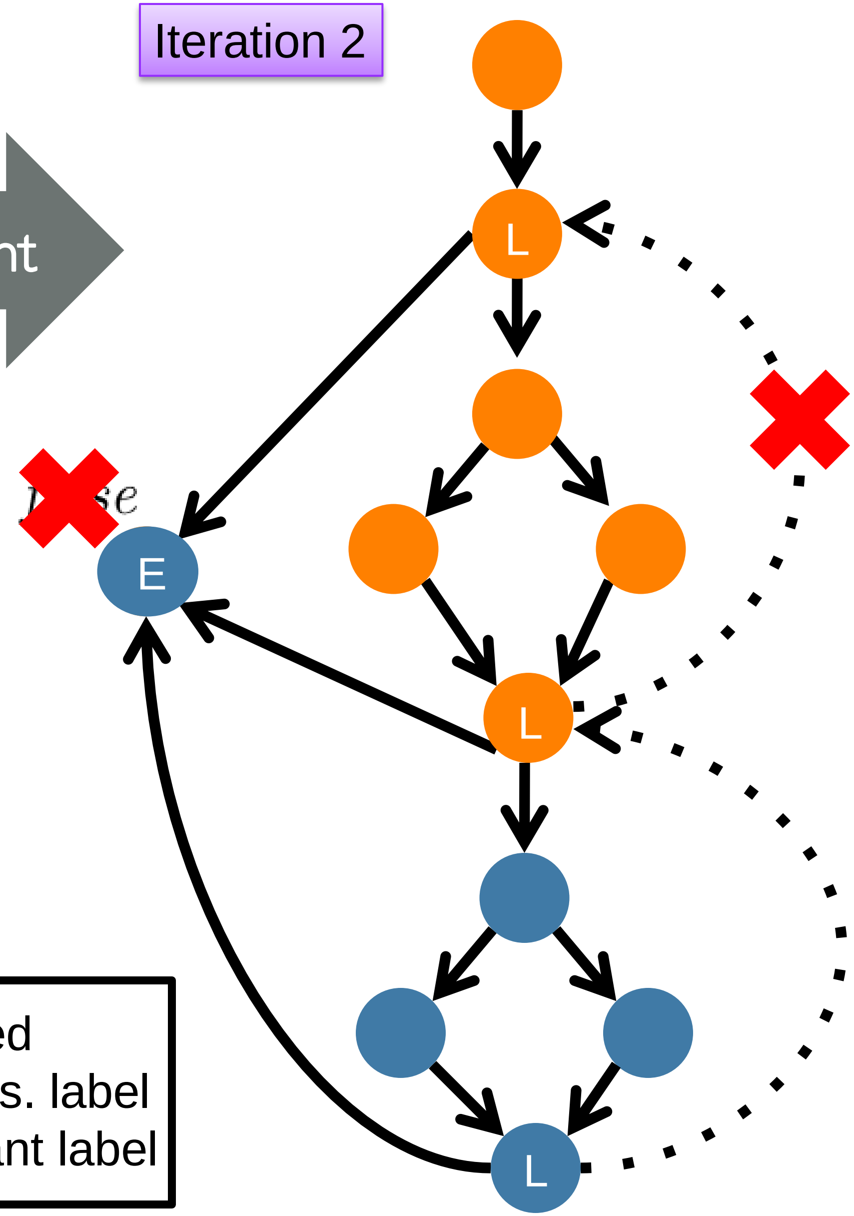


UFO in a Nutshell

Iteration 1



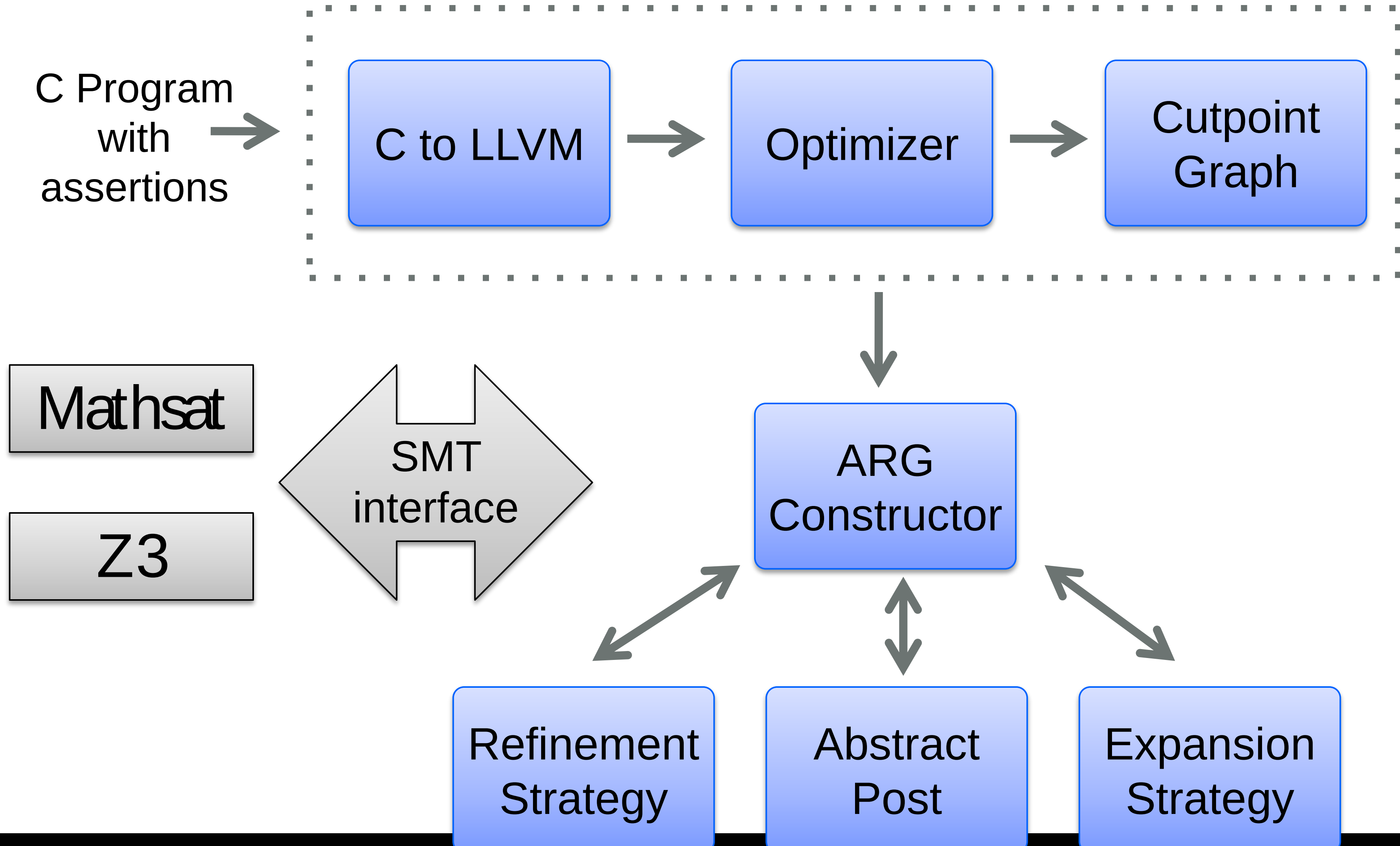
Iteration 2



Imprecise post → UD
Explore from root → OD

- Unlabeled
- Pred. abs. label
- Interpolant label

UFO as a Framework: The Architecture



Recent Related Work

Intra-procedural

Impact [McMillan 06]

- Original lazy abstraction with interpolants

Impact2 [McMillan 10]

- Targets testing/exploration

Wolverine [Weissenbacher 11]

- Bit-level interpolants

Ultimate [Ermis et al. 12]

- Impact with Large Block Encoding for Refinement

Inter-procedural

Whale [Our work 12]

- Inter-procedural verification with interpolants

FunFrog [Sery et al. 11]

- Function summarization using interpolants

More Recent Related Work

Software Model Checking via IC3 [Cimatti and Griggio, 12]

- IMPACT with IC3-style generalization

Duality [McMillan and Rybalchenko, 12]

- Interpolation-based algorithm for Relational Post-Fixed Point

Generalized Property Directed Reachability [Hoder and Bjorner, 12]

- Relational Post-Fixed Point in Z3

Solving Recursion-Free Horn Clauses over LI+UIF [Gupta et al. 11]

- solving DAG interpolation and beyond…

Alternate and Learn [Sinha et al. 12]

- strategies for inlining/instantiating procedures in bounded verification

Software Verification Competition (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 Results

UFO won gold in 4 categories

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

Performed much better than mature Predicate Abstraction-based tools

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

Secret Sauce

UFO Front-End

Vinta: combining UFO with Abstract Interpretation [SAS ‘2012]

Boxes Abstract Domain [SAS ‘2010 w/ Sagar Chaki]

DAG Interpolation [TACAS ‘2012 and SAS ‘2012]

Run many variants in parallel

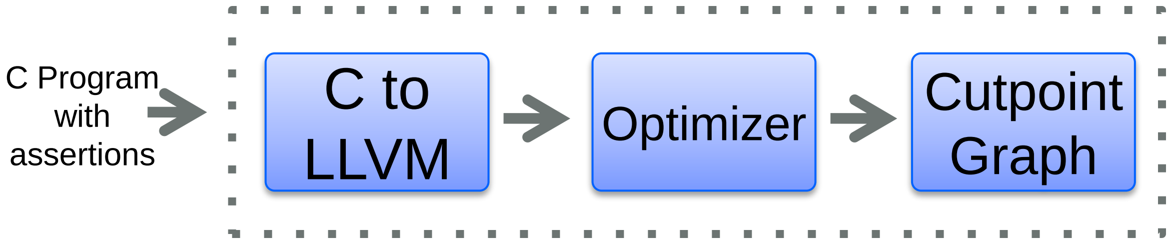
UFO Front End

In principle simple, but in practice very messy

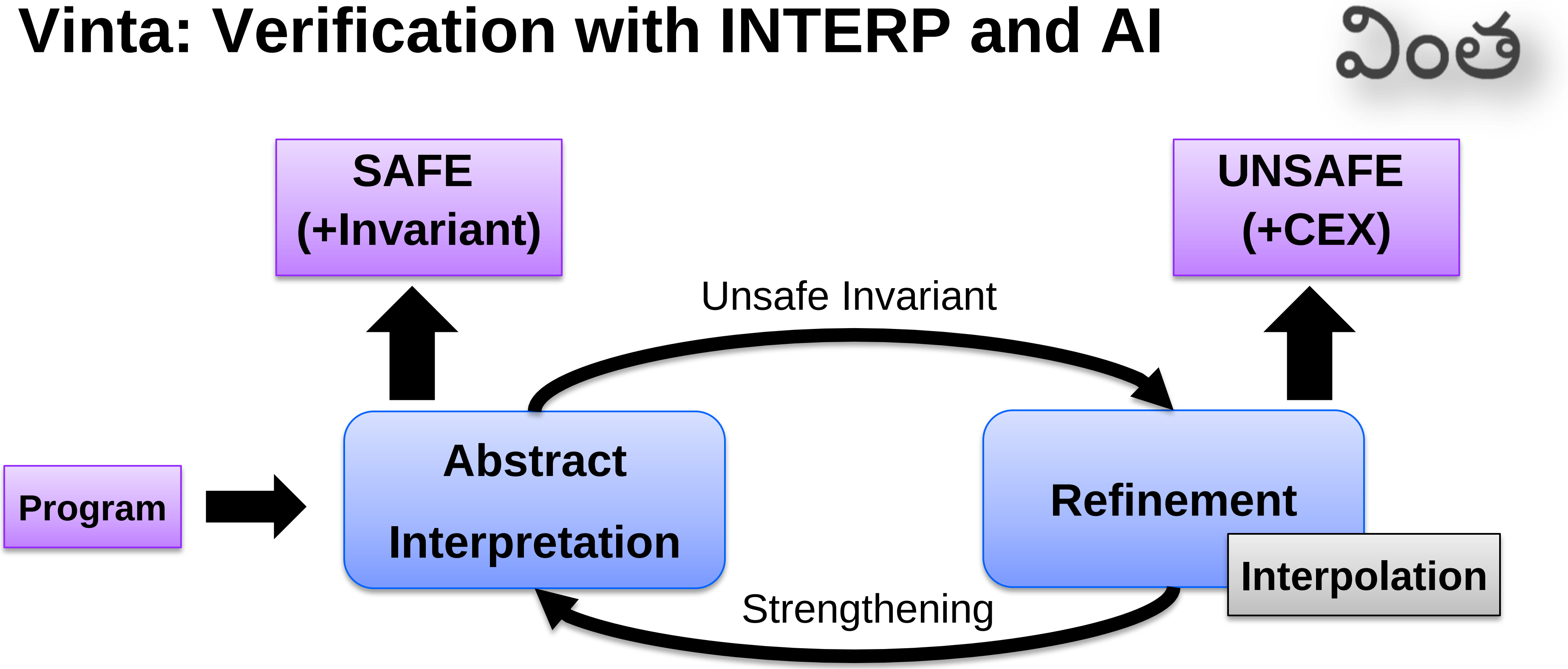
- CIL passes to normalize the code (library functions, uninitialized vars, etc.)
- `llvm-gcc` (without optimization) to compile C to LLVM bitcode
- `llvm opt` with many standard, custom, and modified optimizations
  - lower pointers, structures, unions, arrays, etc. to registers
  - constant propagation + many local optimizations
  - difficult to preserve *intended* 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)



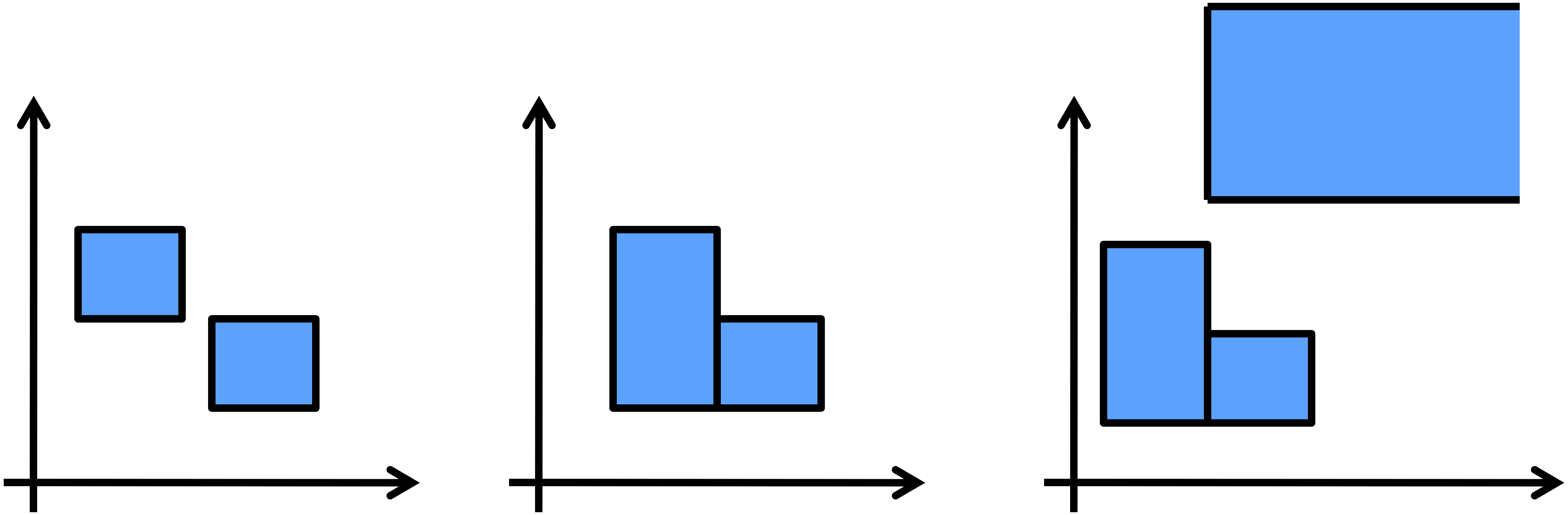
Vinta: Verification with INTERP and AI



- 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

Boxes Abstract Domain: Semantic View *



Boxes are “finite union of box values”

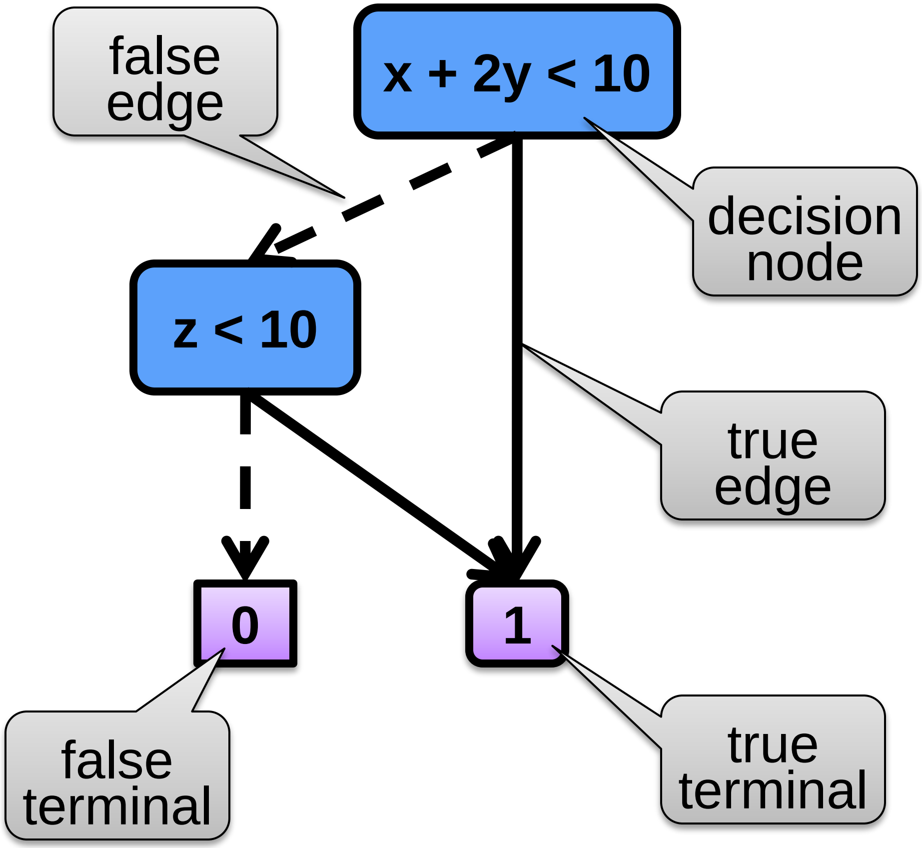
(alternatively)

Boxes are “Boolean formulas over interval constraints”

*joint work w/ Sagar Chaki

Linear Decision Diagrams in a Nutshell*

Linear Decision Diagram



Linear Arithmetic Formula

$(x + 2y < 10)$ **OR**
$(x + 2y \geq 10$ **AND** $z < 10)$

Compact Representation

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

Operations

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

*joint work w/ Sagar Chaki and Ofer Strichman

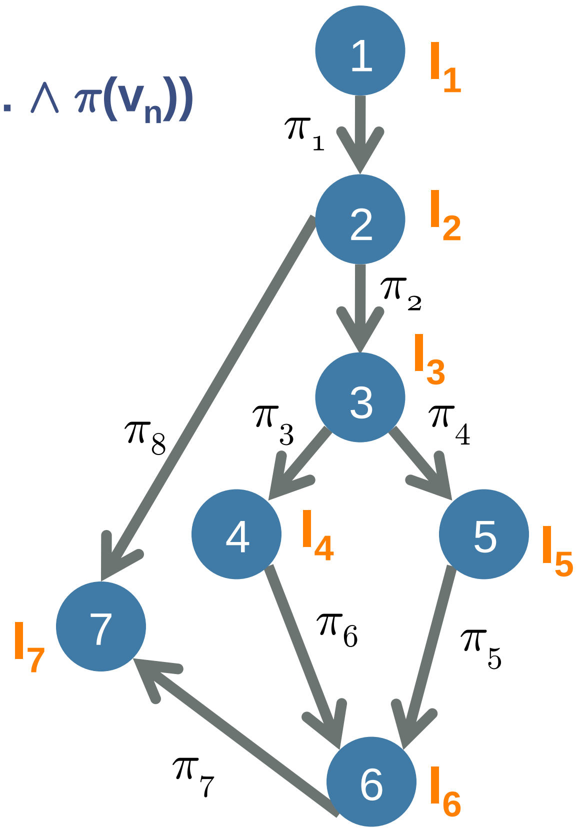
DAG Interpolants: Solving the Refinement Prob.

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

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

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

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



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

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

In The Box

Image courtesy of Aws Albarghouthi

Nutrition Facts

Serving Size: 1 sec spray (.5ml)
Serving Per Container:

Amount Per Serving	
Calories: 4	Cal from Fat: 4
	% Daily Value*
Abstract Interpretation	30%
Predicate Abstraction	
Box/Boxes Domains	
BMC/VCC Encoding	10%
CEGAR Loop	30%
Interpolation	30%
DAG	
RDI	

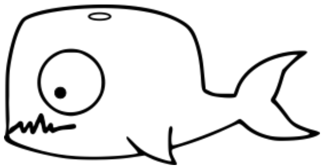
Vitamin A 4% • Vitamin C 4%
Calcium 10% • Iron 4%

*Percent Values are based on a 2,000 calorie diet. Your Daily Values may be higher or lower depending on your calorie needs:

	Calories	2,000	2,500
Total Fat	Less Than	65g	80g
Sat Fat	Less Than	20g	25g
Cholesterol	Less Than	300mg	300mg
Sodium	Less Than	2,400mg	2,400mg
		300g	375g
Dietary Fiber		25g	30g

Calories Per Gram
Fat 9 · Carbohydrate 4 · Protein 4

UFO 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

Thank You!

http://bitbucket.org/arieg/ufo

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