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
[TACAS12] From Under-approximations to Over-approximations and Back
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)

Program → Compute invariant using abstract post

Is safe?
→ Safe

Refine post operator

Is cex feasible?
→ Cex

SMT

Interpolation or WP

e.g., BLAST, SLAM, CPAChecker, YaSM, SATAbs, etc.
Is ERROR Reachable?

Program

Abstraction

Over-Approximation

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

1: ;
2: while (*)
3:     ;
4: if (*)
5:     ERROR;:
6: 

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

CEGAR steps

Abstract ➔ Translate ➔ Check ➔ NO ERROR
Underapproximation-driven Approach (LAWI)

Program → Generate some paths to error → Are these paths feasible? → Cex

Safe → Is result an inductive invariant? → Explain why safe → Interpolation/WP

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

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

Explore
Refine
Explore

Unlabeled
Pred. abs. label
Interpolant label
Over- Driven v.s. Under- Driven in a Nutshell

Explore
Refine
Explore

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

- OD (Over-Driven)
- UD (Under-Driven)
- Unlabeled
- Pred. abs. label
- Interpolant label

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

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

Explore
Refine
Explore
OD vs. UD Approaches

OD

UD

Cost of Exploration

Number of Refinements
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

[TACAS’12]
UFO in a Nutshell

Iteration 1

Imprecise post → UD
Explore from root → OD

Unlabeled
Pred. abs. label
Interpolant label

Iteration 2

Refinement

Imprecise post → UD
Explore from root → OD

Unlabeled
Pred. abs. label
Interpolant label
The UFO Algorithm

1: func UfoMain (Program P) :
2:   create node \( v_e \)
3:   \( \psi(v_e) \leftarrow true, \nu(v_e) \leftarrow \) \( \in \) \( \Delta \)
4:   marked\( (v_e) \leftarrow true \)
5:   labels \( \leftarrow \emptyset \)
6:   while true do
7:     EXPANDARG()
8:     if \( \psi(v_e) \) is UNSAT then
9:       return SAFE
10:    labels \( \leftarrow \) REFINE()
11:   if labels = \( \emptyset \) then
12:      return UNSAFE
13:   clear AH and FN
14: func GetFutureNode (\( \ell \in L \) ) :
15:   if FN(\( \ell \) ) exists then
16:     return FN(\( \ell \) )
17:   create node \( v \)
18:   \( \psi(v) \leftarrow true, \nu(v) \leftarrow \ell \)
19:   FN(\( \ell \) ) \( \leftarrow v \)
20:   return \( v \)
21: func EXPANDNODE (\( v \in V \) ) :
22:   if \( v \) has children then
23:     for all (\( v, w \) ) \( \in E \) do
24:       FN(\( \nu(w) \) ) \( \leftarrow w \)
25:   else
26:     for all (\( \nu(v), T, \ell \) ) \( \in \Delta \) do
27:       \( w \leftarrow GETFUTURENODE(\ell) \)
28:       \( E \leftarrow E \cup \{(v, w)\}; \tau(v, w) \leftarrow T \)
29: func EXPANDARG () :
30:   \( v \leftarrow v_e \)
31:   while true do
32:     EXPANDNODE(\( v \) )
33:     if marked(\( v \) ) then
34:       marked(\( v \) ) \( \leftarrow false \)
35:       \( \psi(v) \leftarrow \bigvee_{(u, v) \in E} Post(u, v) \)
36:     for all (\( v, w \) ) \( \in E \) do marked(\( w \) ) \( \leftarrow true \)
37:     else if labels(\( v \) ) bound then
38:       \( \psi(v) \leftarrow labels(\( v \) ) \)
39:     for all \( \{(v, w) \in E \mid labels(\( w \) ) unbound\} \) do
40:       marked(\( w \) ) \( \leftarrow true \)
41:     if \( v = v_e \) then break
42:     if \( \nu(v) \) is head of a component then
43:       if \( \psi(v) \Rightarrow \bigvee_{u \in AH(\nu(v))} \psi(u) \) then
44:         erase AH(\( \nu(v) \) ) and FN(\( \nu(v) \) )
45:         \( l \leftarrow WTOEXIT(\nu(v)) \)
46:         \( v \leftarrow FN(l) \); erase FN(\( l \) )
47:     for all \( \{(v, w) \in E \mid \exists u \neq v \cdot (u, w) \in E\} \) do
48:       erase FN(\( w \) )
49:     continue
50:   add \( v \) to AH(\( \nu(v) \) )
51:   \( l \leftarrow WTONEXT(\nu(v)) \)
52:   \( v \leftarrow FN(l) \); erase FN(\( l \) )
Weak Topological Ordering

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 \((u, v) \in E\):

\[(u \prec v \land v \not\in \omega(u)) \lor (u \preceq u \land v \in \omega(u))\]

Elements between two matching paren. are called components

First element of a component is called head

\( \omega(u) \) 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 $\text{ITP}(A, B)$, such that

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

**Theorem** (McMillan 2003)
A Craig interpolant $\text{ITP}(A, B)$ can be effectively constructed from a resolution proof of unsatisfiability of $A \land B$

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 $R^i$ be the $i$th step of the transition relation
• Assume: $\text{Init} \land R^0 \land \ldots \land R^n \land \text{Bad}$ is UNSAT (no Bad in $n$ steps)
• Let $A = \text{Init} \land R^0 \land \ldots \land R^n$ and $B = \text{Bad}$
• 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 $A = \{A_i\}_{i=0}^n$, an interpolation sequence $\text{ItpSeq}(A) = \{I_1, \ldots, I_{n-1}\}$ is a sequence of formulas such that

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

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 $\pi : E \rightarrow \text{Expr}$. A DAG Interpolant (if it exists) is a labeling $I : V \rightarrow \text{Expr}$ such that

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

$I_2 = \text{ITP} (\pi_1, \pi_8)$

$I_2 = \text{ITP} (\pi_1, \pi_2 \land \pi_3 \land \pi_6 \land \pi_7)$

$\ldots$

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

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

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

$\ldots$
DAG Interpolation Algorithm

Reduce DAG Interpolation to Sequence Interpolation!

\[
\text{DagItp} \left((V, E), \pi\right) = \\
\{(A_0, \ldots, A_n) = \text{Encode}(V, E, \pi)\} \\
\{(I_1, \ldots, I_{n-1}) = \text{SeqItp}(A_0, \ldots, A_n)\} \\
\text{for } i \text{ in } [1, n-1] \text{ do } J_i = \text{Clean}(I_i) \\
\text{return } (J_1, \ldots, J_{n-1})
\]

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

\[
\begin{align*}
A_1 & : V_1 \\
& : V_1 \implies V_2 \land \pi_1 \\
A_2 & : V_2 \implies (V_3 \land \pi_2) \lor (V_7 \land \pi_8) \\
A_3 & : V_3 \implies (V_4 \land \pi_3) \lor (V_5 \land \pi_4) \\
A_4 & : V_4 \implies V_6 \land \pi_6 \\
A_5 & : V_5 \implies V_6 \land \pi_5 \\
A_6 & : V_6 \implies V_7 \land \pi_7
\end{align*}
\]
DagItp: Sequence Interpolate

\[ A_1 \Rightarrow v_1 \land \pi_1 \]
\[ A_2 \Rightarrow (v_3 \land \pi_2) \lor (v_7 \land \pi_8) \]
\[ A_3 \Rightarrow (v_4 \land \pi_3) \lor (v_5 \land \pi_4) \]
\[ A_4 \Rightarrow v_6 \land \pi_6 \]
\[ A_5 \Rightarrow v_6 \land \pi_5 \]
\[ A_6 \Rightarrow v_7 \land \pi_7 \]
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

Imprecise post → UD
Explore from root → OD

Unlabeled
Pred. abs. label
Interpolant label
UFO as a Framework: The Architecture

C Program with assertions

C to LLVM

Optimizer

Cutpoint Graph

ARG Constructor

Refinement Strategy

Abstract Post

Expansion Strategy

Mathsat

Z3

SMT interface

SMT interface
## 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

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

http://sv-comp.sosy-lab.org/2013/
SV-COMP 2013: Scoring Scheme

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

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

- **false edge**
- **true edge**
- **false terminal**
- **true terminal**
- **decision node**

```
x + 2y < 10
```

```
z < 10
```

Linear Arithmetic Formula

- \((x + 2y < 10) \text{ OR } (x + 2y \geq 10 \text{ AND } z < 10)\)

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 $\pi: E \rightarrow \text{Expr}$. A
\textit{DAG Interpolant} (if it exists) is a labeling $I: V \rightarrow \text{Expr}$ such that

- for any path $v_0, \ldots, v_n$, and $0 < k < n$,
  $I(v_k) = \text{ITP} (\pi(v_0) \land \ldots \land \pi(v_{k-1}), \pi(v_k) \land \ldots \land \pi(v_n))$

- $\forall (u, v) \in E . (I(u) \land \pi(u, v)) \Rightarrow I(v)$

\begin{align*}
I_2 &= \text{ITP} (\pi_1, \pi_8) \\
I_2 &= \text{ITP} (\pi_1, \pi_2 \land \pi_3 \land \pi_6 \land \pi_7) \\
\vdots \\
(I_1 \land \pi_1) &\Rightarrow I_2 \\
(I_2 \land \pi_8) &\Rightarrow I_7 \\
(I_2 \land \pi_2) &\Rightarrow I_3 \\
\vdots
\end{align*}
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

References:
[SAS12] Craig Interpretation
[TACAS12] From Under-approximations to Over-approximations and Back

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

Image courtesy of Aws Al barghouthi
UFO Family

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

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

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