Battling the Infinite: Proving Safety of Programs

Aws Albarghouthi / UToronto
joint work with
Arie Gurfinkel (CMU), Marsha Chechik, Yi Li (UToronto)

March 1st 2013
@ TU München
Aw, Snap!

Something went wrong while displaying this webpage. To continue, reload or go to another page.

If you’re seeing this frequently, try these suggestions.
High-frequency trading and the $440m mistake

By Tim Harford
BBC Radio 4, More or Less
Now: 500+ million devices activated
700K+ apps
Apple redraws maps after Australian drivers led astray in the bush
Mildura police issue warning after motorists lose way in scorching temperatures because town misplaced on Apple Maps

Charles Arthur
guardian.co.uk, Monday 10 December 2012 08.02 GMT
Jump to comments (157)
GPS failure leaves Belgian woman in Zagreb two days later

A 67-year-old Belgian woman set out to drive 38 miles to Brussels under the guidance of her GPS navigation system but arrived in Zagreb two days and 901 miles later.
Automated Verification

Source code → PUSH
Automated Verification

Source code ➔ PUSH

Correct ✓

Buggy! ✗
Automated Verification

Source code → PUSH

Safety properties
“Something **bad** will never happen”

Correct ✓
Buggy! ✗
Turing, 1936
Turing, 1936

“undecidable”
Turing, 1936

“undecidable”
Turing, 1949
How can one check a routine in the sense of making sure that it is right?
Turing, 1949

How can one check a routine in the sense of making sure that it is right? The programmer should make a number of definite assertions which can be checked individually, and from which the correctness of the whole programme easily follows.
How can one check a routine in the sense of making sure that it is right? The programmer should make a number of definite assertions which can be checked individually, and from which the correctness of the whole programme easily follows.
How can one check a routine in the sense of making sure that it is right? The programmer should make a number of definite assertions which can be checked individually, and from which the correctness of the whole programme easily follows.
How can one check a routine in the sense of making sure that it is right? The programmer should make a number of definite assertions which can be checked individually, and from which the correctness of the whole programme easily follows.
How can one check a routine in the sense of making sure that it is right? The programmer should make a number of definite assertions which can be checked individually, and from which the correctness of the whole programme easily follows.
How to automatically infer safe invariants?
Today’s Menu

Following Turing’s path: The UFO approach

UFO in practice

Future goals
Example

1: \( x := 10; \)

2: \( \text{while} \ (\text{in}())\{ \)
3: \( \quad x := x - 2; \)
   \}

4: \( \text{assert} \ (x \neq 9); \)
Example

1: \( x := 10; \)

2: while (in()){
   3:     x := x - 2;
   }

4: assert (x != 9);
Example

1: x := 10;

2: while (in()){
3:   x := x - 2;
}

4: assert (x != 9);
Example

1: x := 10;

2: while (in()){
3:   x := x - 2;
   }  

4: assert (x != 9);
1: \( x := 10; \)

2: \[ \text{while (in())} \{
\]
3: \( x := x - 2; \)
\]
4: \( \text{assert (} x \neq 9); \)
Example

1: x := 10;

2: while (in()){
3:     x := x - 2;
 }

4: assert (x != 9);
Example

1: \( x := 10; \)

2: while (in()) {
   3: \( x := x - 2; \)
     }

4: assert (x \(!=\) 9);
Example

1: \( x := 10; \)

2: \( \textbf{while} \ (\text{in}()) \{ \)
3: \[ \begin{align*}
   & x := x - 2; \\
   & \}
\]

4: \( \textbf{assert} \ (x \neq 9); \)
Example

1: \( x := 10; \)

2: while (in()){
   3: \( x := x - 2; \)
}

4: assert \( (x \neq 9); \)
Example

1: \( x := 10; \)

2: while (in()){
3: \( x := x - 2; \)
} 

4: assert (x != 9);
Example

1: \( x := 10; \)

2: while (in()){
   3: \( x := x - 2; \)
   }

4: assert (x != 9);
The UFO Approach

Verifier

Program → Verifier → Partial Program → Bounded Verifier

Property → Annotation → Correct ✓

Bounded Verifier

Buggy! ✗
Bounded Programs

1. $x := 10$
2. $in()$
3. $x := x - 2$
4. $!in()$

$B$

$x == 9$
Bounded Programs

Bounded Programs

1
2
3
4

B

x := 10

in()

x := x - 2

!in()

x == 9

B

1
2
3
4

B

x := 10

in()

x := x - 2

!in()

x == 9

!in()
Bounded Programs

1. \( x := 10 \)
2. \( x := x - 2 \)
3. \( x := x - 2 \)
4. \( x == 9 \)

Diagram:

- Node 1: \( x := 10 \)
- Node 2: \( x := x - 2 \)
- Node 3: \( x := x - 2 \)
- Node 4: \( x == 9 \)
- Node B: 

Flow:

- From Node 1 to Node 2
- From Node 2 to Node 3
- From Node 3 to Node 4
- From Node 4 to Node B

Actions:

- \( in() \)
- \( !in() \)
- \( !in() \)
Bounded Verification

\[ x := 10 \]

\[ x := x - 2 \]

\[ x = 9 \]
Bounded Verification

\[ L_1 \]

1. \( x := 10 \)
2. \( x := x - 2 \)
3. \( x = 9 \)
4. \( !in() \)
5. \( !in() \)
Bounded Verification

$L_1$

$L_1 \Rightarrow x_1 = 10 \land L_2$

1. $x := 10$

2. \(\text{in}()\)

3. $x := x - 2$

4. $x = 9$

B
$L_1$

$L_1 \Rightarrow x_1 = 10 \land L_2$

$L_2 \Rightarrow (\text{in}_1 \land L_3) \lor (\neg \text{in}_1 \land L_4)$

\[
\begin{align*}
  x &:= 10 \\
  x &:= x - 2 \\
  x &= 9
\end{align*}
\]
Bounded Verification

$L_1$

$L_1 \Rightarrow x_1 = 10 \land L_2$

$L_2 \Rightarrow (\text{in}_1 \land L_3) \lor (\neg \text{in}_1 \land L_4)$

\[ \cdots \]

\[ x := 10 \]

\[ \text{in}() \]

\[ x := x - 2 \]

\[ \neg \text{in}() \]

\[ x = 9 \]

\[ \text{B} \]
Bounded Verification

$L_1$

$L_1 \Rightarrow x_1 = 10 \land L_2$

$L_2 \Rightarrow \left( \text{in}_1 \land L_3 \right) \lor \left( \neg \text{in}_1 \land L_4 \right)$

\[ \ldots \]

\[ \equiv false \]
Bounded Verification

1

x := 10

2

in()

3

x := x - 2

4

!in()

x == 9

!in()

B

2a
Bounded Verification

1. $x := 10$
2. $in()$
3. $x := x - 2$
4. $!in()$

$B$
Bounded Verification

1. \( x := 10 \)

2. \( \text{in}() \)

3. \( x := x - 2 \)

4. \( x == 9 \)

B

2a

\( !\text{in}() \)
Bounded Verification

1. \( x := 10 \)

2. \( \text{in()} \)

3. \( x := x - 2 \)

4. \( x == 9 \)

\( \text{!in()} \)
Interpolants

1

\[ x := 10 \]

2

\[ \text{in()} \]

!\text{in()}

3

\[ x := x - 2 \]

4

\[ x == 9 \]

\text{!in()}

B

2a
Interpolants

\[ x := 10 \]

\[ x := x - 2 \]

\[ x == 9 \]
Interpolants
Interpolants

A

1

x := 10

2

3

in()

!in()

!in()

x := x - 2

B

4

x == 9

2a

!in()

B
Interpolants

\( A \land B \equiv false \)
Interpolants

\[ A \land B \equiv \text{false} \]

\[ A \Rightarrow I \Rightarrow \neg B \]

(Craig 1957)
Interpolants

\[ A \land B \equiv false \]

\[ A \Rightarrow I \Rightarrow \neg B \]

(Craig 1957)
Interpolants

\[ A \land B \equiv \text{false} \]

\( A \Rightarrow I \Rightarrow \neg B \)

(Craig 1957)
DAG Interpolants

\[
\begin{align*}
x &:= 10 \\
\{x \leq 10 \land x \neq 9\} &\rightarrow \text{in()} & \{x \leq 10 \land x \neq 9\} \\
\text{!in()} &\rightarrow \{x \neq 9\} \\
\text{!in()} &\rightarrow \{x = 9\} \\
\{false\} &\rightarrow \text{B}
\end{align*}
\]
The UFO Approach

Program → Verifier

Property → Correct

Partial Program → Annotation

Bounded Verifier → Buggy!

Verified!
Finally...

\[
\begin{align*}
\{ \text{true} \} & \quad \{ x \leq 10 \land x \neq 9 \} \\
\{ x \leq 10 \land x \neq 9 \} & \quad \{ x \leq 10 \land x \neq 9 \} \\
\{ x \neq 9 \} & \quad \{ x \neq 9 \} \\
\{ \text{false} \} & \quad \{ x \leq 10 \land x \neq 9 \}
\end{align*}
\]
Finally...

\[
\begin{align*}
\{ \text{true} \} & \quad \{ x \leq 10 \land x \neq 9 \} \\
\{ x \leq 10 \land x \neq 9 \} & \quad \{ x \leq 10 \land x \neq 9 \} \\
\{ x \neq 9 \} & \quad \{ x \leq 10 \land x \neq 9 \} \\
\{ \text{false} \} & \quad \{ x \neq 9 \} \\
\end{align*}
\]
Finally...

\[ \{ x \leq 10 \land x \neq 9 \} \]

\[ \{ x \neq 9 \} \]

\[ \{ \text{false} \} \]

\[ \{ \text{true} \} \]

\[ x := 10 \]

\[ \text{in()} \]

\[ x := x - 2 \]

\[ x == 9 \]

\[ \text{!in()} \]

\[ \text{B} \]
Finally...

\[
\begin{align*}
\{ x \leq 10 \land x \neq 9 \} & \\
\{ x \leq 10 \land x \neq 9 \} & \\
\{ x \neq 9 \} & \\
\{ false \} & \rightarrow \text{B}
\end{align*}
\]

\[
\begin{align*}
\{ true \} & \\
x := 10 & \\
\{ x \leq 10 \land x \neq 9 \} & \\
\text{in()} & \\
\{ x \neq 9 \} & \\
\text{!in()} & \\
x := x - 2 & \\
x == 9 & \\
\text{B} &
\end{align*}
\]
Finally...

\[
\begin{align*}
\{ \text{true} \} & \quad x := 10 \\
\{ x \leq 10 \land x \neq 9 \} & \quad \text{in}() \\
\{ x \leq 10 \land x \neq 9 \} & \quad \text{!in}() \\
x := x - 2 & \\
x == 9 & \\
\{ \text{false} \} & \quad \text{B}
\end{align*}
\]
Finally...

\begin{itemize}
\item \{true\}
\item \{x \leq 10 \land x \neq 9\}
\item \{x \neq 9\}
\item \{false\}
\end{itemize}
Finally...

1. \( x := 10 \)
2. \( \{ x \leq 10 \land x \neq 9 \} \)
3. \( \text{in()} \)
4. \( x := x - 2 \)
5. \( \{ x \neq 9 \} \)
6. \( x == 9 \)
7. \( \{ \text{false} \} \)
8. B
The program is correct!
Problems with Interps.

1. Initialize `x` to 10.
   
2. Check if `x` is in the set.
   
3. If not in, subtract 2 from `x`.
   
4. Check if `x` equals 9.
   
5. If `x` equals 9, stop with message B.
Problems with Interps.

1. $x := 10$
2. $\{x = 10\}$
3. $\neg \text{in()}$
4. $\{x \neq 9\}$
5. $x = 9$
6. $\{\text{false}\}$

1. $\{\text{true}\}$
2. $\{x = 10\}$
3. $\text{in()}$
4. $\{x \neq 9\}$
5. $\neg \text{in()}$
6. $\text{false}$

Diagram:
- Node 1: $\{\text{true}\}$ with $x := 10$
- Node 2: $\{x = 10\}$ with $\text{in()}$
- Node 3: $\{x = 8\}$ with $x := x - 2$
- Node 4: $\{x \neq 9\}$ with $x = 9$
- Node 5: $\{\text{false}\}$ with $\neg \text{in()}$

Flow:
- From 1 to 2: $\{x = 10\}$
- From 2 to 3: $\text{in()}$
- From 3 to 2a: $\{x = 8\}$
- From 2a to 4: $\neg \text{in()}$
- From 4 to 1: $\{\text{false}\}$
The UFO Approach

Abstract Domain

Verifier

Program

Property

Partial Program

Annotation

Correct

Bounded Verifier

Buggy!
AI Invariant

1.

\[ x := 10 \]

2.

\[ \text{in()} \]

3.

\[ x := x - 2 \]

4.

\[ \text{!in()} \]

B.

\[ x == 9 \]
AI Invariant

1. $x := 10$
2. $\text{in()}$
3. $x := x - 2$
4. $\neg\text{in()}$

$x == 9$

true 1
AI Invariant

\[ x := 10 \]

\[ \text{in()} \]

\[ !\text{in()} \]

\[ x := x - 2 \]

\[ x == 9 \]

\[ \text{true} \]

\[ x = 10 \]
AI Invariant

\[ x := 10 \]

\[ \text{in()} \]

\[ x := x - 2 \]

\[ \text{!in()} \]

\[ x == 9 \]

\[ \text{true} \]

\[ x = 10 \]

\[ x \leq 10 \]
AI Invariant

1. $x := 10$
2. $\text{in()}$
3. $x := x - 2$
4. $\neg\text{in()}$

$x == 9$

B

true

$x = 10$

$x \leq 10$

$x \leq 10$
AI Invariant

\[ x := 10 \]

\[ \text{in()} \]

\[ x := x - 2 \]

\[ \text{!in()} \]

\[ x = 9 \]

\[ \text{true} \]

\[ x = 10 \]

\[ x \leq 10 \]
x := 10

\[ x \in() \]

\[ x := x - 2 \]

!in()

x == 9

B

true

x = 10

x ≤ 10

x ≤ 10

x ≤ 10
**AI Invariant**

\[
\begin{align*}
x &:= 10 \\
in() &
\end{align*}
\]

\[
\begin{align*}
x &:= x - 2 \\
!in() &
\end{align*}
\]

\[
\begin{align*}
x &== 9 \\
B &
\end{align*}
\]
AI Invariant

\begin{align*}
x &:= 10 \\
in() \\
!in() \\
x &:= x - 2 \\
x &== 9 \\
B
\end{align*}
Restricted DAGItp
true

\( x = 10 \)

\( x \neq 9 \land x \leq 10 \)

\( x \neq 9 \land x \leq 10 \)

\( x \neq 9 \land x = 9 \)

\( x \neq 9 \land x \leq 10 \)
Restricted DAGItp

true

\( x = 10 \)

\( x \neq 9 \land x \leq 10 \)

\( x \neq 9 \land x \leq 10 \)

\( x \neq 9 \land x = 9 \)

\( x \neq 9 \land x \leq 10 \)
Restricted DAGItp

true

\( x = 10 \)

\( x \neq 9 \land x \leq 10 \)

\( x \neq 9 \land x \leq 10 \)

\( x \neq 9 \land x = 9 \)
Restricted DAGItp

true

\[ x = 10 \]
\[ x \leq 10 \]
\[ x \neq 9 \land x \leq 10 \]
\[ x \neq 9 \land x = 9 \]
Restricted DAGItp

true

$x = 10$

$x \neq 9 \land x \leq 10$

$x = 8 \land x \leq 10$

$x = 6 \land x \leq 10$

$x = 9$
Restricted DAGItp

1

true

2

\(x = 10\)

2

\(x = 8 \land x \leq 10\)

2

\(x = 6 \land x \leq 10\)

4

\(x \neq 9 \land x \leq 10\)

4

\(x \neq 9 \land x = 9\)

B

\(x = 9\)
Restricted DAGItp

\[ x = 10 \]
\[ x \neq 9 \land x \leq 10 \]
\[ x = 9 \land x \neq 9 \land x \leq 10 \]
\[ x = 8 \land x \leq 10 \]
\[ x = 6 \land x \leq 10 \]

\[ \text{true} \]
Restricted DAGItp
Restricted DAGItp

true

\( x = 10 \)

\( x \neq 9 \land x \leq 10 \)

\( x = 9 \land x \leq 9 \)

\( x = 8 \land x \leq 10 \)

\( x = 6 \land x \leq 10 \)

\( x \leq 6 \)
Restricted DAGILtp

\[
\begin{align*}
1 & \quad true \\
2 & \quad x = 10 \\
2 & \quad x \neq 9 \land x \leq 10 \\
4 & \quad x = 9 \\
2 & \quad x = 6 \land x \leq 10 \\
2 & \quad x \leq 6 \\
B & \quad x \leq 6
\end{align*}
\]
Restricted DAGItp

true

\( x = 10 \)

\( x \neq 9 \land x \leq 10 \)

\( x \neq 9 \land x = 9 \)

\( x = 8 \land x \leq 10 \)

\( x = 6 \land x \leq 10 \)

\( x \leq 6 \)

\( x \leq 6 \)
DAGltp = Horn Clauses

For any edge \((i, j)\)

\[ I_i \land \tau_{i,j} \Rightarrow I_j \]

\(\text{DAGltp}\)

\(true \Rightarrow I_1\)

\(I_B \Rightarrow false\)
DAGI_{tp} = Horn Clauses

\[ \begin{align*}
\tau_{1,2} & \quad 1 \\
\tau_{2,2'} & \quad 2 \\
\tau_{2,3} & \quad 2' \rightarrow B \\
& \quad 2'' \\
\end{align*} \]

\[ \text{For any edge } (i, j) \]
\[ I_i \land \tau_{i,j} \Rightarrow I_j \]

\[ \text{true } \Rightarrow I_1 \]
\[ I_B \Rightarrow \text{false} \]

Results of Abstract Interpretation
\[ AI : V \rightarrow Expr \]
\( \text{DAGItp} = \text{Horn Clauses} \)

**Restricted DAGItp**

\[
\text{true} \Rightarrow I_1 \\
I_B \Rightarrow false
\]

For any edge \((i, j)\)

\[
I_i \land \tau_{i,j} \Rightarrow I_j
\]

**Results of Abstract Interpretation**

\[
AI : V \rightarrow Expr
\]
DAGItp = Horn Clauses

Restricted DAGItp

\[ \text{true} \Rightarrow I_1 \]
\[ I_B \land AI(v_B) \Rightarrow \text{false} \]

For any edge \((i, j)\)
\[ I_i \land AI(v_i) \land \tau_{i,j} \Rightarrow I_j \]

Results of Abstract Interpretation
\[ AI : V \rightarrow \text{Expr} \]
UFO’s Advantages
UFO’s Advantages
UFO’s Advantages
UFO’s Advantages
UFO’s Advantages

$2^n$ paths!

e.g., (McMillan 2003, Henzinger et al. 02)
UFO’s Advantages

Encode DAG as formula
Utilize symbolic reasoning
UFO’s Advantages

Encode DAG as formula
Utilize symbolic reasoning

Source: Malik and Zhang, 09
UFO’s Advantages

Strength of Abstract post

$\textit{predicate abstraction}$

$\# \text{ Calls to Bounded Verifier}$

$\textit{Interpolation-based}$
UFO’s Advantages

Strength of Abstract post

# Calls to Bounded Verifier

predicate abstraction

Interpolation-based
UFO Framework
[CAV'12]

Program with assertions → C to LLVM → Optimizations → DAG Constructor
UFO Framework
[CAV’12]

Program with assertions → C to LLVM → Optimizations

MathSAT
Z3

SMT Interface

DAG Constructor

Abstract Domain
Refinement Strategy
UFO in SVCOMP

2nd International Software Verification Competition,
TACAS 2013
UFO in SVCOMP

2nd International Software Verification Competition, TACAS 2013

Benchmarks

2000+ C programs
2 to 180 KLoC

Linux/Windows drivers, SystemC, SSH, Product Lines, etc.
UFO in SVCOMP

2nd International Software Verification Competition, TACAS 2013

<table>
<thead>
<tr>
<th>Benchmarks</th>
<th>Participants</th>
</tr>
</thead>
<tbody>
<tr>
<td>2000+ C programs</td>
<td>CPAchecker (2), BLAST, CSeq, ESBMC, LLBMC, Predator, Symbiotic, Threader, Ultimate</td>
</tr>
<tr>
<td>2 to 180 KLoC</td>
<td></td>
</tr>
<tr>
<td>Linux/Windows drivers, SystemC, SSH, Product Lines, etc.</td>
<td></td>
</tr>
</tbody>
</table>
UFO in SVCOMP

Control-Flow
Integer
SystemC
Linux Drivers
Product Lines

4 out of 10 categories
Secret Sauce

UFO Front-end
Choice of abstract domain
Parallel strategies
# My Verification Work

<table>
<thead>
<tr>
<th>Verification Algorithms</th>
<th>Decision Procedures</th>
<th>Engineering Aspects</th>
</tr>
</thead>
<tbody>
<tr>
<td>VMCAI’12</td>
<td>Symbolic abstraction</td>
<td>PLDI’12</td>
</tr>
<tr>
<td>TACAS’12</td>
<td>Beautiful interpolants</td>
<td>CAV’12</td>
</tr>
<tr>
<td>SAS’12</td>
<td></td>
<td>TACAS’13</td>
</tr>
</tbody>
</table>
## My Verification Work

<table>
<thead>
<tr>
<th>Verification Algorithms</th>
<th>Decision Procedures</th>
<th>Engineering Aspects</th>
</tr>
</thead>
<tbody>
<tr>
<td>VMCAI’12</td>
<td>Symbolic abstraction</td>
<td>PLDI’12</td>
</tr>
<tr>
<td>TACAS’12</td>
<td>Beautiful interpolants</td>
<td>CAV’12</td>
</tr>
<tr>
<td>SAS’12</td>
<td></td>
<td>TACAS’13</td>
</tr>
</tbody>
</table>
## My Verification Work

<table>
<thead>
<tr>
<th>Verification Algorithms</th>
<th>Decision Procedures</th>
<th>Engineering Aspects</th>
</tr>
</thead>
<tbody>
<tr>
<td>VMCAI’12</td>
<td><strong>Symbolic abstraction</strong>&lt;br&gt;<strong>Beautiful interpolants</strong></td>
<td>PLDI’12&lt;br&gt;CAV’12&lt;br&gt;TACAS’13</td>
</tr>
<tr>
<td>TACAS’12&lt;br&gt;SAS’12</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

?
Today’s Menu

Following Turing’s path: *The UFO approach*

UFO in practice

Future goals
Future Goals

Handling the Heap

The Concurrency Beast

Rising to the Cloud
Handling the Heap

Numerical types

Separation logic

Linked lists, etc.
Concurrency

Program → Verifier → Partial Program → Bounded Verifier

Property → Verifier → Annotation

Correct ✓

Buggy! ✗
Concurrency

Program → Verifier → Partial Program → Bounded Verifier

Property → Annotation → Correct ✓

Bounded Verifier

Correct ✓

Buggy! ❌
Concurrency

Verifier

Program

Property

Correct ✓

Partial Program

Annotation

Bounded Verifier

? ?

Buggy! ×

Correct ✓
Distributing Verification

[PLDI’12]

Program

foo() → bar() → baz() → qux()
Distributing Verification

[PLDI’12]

Program

foo() → bar() → baz() → qux()

summary DB
(Most) Related Work
(Most) Related Work

Rybalchenko and company

• Solving recursion free Horn clauses (APLAS’11, POPL’11)
• Solving recursive Horn clauses (PLDI’12, MSR-TR’13)
(Most) Related Work

Rybalchenko and company

• Solving \textit{recursion free} Horn clauses (APLAS’11, POPL’11)
• Solving \textit{recursive} Horn clauses (PLDI’12, MSR-TR’13)

Hoder and Bjorner (SAT’12)
(Most) Related Work

Rybalchenko and company

- Solving recursion free Horn clauses (APLAS’11, POPL’11)
- Solving recursive Horn clauses (PLDI’12, MSR-TR’13)

Hoder and Bjorner (SAT’12)

Podelski and company

- Automata-theoretic view (POPL’10, POPL’13)
(Most) Related Work

Rybalchenko and company

• Solving recursion free Horn clauses (APLAS’11, POPL’11)
• Solving recursive Horn clauses (PLDI’12, MSR-TR’13)

Hoder and Bjorner (SAT’12)

Podelski and company

• Automata-theoretic view (POPL’10, POPL’13)
Summary

Push-button verification
The UFO approach
Big questions ahead
Thank You!

bitbucket.org/arieg/ufo