Automatically Proving Memory Safety and Termination of C-Programs

Alexander Weinert
RWTH Aachen University

Research Area Computer Science 2
Published: (Termination with Pointer Arithmetic, Ströder et al., 2014)

Feb 23, 2015
int strlen(char* str) {
    char* s = str;
    while(*(++s));
    return s-str;
}

str

↓

0 ... 0
An Example

```c
int strlen(char* str) {
    char* s = str;
    while((*s)++);
    return s-str;
}
```
int strlen(char* str) {
    char* s = str;
    while(*(s++));
    return s-str;
}

str

\[\text{...} 0\]
An Example

```c
int strlen(char* str) {
    char* s = str;
    while(*s) s++;
    return s - str;
}
```

![Diagram of strlen function with pointers s and str]
Big Picture

**THE C PROGRAMMING LANGUAGE**

1. LLVM
2. Symbolic Execution Graph
3. Integer Transition System
4. YES/NO/MAYBE

\[ f(x) \to g(x + 1) \]
\[ g(x) \to g(x - 1) \]
C to LLVM

THE C PROGRAMMING LANGUAGE

LLVM

Symbolic Execution Graph

f(x) → g(x + 1)
g(x) → g(x - 1)

Integer Transition System

YES/NO/MAYBE
```c
int strlen(char *str) {
    char *s = str;
    ...
}
```

```asm
define i32 strlen(i8* str) {
    c0 = load i8* str
    ...
}
```

```asm
strlen_entry:
    push edi
    ...
```

LLVM Compiler Infrastructure

C-program

LLVM Internal Representation

Assembler
Some LLVM Instructions

- Control Flow Instructions
  - ret, br, call, ...
- Arithmetic Instructions
  - add, icmp, ...
- Bitwise Instructions
  - shl, and, ...
- Memory Instructions
  - alloc, load, store, ...

Complete Reference: http://llvm.org/docs/LangRef.html
LLVM to Symbolic Execution Graph

THE

C

PROGRAMMING
LANGUAGE

LLVM

Symbolic Execution Graph

\[ f(x) \rightarrow g(x + 1) \]
\[ g(x) \rightarrow g(x - 1) \]

Integer Transition System
Problem Definition

Given: LLVM Program, Entry Point

Goal: Description of \textit{at least all possible runs} from the entry point

Idea: Abstract interpretation of program states
Abstract States

<table>
<thead>
<tr>
<th>Position:</th>
<th>(0, strlen)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alloc:</td>
<td>[\text{alloc}(str, v_{end})]</td>
</tr>
<tr>
<td>Stack → Memory:</td>
<td>$v_{end} \leftarrow 0$</td>
</tr>
<tr>
<td>Relations:</td>
<td>$\emptyset$</td>
</tr>
</tbody>
</table>

Entry Point:

\[
\text{define i32 strlen(i8* str) \{}
\]
\[
c0 = \text{load i8* str}
\]
\[
\ldots
\]


Evaluation of Abstract States

\[ c_0 = \text{load}\ i8*\ str \]

<table>
<thead>
<tr>
<th>Position:</th>
<th>(0, strlen)</th>
<th>Position:</th>
<th>(1, strlen)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alloc:</td>
<td>[alloc(str, v_{end})]</td>
<td>Alloc:</td>
<td>[alloc(str, v_{end})]</td>
</tr>
<tr>
<td>Stack → Memory:</td>
<td>v_{end} ↦ 0</td>
<td>Stack → Memory:</td>
<td>v_{end} ↦ 0, str ↦ c0</td>
</tr>
<tr>
<td>Relations:</td>
<td>\emptyset</td>
<td>Relations:</td>
<td>\emptyset</td>
</tr>
</tbody>
</table>
Refinement of Abstract States

\[ c0\text{zero} = \text{icmp eq i8 c0, 0} \]

<table>
<thead>
<tr>
<th>Position:</th>
<th>Alloc:</th>
<th>Stack → Memory:</th>
<th>Relations:</th>
</tr>
</thead>
<tbody>
<tr>
<td>(1, strlen)</td>
<td>([\text{alloc}(\text{str}, \text{v}_{\text{end}})])</td>
<td>(\text{v}_{\text{end}} \mapsto 0, \text{str} \mapsto \text{c0})</td>
<td>(\emptyset)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Position:</th>
<th>Alloc:</th>
<th>Stack → Memory:</th>
<th>Relations:</th>
</tr>
</thead>
<tbody>
<tr>
<td>(2, strlen)</td>
<td>([\text{alloc}(\text{str}, \text{v}_{\text{end}})])</td>
<td>(\text{v}_{\text{end}} \mapsto 0, \text{str} \mapsto \text{c0})</td>
<td>(?)</td>
</tr>
</tbody>
</table>
Refinement of Abstract States

\[ \text{c0zero} = \text{icmp eq i8 c0, 0} \]

<table>
<thead>
<tr>
<th>Position: ( (1, \text{strlen}) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alloc: ([\text{alloc} (\text{str}, \nu_{\text{end}})])</td>
</tr>
<tr>
<td>Stack (\rightarrow) Memory: (\nu_{\text{end}} \rightarrow 0, \text{str} \rightarrow \text{c0})</td>
</tr>
<tr>
<td>Relations: (\emptyset)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Position: ( (1, \text{strlen}) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alloc: (\ldots)</td>
</tr>
<tr>
<td>Stack (\rightarrow) Memory: (\ldots)</td>
</tr>
<tr>
<td>Relations: ({\text{c0} = 0})</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Position: ( (1, \text{strlen}) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alloc: (\ldots)</td>
</tr>
<tr>
<td>Stack (\rightarrow) Memory: (\ldots)</td>
</tr>
<tr>
<td>Relations: ({\text{c0} \neq 0})</td>
</tr>
</tbody>
</table>
Merging of Abstract States

```c
int strlen(char *str) {
    char *s = str;
    while(*s) s++;
    return s-str;
}
```
Merging of Abstract States
### Merging of Abstract States

<table>
<thead>
<tr>
<th>Position: ((3, \text{strlen}))</th>
<th>Position: ((3, \text{strlen}))</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Alloc:</strong> ([\textit{alloc}(\textit{str}, \textit{v}_{\text{end}})])</td>
<td><strong>Alloc:</strong> ([\textit{alloc}(\textit{str}, \textit{v}_{\text{end}})])</td>
</tr>
<tr>
<td><strong>Stack → Memory:</strong></td>
<td><strong>Stack → Memory:</strong></td>
</tr>
<tr>
<td>(s \leftarrow \textit{i}_8 c, \ldots)</td>
<td>(s \leftarrow \textit{i}_8 c, \ldots)</td>
</tr>
<tr>
<td><strong>Relations:</strong> ({s = \textit{str} + 1, \ldots})</td>
<td><strong>Relations:</strong> ({s = \textit{str} + 2, \ldots})</td>
</tr>
</tbody>
</table>

### Relations

- \(Position: (3, \text{strlen})\)
- **Alloc:** \([\textit{alloc}(\textit{str}, \textit{v}_{\text{end}})]\)
- **Stack → Memory:**
  - \(s \leftarrow \textit{i}_8 c, \ldots\)
- **Relations:** \(\{s \geq \textit{str}, \ldots\}\)
Merging of Abstract States
Summary: Symbolic Execution Graphs

- Evaluation
- Refinement
- Generalization
- Instance
Symbolic Execution Graph to Integer Transition System

Symbolic Execution Graph

f(x) → g(x + 1)
g(x) → g(x - 1)

Integer Transition System

YES/NO/MAYBE
Integer Transition Systems

Term Rewriting System:

\[ f(x, y) \rightarrow g(y, x) \]

Integer Transition System:

\[ f(x, y) \rightarrow g(y + 1, x - 2) \]

Integer Transition System:

\[ f(x, y) \rightarrow g(y + 1, x - 2) \quad | \quad x > 0 \]
Symbolic Execution Graph to Integer Transition System

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Position:</strong> $\mathbf{(1,\text{strlen})}$</td>
<td><strong>Position:</strong> $\mathbf{(1,\text{strlen})}$</td>
</tr>
<tr>
<td><strong>Allocations:</strong> [$\text{alloc}(\text{str}, \text{v}_{\text{end}})$]</td>
<td><strong>Allocations:</strong> [$\text{alloc}(\text{str}, \text{v}_{\text{end}})$]</td>
</tr>
<tr>
<td><strong>Stack $\rightarrow$ Memory:</strong></td>
<td><strong>Stack $\rightarrow$ Memory:</strong></td>
</tr>
<tr>
<td>$\text{v}_{\text{end}} \leftarrow \text{i8} \ 0$, $\text{str} \leftarrow \text{i8} \ c0$</td>
<td>$\text{v}_{\text{end}} \leftarrow \text{i8} \ 0$, $\text{str} \leftarrow \text{i8} \ c0$</td>
</tr>
<tr>
<td><strong>Relations:</strong> $\emptyset$</td>
<td><strong>Relations:</strong> ${c0 \neq 0}$</td>
</tr>
</tbody>
</table>

$f_A(\text{str}, \text{v}_{\text{end}}, c0) \rightarrow f_B(\text{str}, \text{v}_{\text{end}}, c0) \mid c0 \neq 0$
Termination of Integer Transition System

Symbolic Execution Graph

\[ f(x) \to g(x + 1) \]
\[ g(x) \to g(x - 1) \]

Integer Transition System
Termination of Integer Transition System

Well-studied problem

Use known techniques to show termination, e.g. (Termination of Integer Term Rewriting, Fuhs et al., 2009)
Termination of Integer Transition System

THE

C

PROGRAMMING LANGUAGE


LLVM

Symbolic Execution Graph

YES/NO/MAYBE

\[ f(x) \rightarrow g(x + 1) \]
\[ g(x) \rightarrow g(x - 1) \]

Integer Transition System
Overview

THE C PROGRAMMING LANGUAGE

LLVM

Symbolic Execution Graph

YES/NO/MAYBE

\[ f(x) \rightarrow g(x + 1) \]

\[ g(x) \rightarrow g(x - 1) \]

Integer Transition System
My Contributions

THE C PROGRAMMING LANGUAGE

LLVM

Symbolic Execution Graph

f(x) → g(x + 1)
g(x) → g(x - 1)

YES/NO/MAYBE

Integer Transition System
My Contributions

Given: Abstract State $s$, Integer Relation $r$

Question: Does $s \models r$?
My Contributions

- Reduction from state to set of arithmetic relations
- Inference of knowledge from states
  - Formulation of inference in terms of integer relations
  - New framework for inference
  - Parameterization of framework with abstract arithmetic domain
  - Formulation of existing inference in framework
My Contributions

- Use of Octagon Domain for inference of relations
  - (The Octagon Abstract Domain, Miné, 2006)
- Experimental comparison
  - “Traditional” inference
  - Inference in framework
  - New inference based on Octagons
Evaluation: Empirical Evaluation

79 Integer Programs, Timeout: 300s

Result

YES

MAYBE

NO

TIMEOUT

Number of Examples
Evaluation: Empirical Evaluation

79 Integer Programs, Timeout: 300s

![Graph showing runtime vs. number of examples for different tools.](image-url)
Evaluation: Empirical Evaluation

129 Pointer Programs, Timeout: 300s
Evaluation: Empirical Evaluation

129 Pointer Programs, Timeout: 300s

![Graph showing runtime vs. number of examples for different programs. The x-axis represents the number of examples, and the y-axis represents the runtime in seconds. The graph shows the performance of AProVE, KITTeL, T2, and Tan.]
Complete evaluation at
http://aprove.informatik.rwth-aachen.de/eval/Pointer/
Thank you for your attention

www.alexanderweinert.net
alexander.weinert@rwth-aachen.de
THE C PROGRAMMING LANGUAGE

LLVM

Symbolic Execution Graph

YES/NO/MAYBE

$f(x) \rightarrow g(x + 1)$

$g(x) \rightarrow g(x - 1)$

Integer Transition System