Static Analysis By Elimination

Pavle Subotic, Andrew Santosa, Bernhard Scholz
pavle.subotic@it.uu.se, andrew.santosa@usyd.edu.au,
bernhard.scholz@usyd.edu.au

Uppsala University, Sweden
University of Sydney, Australia

Bytecode workshop 2013
Introduction

- **Range Analysis**
  - Finds *lower* and *upper* bounds of variables values
- **Challenges**
  - Conceptionally infinitely ascending chains
  - Identify Loops
- **Existing techniques**
  - Relies on code structure (e.g. Astrée [Cousot et al., 2006])
  - Require a pre-processing stage to discover loop headers ([Bourdoncle, 1993])
Introduction

- Our technique:
  1. Extends elimination-based data flow analysis to a lattice with infinite ascending chains
  2. Fast termination
  3. Loops are detected intrinsically within the data flow analysis.

- Implemented as an analysis pass in the LLVM compiler framework.
Motivating Example

```
int i, k = 0;
int arr[5]; ...

if (i < 5)
goto B2
else
goto B7;

int j = 0;
if (i < 5)
goto B3
else
goto B5;

I1: i ≥ 0 ∧ j ≤ 3
if (arr[j] > arr[j+1])
goto B5
else
goto B6;

swap(arr, j, j+1);
k++;

B0:
B1
B2
B3
B4
B5
B6
B7
```
Foundations

- Range Analysis is a complete lattice
- $x \sqsubseteq y$, $x$ is as or less precise than $y$
- $\top$ least element (least precise),
- $\bot$ greatest element, so $\top \sqsubseteq \bot$
- $\sqcap$ merges information
- $\sqcap$ constrains information
Representing Information with Intervals
Some Existing Techniques

- **Iterative Data-Flow Analysis [Kildall, 1973]**:
  - A technique for iteratively gathering variable information at various points in a computer program.
  - Operates on finite and short lattice structures

- **Abstract Interpretation [Cousot & Cousot, 1977]**:
  - A theory of sound approximation of the semantics of computer programs
  - Approximating the execution behaviour of a computer program
  - Additional theory of widening/narrowing to accelerate convergence, required with high and unbounded domains
Iterative Data-Flow Analysis

- Input in the form of a Control Flow Graph (CFG)
- Initialise to $\bot$
- Every block transforms the values
- Iterate through CFG until a fixpoint is reached
Attempt 1: Iterative Data-Flow Analysis

```
if (a < 3)
  a = [5,5]
condition: a < 3
condition: a >= 3
a = [1, 4]
[1,4] ∩ [-∞, 2] = [1,2]
[1,4] ∩ [3, ∞] = [3,4]
[5,5] ∪ [3,4] = [3,5]
```
Attempt 1: Iterative Data-Flow Analysis
With Kleene Iteration

```java
int j = 0;
int i = 0;

if (j <= 3)
    j++;
k++;
...
```

Subotic, Santosa, Scholz
With Kleene Iteration

\[ \forall l_i \in L. l_1 \sqsubseteq l_2 \sqsubseteq l_3 \sqsubseteq l_4 \ldots \sqsubseteq l_n \]

where:
In the example, when the inner loop is first visited, we have that \( j \mapsto [0, 0] \) and \( k \mapsto [0, 0] \). In subsequent visits,

\[
\begin{align*}
  j &\mapsto [0, 1] \text{ and } k \mapsto [0, 1], \\
  j &\mapsto [0, 2] \text{ and } k \mapsto [0, 2], \\
  j &\mapsto [0, 3] \text{ and } k \mapsto [0, 3], \\
  \vdots \\
  j &\mapsto [0, 4] \text{ and } k \mapsto [0, \infty].
\end{align*}
\]
The Problem: Slow Termination

- Impractically slow termination
  - Conditions not incorporating increasing variables
  - Large loop bounds
Attempt 2: Abstract Interpretation

- General method to compute a sound approximation of program semantics
  - Define an abstract semantics, soundly connect to the concrete semantics
  - Soundness ensures that if a property does not hold in the abstract world, it will not hold in the concrete world
  - Define widening and narrowing operator
Abstract Interpretation

Widening and narrowing enforce termination

- Widening safely approximates the fixpoint solution
- Narrowing recovers some precision
Attempt 2: Abstract Interpretation

- Red / FP
- Ext / FP
- Fixed-Point (FP)
- ⊤
- ⊥
- Less precision
- More precision
- Widening
- Narrowing
Abstract Interpretation

- Requires to know where to perform widening
- Previously approaches
  - Use the syntax to determine the loop
  - Perform complicated pre-processing to find loop headers
Our Approach

- Discovers loops implicitly using elimination-based data flow analysis
- Various acceleration techniques can be embedded such as widening and narrowing
Our Approach

- Elimination-based approach: Based on Gaussian elimination
- Instead of iterating, we eliminate variables from the flow equations
  - substitution
    e.g. $x = \text{true}, y = x \lor \text{false} \leadsto y = \text{true} \lor \text{false}$
  - loop-breaking
    e.g. $x = x \land \text{true} \leadsto x = \text{true}$
- When all variables are eliminated, we compute a solution
Elimination-based Approach Example - Diverging

Figure: An Irreducible CFG of a Diverging Program
Elimination

EQS = \[
\begin{cases}
X_0 = f_0(\top) \\
X_1 = f_1(X_0, X_2) \\
X_2 = f_2(X_0, X_1)
\end{cases}
\]

Substitution \(\leadsto\)

EQS_0 = \[
\begin{cases}
X_0 = f_0(\top) \\
X_1 = f_1(f_0(\top), X_2) \\
X_2 = f_2(f_0(\top), X_1)
\end{cases}
\]

Substitution \(\leadsto\)

EQS_1 = \[
\begin{cases}
X_0 = f_0(\top) \\
X_1 = f_1(f_0(\top), X_2) \\
X_2 = f_2(f_0(\top), f_1(f_0(\top), X_2))
\end{cases}
\]

Break Loop, Substitute Back \(\leadsto\)

EQS_2 = \[
\begin{cases}
X_0 = f_0(\top) \\
X_1 = f_1(f_0(\top), F^*(f_2(f_0(\top), f_1(f_0(\top), X_2), X'_2))) \\
X_2 = F^*(f_2(f_0(\top), f_1(f_0(\top), X_2), X'_2))
\end{cases}
\]
Solve

- \( X_1 = f_1(f_0(\top), F^*(f_2(f_0(\top), f_1(f_0(\top), X_2), X'_2))) \)
- \( F^* \) performs widening and narrowing
LLVM Prototype

- Implemented in LLVM for core instructions
- Implementation supports both Intervals and Symbolic Intervals
### Table: Motivating Example

<table>
<thead>
<tr>
<th>Block</th>
<th>$i$</th>
<th>$j$</th>
<th>$k$</th>
</tr>
</thead>
<tbody>
<tr>
<td>B0</td>
<td>[0, 0]</td>
<td>⊥</td>
<td>[0, 0]</td>
</tr>
<tr>
<td>B1</td>
<td>[0, 5]</td>
<td>[0, 5]</td>
<td>[0, $\infty$]</td>
</tr>
<tr>
<td>B2</td>
<td>[0, 4]</td>
<td>[0, 0]</td>
<td>[0, $\infty$]</td>
</tr>
<tr>
<td>B3</td>
<td>[0, 4]</td>
<td>[0, 5]</td>
<td>[0, $\infty$]</td>
</tr>
<tr>
<td>B4</td>
<td>[0, 4]</td>
<td>[1, 4]</td>
<td>[1, $\infty$]</td>
</tr>
<tr>
<td>B5</td>
<td>[1, 5]</td>
<td>[5, 5]</td>
<td>[1, $\infty$]</td>
</tr>
<tr>
<td>B6</td>
<td>[5, 5]</td>
<td>[5, 5]</td>
<td>[0, $\infty$]</td>
</tr>
</tbody>
</table>
### Table: Variable Bounds Per Test Case

<table>
<thead>
<tr>
<th>Test</th>
<th>Exact</th>
<th>Bounded</th>
<th>Part Widen</th>
<th>Full Widen</th>
</tr>
</thead>
<tbody>
<tr>
<td>T1</td>
<td>1</td>
<td>5</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>T2</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>T3</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>T4</td>
<td>1</td>
<td>3</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>T5</td>
<td>0</td>
<td>10</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>T6</td>
<td>3</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>T7</td>
<td>1</td>
<td>2</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>T8</td>
<td>4</td>
<td>4</td>
<td>5</td>
<td>0</td>
</tr>
<tr>
<td>T9</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>5</td>
</tr>
<tr>
<td>T10</td>
<td>1</td>
<td>0</td>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td>T11</td>
<td>2</td>
<td>2</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>T12</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>T13</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>T14</td>
<td>3</td>
<td>6</td>
<td>6</td>
<td>0</td>
</tr>
<tr>
<td>T15</td>
<td>3</td>
<td>5</td>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td><strong>All</strong></td>
<td><strong>27</strong></td>
<td><strong>45</strong></td>
<td><strong>26</strong></td>
<td><strong>6</strong></td>
</tr>
<tr>
<td><strong>(%)</strong></td>
<td><strong>26</strong></td>
<td><strong>43</strong></td>
<td><strong>25</strong></td>
<td><strong>6</strong></td>
</tr>
</tbody>
</table>
Summary

- Implemented in the LLVM Compiler Framework
- Feasibility shown using several test programs
Some Future Work

- Conduct comparison with existing techniques
- Add non-numerical domains
- Improve precision through additional abstract domains (Template Polyhedra [Sankaranarayanan et al., 2005])
- Integrate with acceleration methods such as policy iteration [Gawlitza & Seidl, 2007]
References I


References II

