Towards Abstraction Refinement in TVLA

Alexey Loginov
UW Madison
alexey@cs.wisc.edu
Need for Heap Data Analysis

• Morning talks by UW students
  – Static analyses for de-obfuscation of code
  – Limited ability to analyze heap data
  – Need understanding of possible shapes
• Want to handle unbounded heap object creation
• Java objects (e.g. threads) are in heap
Linked List Abstractions

• Informally

• Formally
Linked List Abstractions

• Informally

• Formally
Linked List Abstractions

- **Informally**
  
- **Formally**

![Diagram](image)
Linked List Abstractions

- Informally

- Formally
Abstraction Refinement

- Devising good analysis abstractions is hard
  - Precision/cost tradeoff
    - Too coarse: “Unknown” answer
    - Too refined: high space/time cost
- Start with simple (and cheap) abstraction
- Successively refine abstraction
  - Adaptive algorithm
Abstraction Refinement

• Iterative process
  – Create an abstraction (e.g. set of predicates)
  – Run analysis
  – Detect indefinite answers (stop if none)
  – Refine abstraction (e.g. add predicates)
  – Repeat above steps
Previous Work on Abstraction Refinement

• Counterexample guided [Clarke et al]
  – Finds shortest invalid prefix of spurious counterexample
  – Splits last state on prefix into less abstract ones

• SLAM toolkit [Ball, Rajamani]
  – Temporal safety properties of software
  – Identifies correlated branches
Abstraction Refinement for TVLA: New Challenges

• Need to refine abstractions of linked data structures
  – Identify appropriate new distinctions between
    • Nodes
    • Structures
• Need to derive the associated abstract interpretation
  – What are the update formulas?
Control Over the Merging of Nodes

- Unary abstraction predicates
  \[ r_x(v) = \exists v' : (x(v') \text{ reachable from } n^*(v',v)) \]
  - Distinguish nodes reachable from x (and y)
  - Can now tell if lists are disjoint
Control Over the Merging of Structures

• Nullary abstraction predicates

\[ \text{nn}_x() = \exists v : x(v) \]

• Distinguish structures based on whether x is NULL
• Can now tell that x is NULL whenever y is (and vice versa)

\[
\begin{align*}
\text{x} & \quad \text{y} \\
\text{empty structure}
\end{align*}
\]
Need for Update Formulas

• Re-evaluating formula is imprecise

\[ r_x(v) = \exists v' : (x(v') \uparrow n^*(v',v)) \]

Action “x == NULL” (no change to heap)
Creating Update Formulas Automatically

• Frees user from having to provide update formulas
  – A lot of work (esp. with iterative refinement)
  – Error prone
• Idea: Finite differencing of formulas

\[ F(\phi) = p_{\phi} \triangleleft \phi^\Delta \]
Differencing $\approx$ Differentiation

\[
\begin{align*}
0^\Delta &= 0 \\
1^\Delta &= 0 \\
(\varphi \psi)^\Delta &= \varphi^\Delta \psi^\Delta \\
(f+g)'(x) &= f'(x) + g'(x) \\
(f\cdot g)'(x) &= f'(x)\cdot g(x) + f(x)\cdot g'(x)
\end{align*}
\]
Laws for $\phi^\Delta$

$0^\Delta = 0$

$1^\Delta = 0$

$(v = w)^\Delta = 0$

$(\phi \vartriangleleft \psi)^\Delta = \phi^\Delta \vartriangleleft \psi^\Delta$

$(\phi \vartriangleright \psi)^\Delta = \phi \vartriangleright \psi^\Delta \vartriangleleft \phi^\Delta \vartriangleright \psi \vartriangleleft \phi^\Delta \vartriangleright \psi^\Delta$

$(\uparrow v: \phi)^\Delta = (\uparrow v: \phi) \vartriangleleft \phi^\Delta$

$(\text{TC}: \phi)(v,w)^\Delta = (\text{TC}: \phi)(v,w) \vartriangleleft (\text{TC}: \phi \vartriangleright \phi^\Delta)(v,w)$

8/5/2003  alexey@cs.wisc.edu
Formula Differencing Limitations

- Differencing output often imprecise

\[ r_x(v) = \exists v' : (x(v') \mathbin{\text{\rightarrow}} n^*(v',v)) \]

Action “x = x → n”
Reducing Loss of Precision from Differencing

• Semantic minimization of formulas
  – Simplest case: propositional formulas

• Work in progress
  – Removing unnecessary reevaluation
    • Finding common sub-formulas
  – Improvements to materialization
  – Exploiting important special cases
    • E.g., reachability in lists
Three-Valued Logic

- **1**: True
- **0**: False
- **1/2**: Unknown
- A join semi-lattice: $0 \sqcup 1 = 1/2$

```
\[
\begin{array}{c}
\frac{1}{2} \\
\downarrow \\
0 \quad 1
\end{array}
\]
```

Information order:

- $0 \leq \frac{1}{2}$
- $1 \leq \frac{1}{2}$
Semantic Minimization

- $\overline{\phi} \rightarrow (A)$: Value of formula $\phi$ under assignment $A$
- In 3-valued logic, $\overline{\phi} \rightarrow (A)$ may equal $\frac{1}{2}$
  \[ \overline{p + p'} \rightarrow ([p \equiv 0]) = 1 \]
  \[ \overline{p + p'} \rightarrow ([p \equiv \frac{1}{2}]) = \frac{1}{2} \]
  \[ \overline{p + p'} \rightarrow ([p \equiv 1]) = 1 \]
- However,
  \[ \overline{1} \rightarrow ([p \equiv 0]) = 1 \]
  \[ \overline{1} \rightarrow ([p \equiv \frac{1}{2}]) = 1 \]
  \[ \overline{1} \rightarrow ([p \equiv 1]) = 1 \]
Semantic Minimization

\[ \iff 1 \rightarrow ([p \equiv 0]) = 1 = \iff p + p' \rightarrow ([p \equiv 0]) \]

\[ \iff 1 \rightarrow ([p \equiv \frac{1}{2}]) = 1 \iff \frac{1}{2} = \iff p + p' \rightarrow ([p \equiv \frac{1}{2}]) \iff 1 \rightarrow ([p \equiv 1]) = 1 = \iff p + p' \rightarrow ([p \equiv 1]) \]

2-valued logic: 1 is equivalent to \( p + p' \)

3-valued logic: 1 is better than \( p + p' \)

For a given \( \varphi \), is there a best formula? Yes!
Minimal?

\[ x + x' \]  No!
\[ x \equiv x' \]  Yes!
\[ xy + x'z \]  No!
\[ xy + x'y' \]  Yes!
\[ xy + x'z + yz \]  Yes!
\[ xy' + x'z' + yz \]  No!
Example

Original formula ($\varphi$)

$$xy + x'z$$

Minimal formula ($\psi$)

$$xy + x'z + yz$$

<table>
<thead>
<tr>
<th>$A$</th>
<th>$\leftarrow \psi \rightarrow (A)$</th>
<th>$\leftarrow \varphi \rightarrow (A)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$[x \approx \frac{1}{2}, y \approx 1, z \approx 1]$</td>
<td>1</td>
<td>$\frac{1}{2}$</td>
</tr>
</tbody>
</table>
Example

Original formula ($\varphi$)
\[ xy' + x'z' + yz \]

Minimal formula ($\psi$)
\[ x'y' + x'z' + yz + xy' + xz + y'z' \]

<table>
<thead>
<tr>
<th>$A$</th>
<th>$\longleftrightarrow \psi \rightarrow (A)$</th>
<th>$\longleftrightarrow \varphi \rightarrow (A)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$[x \approx \frac{1}{2}, y \approx 0, z \approx 0]$</td>
<td>1</td>
<td>$\frac{1}{2}$</td>
</tr>
<tr>
<td>$[x \approx 0, y \approx 1, z \approx \frac{1}{2}]$</td>
<td>1</td>
<td>$\frac{1}{2}$</td>
</tr>
<tr>
<td>$[x \approx 1, y \approx \frac{1}{2}, z \approx 1]$</td>
<td>1</td>
<td>$\frac{1}{2}$</td>
</tr>
</tbody>
</table>
Semantic Minimization

- When $\phi$ contains no occurrences of $\frac{1}{2}$ and $\equiv$
  $\{\text{primes}(\leftrightarrow \phi \rightarrow)\}$
- In general, somewhat more complicated
  - Represent $\leftrightarrow \phi \rightarrow$ with a pair
    floor: $\& \leftrightarrow \phi \rightarrow \& 
    \&^{1/2} = 0$
  ceiling: $\& \leftrightarrow \phi \rightarrow \& 
    \&^{1/2} = 1$
  - Semantically minimal formula

8/5/2003 alexey@cs.wisc.edu
Current and Future Work

- Finite differencing of formulas
- Minimization of first-order formulas
- Generation of instrumentation predicates
Towards Abstraction Refinement in TVLA

Alexey Loginov
UW Madison
alexey@cs.wisc.edu