Relational Inductive Shape Analysis

Bor-Yuh Evan Chang
University of California, Berkeley

Xavier Rival
INRIA

POPL 2008
Example: Removing duplicates

Concrete Example

Example: Removing duplicates

Invariant/Abstraction

Cur = l→next;

while (Cur != null) {
    Cur = remove_if_dup(Cur);
    Cur = Cur→next;
}

Cur = l→next;
Utilize “dynamic checking code” as specification for static analysis

**Checking code** expresses a precise invariant of interest (but only at “steady states”)

\[
\text{sorteddll}(l, \text{prev}, \text{min}) = \\
\text{if } (l = \text{null}) \text{ then true else } \\
\text{if } l \rightarrow \text{prev} = \text{prev and } \text{min} \leq l \rightarrow \text{val} \text{ and } \\
\text{sorteddll}(l \rightarrow \text{next}) \text{ then true else false}
\]

assert(sorteddll(l, null, 0));

cur = l;

while (cur != null) {
    cur = remove_if_dup(cur);
    cur = cur \rightarrow \text{next};
}

assert(sorteddllset(l, null, 0));

automatically generalize for intermediate states
Our framework is ...

An automated **shape analysis** with a precise memory abstraction based around **invariant checkers**.

```plaintext
sorteddll(l, prev, min) =
  if (l = null) then
    true
  else
    l's prev = prev and
    min ≤ l's val and
    sorteddll(l's next, l, l's val)
```

- **Compact abstraction**
  - Data structure-specific based on properties of interest to the developer

- **Extensible**
  - Parametric in developer-supplied checkers
Challenges

cur = l->next;
while (cur != null) {

if (cur->prev->val == cur->val) {
  cur = cur->prev; remove_after(cur);
}

cur = cur->next;
}

“split” segments (back pointers)

“sorted dl set segment (v ≤ \bullet)”

“sorted dl list (\bullet ≤ v)”

“sorted dl list (v ≤ \bullet)”

“sorted dl set segment (\bullet ≤ u)”

u < v = w

numerical constraints (linking shape and data) (see paper)
Shape analysis is an abstract interpretation on memory states with ...
sorteddll(l, prev, min) =
if (l = null) then
  true
else
  l→prev = prev and
  min ≤ l→val and
  sorteddll(l→next,l,l→val)

checkers

Outline

1. Materialization and update
   - Widening
   - Abstract interpretation

2. Type “pre-analysis”
   - Shape analyzer

- See paper
Abstract memory using inductive predicates

Edges represent disjoint memory regions

values (e.g., address)

points-to (memory cell)

checker (inductive pred)

update: cur→next = cur→next→next

One traversal parameter with fields

\[ \pi \]:= \exists \eta.\
\[
\begin{align*}
\text{emp} & \Rightarrow \pi = \text{null} \\
\rho & \Rightarrow \pi \rightarrow \eta \Rightarrow \text{dll}(\pi) \\
\pi & \neq \text{null}
\end{align*}
\]

\[ cur = l\rightarrow next; \]
while (cur != null)
{ \[
\begin{align*}
\text{if} & \ (cur\rightarrow prev\rightarrow val \ \\
& \quad == \ cur\rightarrow val) \\
\{ & \ \\n\text{cur} & = \ cur\rightarrow prev; \\
\text{remove_after}(\text{cur}); \\
\}
\]
}
Materialize by unfolding inductive definition

```
cur = l.next;
while (cur != null)
{
  if (cur.prev.val == cur.val)
  {
    cur = cur.prev;
    remove_after(cur);
  }
  cur = cur.next;
}
```
Segments as partial checkers

Summary

\[
\alpha \xrightarrow{i} \gamma \xrightarrow{\text{dll}(\beta)} \beta
\]

Instance

null \xleftarrow{\text{prev}} \alpha \xrightarrow{\text{next}} \beta \xrightarrow{\text{next}} \gamma \xrightarrow{\text{next}} \delta \xrightarrow{\text{null}}

Checker “Run”

\[
\alpha.\text{dll}(\text{null}) \xrightarrow{i} \beta.\text{dll}(\alpha) \xrightarrow{i} \gamma.\text{dll}(\beta) \xrightarrow{i} \delta.\text{dll}(\gamma) \xrightarrow{i} \text{null. dll}(\delta)
\]

\[
\alpha \xrightarrow{\text{c}(\gamma)} \beta \xrightarrow{\text{c}'(\gamma')}
\]
Segments as partial checkers

Summary

Instance

Checker “Run”

\[ i = 0 \]

\[ \gamma.\text{dll}(\beta) \]

\[ \beta = \text{null} \]

\[ \delta.\text{dll}(\gamma) \]

\[ \text{null.}\text{dll}(\delta) \]

\[ c = c' \]

\[ \alpha = \beta \]

\[ \gamma = \gamma' \]
To unfold backward, split the segment and then unfold forward

```
cur = l\rightarrow next;
while (cur != null)
{
  if (cur\rightarrow prev\rightarrow val == cur\rightarrow val)
  {
    cur = cur\rightarrow prev;
    remove_after(cur);
  }
  cur = cur\rightarrow next;
}
```
sorteddll(l, prev, min) =
if (l = null) then
  true
else
  l→prev = prev and
  min ≤ l→val and
  sorteddll(l→next,l,l→val)

checkers

type pre-analysis

shape analyzer

1. materialization and update

abstract interpretation

Bor-Yuh Evan Chang and Xavier Rival - Relational Inductive Shape Analysis
Types for deciding where to unfold

Summary

Instance

Checker “Run”

If it exists, where is:
\[ \gamma \rightarrow \text{next} ? \quad 0 \]
\[ \beta \rightarrow \text{next} ? \quad -1 \]

Checker Definition

\[ \pi: \{ \text{next}(0), \text{prev}(0) \} \]
\[ \rho: \{ \text{next}(-1), \text{prev}(-1) \} \]

\[ \pi_{\text{dll}(\rho)} := \exists \eta. \]

emp
\[ \pi = \text{null} \]
\[ \pi \neq \text{null} \]

\[ \rho \xleftarrow{\text{prev}} \pi \xrightarrow{\text{next}} \eta \xleftarrow{\text{dll}(\pi)} \]
Types for deciding where to unfold

- Types help the analysis decide where to unfold
- Types can be inferred automatically

(see paper)
Summary: Given checkers, everything is automatic

checkers

sorteddll(l, prev, min) =
  if (l = null) then
    true
  else
    l->prev = prev and
    min ≤ l->val and
    sorteddll(l->next, l, l->val)

pre-analysis

materialization and update

abstract interpretation

shape analyzer

Bor-Yuh Evan Chang and Xavier Rival - Relational Inductive Shape Analysis
### Experiments

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Max. Num. Graphs at a Program Point</th>
<th>Max. Num Iterations at a Program Point</th>
<th>Analysis Time (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>doubly-linked list reverse</td>
<td>1</td>
<td>3</td>
<td>1.4</td>
</tr>
<tr>
<td>doubly-linked list copy</td>
<td>2</td>
<td>3</td>
<td>5.3</td>
</tr>
<tr>
<td>doubly-linked list insert</td>
<td>2</td>
<td>4</td>
<td>3.8</td>
</tr>
<tr>
<td>doubly-linked list remove</td>
<td>5</td>
<td>4</td>
<td>6.5</td>
</tr>
<tr>
<td>doubly-linked list remove and back</td>
<td>5</td>
<td>4</td>
<td>6.8</td>
</tr>
<tr>
<td>search tree with parent insert</td>
<td>5</td>
<td>5</td>
<td>8.3</td>
</tr>
<tr>
<td>search tree with parent insert and back</td>
<td>5</td>
<td>5</td>
<td>47.0</td>
</tr>
</tbody>
</table>

Verified shape invariant as given by a checker is preserved across the operation.
Conclusion

• Inductive checkers can form the basis of an effective memory abstraction and analysis
  - Easily extensible on a per-program basis

• To enable materialization anywhere
  - Segments defined as partial checker runs
  - Type pre-analysis on checker definitions to decide where to unfold robustly

• Numerical reasoning via coordination with a base domain (see paper)
What can inductive shape analysis do for you?