Parametric Shape Analysis via 3-Valued Logic

Mooly Sagiv, Thomas Reps, and Reinhard Wilhelm
presented by Morgan Kleene for CMSC631

Shape Analysis

- Goal: prove invariants about heap structures statically
  - Examples:
    - Program vars x and y can never point to the same heap object
    - The data structure that x points to is acyclic
  - Allows certain optimizations
    - Static garbage collection
    - Parallel execution of code; traversals of linked lists that cannot 'interfere'

Why Parametric?

- No advice about what to reason about, but how to do it
- Programmer supplies descriptions of properties and lets the algorithm go to work
- Can be thought of as abstract interpretation for the heap

How we talk about the heap

- Program var X points to U that has an N field pointing to V
- We use first order predicate logic
  - Description is called a logical structure
  - Heap objects are the logical variables
  - Predicates represent program variables

Program var X points to U that has an N field pointing to V

\[
\begin{align*}
X(U) = 1, & \quad X(V) = 0 \\
N(U,V) = 1, & \quad N(V,U) = 0, \\
N(U,U) = 0, & \quad N(V,V) = 0
\end{align*}
\]
How program statements affect structures

- Specify a predicate update formula for statement \( st \)
  - Denoted by \( \varphi_s \)
- New value of a predicate \( P \) is specified by \( P' \)
- Formula with 'X=X->N'  
  - \( N \) predicate should be unchanged so  
    \[ \forall K: N'(K) = N(K) \]  
  - \( X'(K) = \exists V: X(V) \land N(V,K) \)

Evaluation of 'X=X->N' on S

\[
\begin{align*}
U & \quad N & \quad V \\
X(U) &= 0, & X(V) &= 1, & N(V,U) &= 0, & N(U,U) &= 0 \\
N(U,V) &= 1, & N(V,V) &= 0
\end{align*}
\]

The need for 3-valued logic

- Two valued structures describe what is definitely true  
- Imagine a function that takes as a parameter a linked list  
  - Describing linked lists of all possible lengths requires infinite set of definite structures  
- Not very useful for abstract interpretation  
- Goal: find some way to represent infinite set of definite structures

3-valued logic

- Has three truth values 0 = false, \( \frac{1}{2} \) = maybe, 1 = true  
- The truth tables are what one would expect:
3-Valued Logic

- Tool for abstraction – represent many concrete worlds with one formula
  - Suppose it's ½ true that I prepared this presentation assiduously and ½ true that it's going well
  - This represents four concrete situations
    - I prepared and it's going well
    - I prepared and it's not going well
    - I didn't prepare and it's going well
    - I didn't prepare and it's not going well

3-valued logic

- Can encode infinitely many worlds into one 3-valued formula
- Suppose it might be the case that X represents a single object
  - Then we have represented infinitely many concrete worlds
  - One where X is a single object, one where X represents two, one where X represents 3, etc...

Back to the heap

- Map from concrete (2-valued) world to abstract world: 'Truth-blurring embedding'
- Many concrete variables (heap objects) potentially map to one abstract variable (sets of 'indistinguishable' heap values)
- Abstract variables with more than one pre-image are called summary nodes and are henceforth red
- Introduce a new abstract predicate \( sm \)
  - \( sm(U) = \frac{1}{2} \) indicates U may be a summary node

Example of Truth-Blurring

Concrete Structure S

<table>
<thead>
<tr>
<th>X</th>
<th>Y</th>
<th>T</th>
<th>E</th>
<th>N</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>A</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>B</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>B</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>C</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>C</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>D</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>D</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

X \rightarrow A \rightarrow B \rightarrow C \rightarrow D
Y
Example (cont.)

<table>
<thead>
<tr>
<th>X</th>
<th>Y</th>
<th>T</th>
<th>E</th>
<th>sm</th>
</tr>
</thead>
<tbody>
<tr>
<td>U_A</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>U_BCD</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Things to Notice

- Heap cells mapping to same individual have same values at some subset of unary predicates
  - For us it was all of them
- In some sense indistinguishable
- More predicates increases precision, cost
- Abstract structure – represents all linked lists with length $\geq 1$, plus many more

Extracting Store Properties

- Embedding Thm: 3-valued interpretation of structure S is consistent with formula's two valued interpretation in every store (state of the heap) that S represents
- Evaluate a formula representing a property in 3-valued structure S
  - If we get 0 then nothing in $\gamma(S)$ has the property, opposite for 1
  - If $½$ we have no idea

A Formula

Let $\varphi = N^*(U,U)$ (+ denotes transitive closure and the predicate denotes cells that lie on cycles of N fields)

What is the value of $\varphi(A)$?
- 0, you cannot follow any chain of N relations from A and reach A again
- The value of $\varphi(B)$, however is $½$
  - This means that for any of the concrete heap elements E summarized by B, we may or may not be able to follow a chain of N fields to E
Instrumentation Predicates

- What if we have a function taking an acyclic list as argument?
  - No way to represent acyclicity of param currently
- Introduce new predicates that act like assumptions
- Instrumentation principle: We can extract more information by explicitly storing properties of 2-valued structures in their abstract representations

On Instrumenting Example

- Looks the same, but represents a smaller set of structures
  - A bad example: yes, but illustrates that diagram may not change
  - If we had X-abstracted with $\varphi=1$ for all nodes we would get something different – add edges so all nodes may lie on a cycle

Program Statements in Abstract World

- Simply evaluate predicate update formula on the abstraction we are considering using 3-valued semantics
- The effect of ‘$X=X\rightarrow N$’ on

X-abstracting using $\varphi=0$ on all nodes (we assert that this is an acyclic list)

(only difference: we make a note that no node lies on a cycle)
On Instrumentation in General

- Instrumentation is rather like programmer assertions
- Instrumentation updates supplied separately
- Sound (I think)
  - If instruments imply one (definite) thing and core predicates another we represent no structures
- Ensure conservatism by maintaining ‘correct instrumentation’ – meaning of p is same if we change p with statement or evaluate p in new structure after statement changes environment

How do we use this?

- Lattices, or something like them
  - Least fixed point computation – use SSA
  - Bounded structures
    - If we have all individuals differ on some abstraction predicate there is an upper bound the size of the structure
    - Since abstraction causes us to map to structures with the same or fewer heap elements, we terminate
  - Can be obtained by embedding we saw earlier, generalized to three-valued structures, but ignoring ½ values

Embeddings

- Information order: $0 \leq 1/2, 1 \leq 1/2$
- $S$ embeds into $S'$ if there exists an $f$ such that for all predicates $p$, $p(u_1,\ldots,u_n) \leq p(f(u_1),\ldots,f(u_n))$
  - These structures will have different heap elements, in general
- Intuitively, can map to individuals and arrive at a structure so $S'$ is consistent with all properties of $S$
- Embedding theorem applies here as well

Refinements

- **Focus**
  - It can’t be that a formula is true and false
  - Given a set $S$ of formulae, focus creates a set of new structures such that all formulae in $S$ take on 0 or 1
  - But may be infinite
  - Produce a finite, conservative approximation
  - Amounts to stepping through structure and considering why a formula evaluates to $1/2$
**Focus in Action**

Focus on \( \varphi = \exists V: Y(U) \land N(U,V) \), creates three structures

\( \varphi = 0 \), for all nodes

\( p=0 \) \( N \) \( \varphi=1 \)

\( p=1 \) \( N \) \( \varphi=1 \) \( \varphi=0 \)

**Refinements (cont.)**

- **Coerce**
  - Finding best abstraction to work with
  - Based on observation that a formula stored in \( S' \) should be at least as precise as what we can infer from \( S' \)
- Two operations greatly increase accuracy

**Main Contributions**

- Use of abstract interpretation to analyze the heap
- Generalized previous work
  - Creation of new nodes that occurs in *focus* also occurs in other shape analysis work, but action of *focus* appears to be more natural
- Parametric framework