Dynamic Inference of Abstract Types
Abstract Types

• Even declared types only capture a portion of programmer intent for use of variable values
• Want finer-grained types – these will be the abstract types
• Find them by noticing what values interact with each other (by parameterizable definitions of interact). If they interact, they have the same abstract type
To be clear...

- **Value**: a concrete instance of an entity that a program operates on
- **Variable**: a container for a value, may hold different values over its lifetime.

- Paper introduces a method to find abstract types dynamically using values, rather than statically using variables
Example

1. int totalCost(int a, int b, int c, int d) {
2. 
3.   if ((a > 1000) && (d > 2000)) {
4.     int e = 10;
5.     return b + c + e;
6.   } else {
7.     return a + b;
8.   }
9. }

One dynamic solution...

- \( \text{totalCost}(3000, 50, 3, 2006) \)

<table>
<thead>
<tr>
<th></th>
<th>a: 3000</th>
<th>b: 50</th>
<th>c: 3</th>
<th>d: 2006</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>a: 3000 1000</td>
<td>b: 50</td>
<td>c: 3</td>
<td>d: 2006 2000</td>
</tr>
<tr>
<td>4</td>
<td>a: 3000 1000</td>
<td>e: 10</td>
<td>b: 50</td>
<td>c: 3</td>
</tr>
<tr>
<td>5</td>
<td>a: 3000 1000</td>
<td>e: 10</td>
<td>b: 50</td>
<td>c: 3</td>
</tr>
</tbody>
</table>

Precise results: group in abstract types only variables that could actually interact in execution. When would variable a not be in its own abstract type?
Interactions:

• **Dataflow**: nothing counts as interaction between values, every value is a unique abstract type. Why is this interesting?
• **Dataflow & comparisons**: operands to a comparison operator interact.
• **Units**: Add in addition and subtraction to count as interactions. Variables with same abstract type could be assigned same scientific units.
• **Arithmetic**: Add all arithmetic and bitwise operators are interactions. Shift operations are interactions between thing being shifted and result (not shift amount).
• **Logical operators**?
Dynamic Value method:

- Every time a value is created, a unique tag is associated with it, and it’s initially in its own set.
- A global union-find data structure (value UF) groups tags into interaction sets.
- Whenever two values interact (by whatever definition of interact), the sets they belong to get combined into the same interaction set.
From Values to Variables

• Variables will be in the same abstract type if they held values from the same interaction set. Compute these separately at certain program points (a site).

• Two approaches: simple and complex.
  – Simple: look at site at moment of execution, if two variables have values from the same value-interaction set, combine the variables into the same abstract set.
  – Complex: similar, but keep track of per-site value interaction sets, augmenting them every time you visit a site. Do one more pass at the end of execution – now variable abstract type sets are independent of the order of value-interactions.
Results

- Two implementations, one for binary-compiled exe’s like C and C++, one for JVM-compiled class files (Java).
- Abstract types produced were nearly identical to those produced manually on a small program.
- User studies produced results that were beneficial to the users of the tool, with users indicating that use of the tool would have saved them significant time in their tasks.
- Using this as a pre-processing step to Daikon resulted in faster runtimes with less spurious invariants reported.
Closing thoughts

• Compared to static? In general, better (feel from paper).

• Problem with good inputs for dynamic generation? Not really, even small and trivial inputs produced fairly good type abstractions.