CS711 Advanced Programming Languages

Inter-Procedural Analysis

Radu Rugina

1 Sep 2005
Inter-Procedural Analysis

• Standard intra-procedural dataflow analysis:
  – Build flow graph, propagate dataflow facts
  – Assumes no procedure calls
  – Or uses worst-case assumptions about procedure calls

• Inter-procedural analysis
  – Analyze procedure interactions more precisely
  – Difficult to do it efficiently and precisely
The problem

- Transfer function of call = analysis of callee’s body
- Two quick ‘solutions” to this problem
Quick Solution 1: Inlining

• Inline callees into callers
  – End up with one big procedure
  – CFGs of individual procedures = duplicated many times

• **Good**: it is precise
  – distinguishes between different calls to the same function

• **Bad**: exponential blow-up, not efficient

```c
main() { f(); f(); }
f() { g(); g(); }
g() { h(); h(); }
h() { ... }
```

• **Bad**: doesn’t work with recursion
Quick Solution 2: Extend CFG

• Build a “supergraph” = inter-procedural CFG

• Replace each call from P to Q
  – An edge from point before the call (call point) to Q’s entry point
  – An edge from Q’s exit point to the point after the call (return pt)
  – If necessary, add assignments of actuals to formals, and assignment of return value

• **Good**: efficient
  – Graph of each function included exactly once in the supergraph
  – Works for recursive functions (although local variables need additional treatment)

• **Bad**: imprecise, “context-insensitive”
  – The “unrealizable paths problem”: dataflow facts can propagate along infeasible control paths
Unrealizable Paths

\[ \text{read}(x) \]
\[ \text{call } Q() \]
\[ \text{print}(y) \]

\[ y = x \]

\[ \text{call } Q() \]

\[ \text{x = z} \]
\[ \text{call } Q() \]
\[ \text{print}(1) \]
Unrealizable Paths

```
Q()
y = x
call Q()

R()
x = z
call Q()
print(1)
```

```
P()
read(x)
call Q()
print(y)
```
DFA Review

• CFG with nodes $n \in N$
• Dataflow facts: $d \in L$ (lattice)
• Transfer function: $\llbracket n \rrbracket : L \to L$
• MFP (maximal fixed point) solution = greatest solution of:
  \[ X(n) = d_0, \text{ if } n = \text{entry} \]
  \[ X(n) = \bigcap \{ \llbracket m \rrbracket X(m) \mid m \in \text{preds}(n) \} \]
• MOP (meet-over-paths) solution:
  \[ \text{MOP}(n) = \bigcap \{ (\llbracket p_k \rrbracket \circ \ldots \circ \llbracket p_1 \rrbracket \circ \llbracket p_0 \rrbracket) (d_0) \mid \]
  \[ p_0 p_1 \ldots p_k \text{ is a path to } n \} \]
• Safe: $\text{MOP} \subseteq \text{MFP}$
• Precise if transfer functions are distributive: $\text{MOP} = \text{MFP}$
Inter-Procedural DFA

• Consider the supergraph

• Additionally, for each call \( i \):
  – label call \( \rightarrow \) entry edge with \( (i \)
  – label exit \( \rightarrow \) return edge with \( )_i \)

• Consider only valid paths through the supergraph:
  \[
  \text{matched} ::= \text{matched } (i \text{ matched } )_i \mid \varepsilon \\
  \text{valid} ::= \text{valid } (i \text{ matched } \mid \text{ matched }
  \]

• \( \text{MOVP} = \text{meet-over-valid-paths} \)
  \[
  \text{MOVP}(n) = \cap \{ (\llbracket p_k \rrbracket \circ \ldots \circ \llbracket p_1 \rrbracket \circ \llbracket p_0 \rrbracket) (d_0) \mid
  
  \begin{aligned}
  p_0 p_1 \ldots p_k & \text{ is a valid path to } n \\
  \end{aligned}
  \]

CS711  Inter-Procedural Analysis  9
Valid Paths

P(())
  read(x)
  call Q()
  print(y)

Q(())
  y = x

R(())
  x = z
  call Q()
  print(1)
IFDS Problems

• Finite subset, distributive problems:
  – Lattice: \( L = 2^D \) for some finite set \( D \)
  – Partial order is \( \subseteq \), meet is \( \cup \)
  – Transfer functions are distributive

• A precise, efficient solution to IPA for such dataflow problems
  1: an encoding of transfer functions
  2: a formulation of the problem using CFL reachability
  3: an efficient CFL reachability algorithm for the matched parentheses grammar
Transfer Function Encoding

- Enumerate all input space and output space
- Represent functions as graphs with \(2(D+1)\) nodes
- Use a special symbol “0” to describe empty sets
- Example: \(D = \{ a, b, c \}\)

\[
f(S) = (S - \{ a \}) \cup \{ b \}
\]

\[
\begin{array}{cccc}
0 & a & b & c \\
\bullet & & & \\
\end{array}
\]

\[
\begin{array}{cccc}
0 & a & b & c \\
\bullet & & & \\
\end{array}
\]
Exploded Supergraph

• Exploded supergraph:
  – Start with supergraph
  – Replace each node by its graph representation
  – Add edges between corresponding elements in D at consecutive program points

• CFL reachability:
  – Finding MOVP solution is equivalent to computing CFL reachability over the exploded supergraph using the valid parentheses grammar.
The Tabulation Algorithm

- Worklist algorithm, start from entry of “main”
- Keep track of:
  - Path edges: matched paren paths from procedure entry
  - Summary edges: matched paren call-return paths
- At each instruction:
  - Propagate facts using transfer functions; extend path edges
- At each call:
  - Propagate to procedure entry, start with an empty path
  - If a summary for that entry exits, use it
- At each exit:
  - Store paths from corresponding call points as summary paths
  - When a new summary is added, propagate to the return node
Complexity

• Polynomial-time complexity
  – Recall that inlining is exponential

• Inter-procedural: $O(ED^3)$
  – $E =$ number of edges
  – $D =$ size of the dataflow set

• Locally-separable (bit-vector): $O(ED)$
Experiments

<table>
<thead>
<tr>
<th>Example</th>
<th>Tabulation Algorithm (realizable paths)</th>
<th>Naive Algorithm (any path)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Time (sec.)</td>
<td>Reported uses of possibly uninitialized variables</td>
</tr>
<tr>
<td>struct-beauty</td>
<td>4.83+0.75</td>
<td>543</td>
</tr>
<tr>
<td>C-parser</td>
<td>0.70+0.19</td>
<td>11</td>
</tr>
<tr>
<td>ratfor</td>
<td>3.15+0.58</td>
<td>894</td>
</tr>
<tr>
<td>twig</td>
<td>5.45+1.20</td>
<td>767</td>
</tr>
</tbody>
</table>