CSC2/455 Software Analysis and Improvement
Interprocedural Analyses - II

Sreepathi Pai
Mar 25, 2020
URCS
Outline

Interprocedural Analyses

Region-based Analysis Framework

Interprocedural Points-to Analysis

Postscript
Interprocedural Analyses

Region-based Analysis Framework

Interprocedural Points-to Analysis

Postscript
Cloning-based Context-Sensitive Analysis

```c
for(i = 0; i < n; i++) {
    c1: t1 = f1(0);
    c2: t2 = f2(243);
    c3: t3 = f3(243);
    X[i] = t1 + t2 + t3;
}

int f1(int v) {
    return (v+1);
}

int f2(int v) {
    return (v+1);
}

int f3(int v) {
    return (v+1);
}
```

- Create a clone for each unique calling context and then apply context-insensitive analysis
- Is this the same as inlining?
  - See textbook for a differentiating example
The CFG on the left does not distinguish context, the one on the right does.
for(i = 0; i < n; i++) {
    c1: t1 = g(0);
    c2: t2 = g(243);
    c3: t3 = g(243);
    X[i] = t1 + t2 + t3;
}

int g(int v) {
    if(v > 1)
        return f(v);
    else
        return (v+1);
}

int f(int v) {
    return (v+2);
}

To what depth shall we clone functions?
A function call may be distinguished by its context
  - Calling functions or
  - Call-sites (i.e. call stack)

If we do not distinguish contexts,
  - context-insensitive
  - $k = 0$

Different values of $k$ may yield different precision

No value of $k$ may be sufficient
  - recursive function calls
  - indirect function calls
Some numbers

- If there are $N$ functions in a program, how many calling contexts are possible
  - if no recursion is involved?
  - if recursion is involved?
Handling Recursion in Contexts

- Consider nodes in a call graph
  - non-recursive functions
  - self-recursive functions
  - mutually recursive functions
- Look for strongly-connected components
  - trivial (non-recursive)
  - non-trivial (the latter two)
Methods to “finitize” Recursion

- Model them using regular expressions
  - \( f(g \ h \ i)^*j \)
- Eliminate all call information within SCC
  - \( f \ g \ j \)
• Cloning-based analysis
  • Clone functions, once per context
  • Followed by context-insensitive analysis
• Summary-based analysis
  • (Bottom-up phase) Compute summaries of each function for an analysis (e.g. constant propagation) in terms of input parameters
  • (Top-down phase) Pass inputs to summaries, one per context OR merge contexts using meet operator
• Based on Region-based analysis
Outline

Interprocedural Analyses

Region-based Analysis Framework

Interprocedural Points-to Analysis

Postscript
• Operates on *regions* of the control flow graph
• A region is defined (informally) as a portion of code with a single entry and single exit
  • Basic blocks are regions
• Recall we need to iterate (in iterative data flow analysis, IDFA) because of loops
• Can we get rid of loops in some way?
A region is a subset $N$ of the nodes, and $E$ of the edges of a (control) flow graph such that:

- There is a header node $h$ that dominates all nodes in $N$
- If there is a path from $m$ to $n$ that does not go through $h$, then $m \in N$
- $E$ is the set of edges that connect two nodes $n_1$ and $n_2$ in $N$
  - edges into $h$ from outside the region are not part of $E$

Additionally, if the flow graph is reducible, we can organize the regions into a hierarchy.
Reducible Graphs

The T1–T2 definition of reducible graphs:

- **T1**: Remove all self edges on a node
- **T2**: If a node $n$ has a single predecessor $m$, combine them into a single node $x$. Edges into $m$ and out of $n$ are connected to $x$ instead.
- Repeat until neither T1 nor T2 can be applied

A graph is a reducible if at the end of the above procedure the entire graph is reduced to a single node.
Example: Repeated applications of T2
Example: Application of T1 and T2

\[ B_0 \]

\[ B_0 + B_1 + B_2 + B_3 + B_4 \]
Non-reducible (or Irreducible) graphs

- Structured code usually produces reducible graphs
- Can you construct an irreducible graph?
- Textbook details some ways of transforming irreducible graphs into reducible graphs
The smallest regions form leaf regions
  - Basic blocks are leaf regions

Using a process similar to T1/T2 we combine regions into bigger regions

Until we obtain a single large region

The largest region (i.e. final node) has no loops, and if we could construct an appropriate transfer function, we could analyze this region just as we analyze a basic block.
Basic ideas

- If the region consists of a “linear” sequence of basic blocks
  - Say $B_1$ followed by $B_2$, with transfer functions $f_1$ and $f_2$ respectively
  - We need to construct the composition $f_2 \circ f_1$
  - This can be extended to regions, i.e. if we have a linear sequence of regions
- If you encounter alternate paths (akin to join nodes)
  - Apply the meet operator on the transfer functions (not the values!)
  - i.e. $(f_1 \land f_2)(x)$, which is defined as $f_1(x) \land f_2(x)$
  - Note the second $\land$ is the meet operator on data-flow values
Recall that reaching definitions has a gen, kill form for its transfer functions

\[ f_b(x) = gen_b \cup (x - kill_b) \]

Here:

\[ f_1(x) = \{d_1, d_2\} \cup (x - \emptyset) \]
\[ f_2(x) = \{d_3\} \cup (x - \{d_1\}) \]

The composed function is:

\[ (f_2 \circ f_1)(x) = \{d_2, d_3\} \cup (x - \{d_1\}) \]

Which is also in gen–kill form
For gen–kill form

- Composition for gen–kill form is then
  - $kill_\circ$: Union of all kill sets
  - $gen_\circ$: Union of all gen sets - $kill_\circ$

- $f_\circ(x) = gen_\circ \cup (x - kill_\circ)$
Meet for Reaching Definitions

- Merging $B_0$ and $B_1$, we would get:
  - $f_{B_0}(x) = \{d_1, d_2\} \cup (x - \emptyset)$
  - $f_{B_1}(x) = \{d_3\} \cup (x - \{d_1\})$

- Recall that $\land$ for reaching definitions is $\cup$

- $(f_{B_0} \land f_{B_1})(x) = f_{B_0}(x) \cup f_{B_1}(x)$

- $(f_{B_0} \land f_{B_1})(x) = \{d_1, d_2, d_3\} \cup (x - \emptyset)$
  - $\text{gen}_\land = \text{gen}_{B_0} \cup \text{gen}_{B_1}$
  - $\text{kill}_\land = \text{kill}_{B_0} \cap \text{kill}_{B_1} = \emptyset$

- $f_\land(x) = \text{gen}_\land \cup (x - \text{kill}_\land)$
Loop regions for reaching definitions

- Loop region \((L)\) is BH, B1, and B2
- If \(L\) is not executed:
  - \(f^0_L(x) = x\)
- If \(L\) is executed once?
  - BH → B1 → B2 → BH
    (ignore edge from B0 to BH)
  - \(f^1_L(x) = \{d3, d4\} \cup (x - \{d1, d2\})\)
- If \(L\) is executed twice?
  - \(f^2_L(x) = f_L(f_L(x))\)
  - \(f^2_L(x) = \{d3, d4\} \cup (x - \{d1, d2\})\)
• Loop region \((L)\) is BH, B1, and B2

• We have:
  - \(f^0_L(x) = x\)
  - \(f^1_L(x) = \{d3, d4\} \cup (x - \{d1, d2\})\)
  - \(f^2_L(x) = f_L(f_L(x))\)
  - \(f^n_L(x) = \{d3, d4\} \cup (x - \{d1, d2\})\)

• The gen set for a loop is simply the gen set of its body, and likewise for its kill set
Dealing with loop regions

- If the region consists of a loop,
  - Compose the transfer functions for the body, obtaining $f_{body}$
  - Compute the effect of one iteration (or one cycle), $f$
  - Compute the closure of $f$, denoted $f^*$
  - $f^*$ is the transfer function of the loop region

- $f^* = \bigwedge_{n \geq 0} f^n$
  - $f^n$ is $f$ applied to itself $n$ times
  - $f^0$ is loop does not execute, so identity

- Informally:
  - Compute the transfer function of not going into the loop (essentially, identity), meet it with
  - Compute the transfer function of executing the loop once, and meet it with
  - the transfer function of executing the loop twice, and meet it with
  - ...
Loop regions for Reaching Definitions

- $f^* = f^0 \land f^1 \land f^2 \ldots$
- $f^* = x \cup (\text{gen} \cup (x - \text{kill})) \cup (\text{gen} \cup (x - \text{kill})) \ldots$
- $f^* = x \cup (\text{gen} \cup (x - \text{kill}))$
- $f^* = x \cup \text{gen} \cup x$
- $f^* = \text{gen} \cup (x - \emptyset)$

For a loop region, in reaching definitions, the transfer function (i.e., the closure) only generates definitions, but doesn’t kill any definition
Why we need reducible graphs

- In reducible graphs:
  - loops are properly nested or are disjoint
- Repeat composition, meet and closure until you obtain the transfer function for the whole region
The Region-based Analysis Framework

- Compute regions of the flow graph
- Compute, in a bottom-up fashion (from innermost region to outermost), the transfer functions for each region
- Compute, in a top-down fashion (from outermost to innermost), the results of the analysis

- Algorithm 9.53 in the Dragon Book
- Work out Example 9.54 in the Dragon book
- Example 12.8 in the textbook uses summary-based analysis for interprocedural constant propagation
Outline

Interprocedural Analyses

Region-based Analysis Framework

Interprocedural Points-to Analysis

Postscript
Recall how we compute and update pointsTo sets from last class...
Flavours

- Flow-sensitive/Flow-insensitive
- Context-insensitive
- Context-sensitive
  - Cloning-based
  - Summary-based
What the textbook describes

- Flow-insensitive
- Context-sensitive
  - With non-trivial SCCs treated as a single node
- Cloning-based

Additionally, the Dragon book formulates the points-to analysis as a (datalog) logical formula to be solved.
What is a potential call graph for `a.n()` from the points-to relationships?
Interprocedural Analyses

Region-based Analysis Framework

Interprocedural Points-to Analysis

Postscript
References

- Chapter 12 of the Dragon Book
  - Region-based analysis is from Chapter 9, Section 9.7
- Paper recommended:
  - Reps et al. "Precise interprocedural dataflow analysis via graph reachability"