Interprocedural Analysis

Wei Le

February 8, 2019

Overview topics

what is a valid interprocedural path over control flow graphs and call graphs?

- context-sensitivity
- realizable paths and CFL reachability problems

how to make interprocedural analysis scalable?

- summary based approach (IFDS)
- demand-driven analysis if have time

Context-sensitivity

context-sensitive: do we distinguish from different callsites of a same function when interprocedural information is needed?
context-insensitive: function p analyzed with merge of dataflow facts from all call sites

See ppt slides from Harvard for examples to understand context sensitivity

What context to consider?

- Challenge: enumerating and copying all the calling context is impossible
- Solution: make a finite number of copies
- Use context information to determine when to share a copy
- Choice of what to use for context will produce different tradeoffs between precision and scalability
- Choice of contexts determines which calls are differentiated
  - Common choice: approximation of call stack
  - object-sensitivity
  - assumption set: what data flow facts hold at the call sites
  - combination of contexts, e.g., combining assumption set with object

Realizable paths

Idea: restrict attention to **realizable paths**: paths that have proper nesting of procedure calls and exits

For each call site i, let's label the call edge "$(_i$" and the return edge "$)_i$"

Define a grammar that represents balanced paren strings

```
matched ::= ∊                        empty string
          | e                        anything not containing parens
          | matched matched
          | (ᵢ matched )ᵢ
```

- Corresponds to matching procedure returns with procedure calls

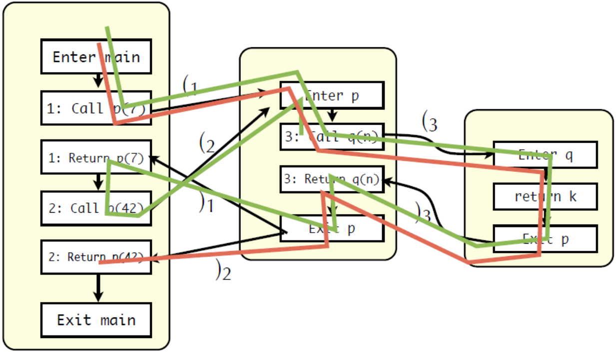
Define grammar of partially balanced parens (calls that have not yet returned)

```
realizable ::= ∊
             | (ᵢ realizable
             | matched realizable
```

Realizable paths

```
main() {
  1: p(7);
  2: x:=p(42);
}

p(int n) {
  3: q(n);
}
q(int k) {
  return k;
}
```



CFL (context-free language) reachability

Many program analysis problems can convert to *CFL reachability problems*.

Let L be a context-free language over alphabet Σ

- Let G be graph with edges labeled from Σ
- Each path in G defines word over Σ
- A path in G is an L-path if its word is in L

CFL reachability problems

Computing *realizable paths*: $O(n^3)$

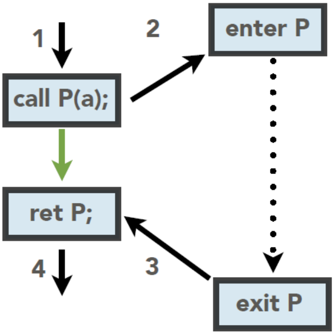
- All-pairs L-path problem: all pairs of nodes n1, n2 such that there is an L-path from n1 to n2
- Single-source L-path problem: all nodes n2 such that there is an L-path from given node n1 to n2
- Single-target L-path problem: all nodes n1 such that there is an L-path from n1 to given node n2
- Single-source single-target L-path problem: is there an L-path from given node n1 to given node n2

MOP vs MRP Solutions

- *MOP*: calculate the dataflow facts that hold at a node in the CFG by taking the meet over all paths
- *MRP*: For a given node n, we want the meet of all realizable paths from the start of the CFG to n
  - May have paths that don't correspond to any execution, but every execution will correspond to a realizable path
  - realizable paths are a subset of all paths
  - MRP sound but more precise: MRP $\subseteq$ MOP

IFDS Key Idea [Reps: 1995]

- Calls & Returns must match
- Enforced by **call** & **ret** nodes
- Track local variables with a call-to-return edge



a polynomial time algorithm without loss of precision

IFDS Definition [Reps: 1995]

IFDS: interprocedural, finite, distributive, subset problem

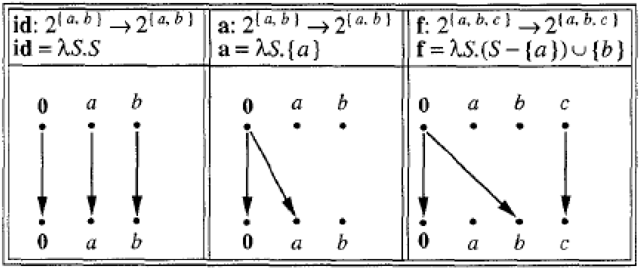
- the set of dataflow facts D is a finite set
- the dataflow functions ($2^D \rightarrow 2^D$) distribute over the meet operator(either union or intersection depending on the problem: f (a$\sqcap$b) = f(a) $\sqcap$ f(b)
- Subset: The dataflow domain is restricted to be a subset domain 2^D,where D is a finite set

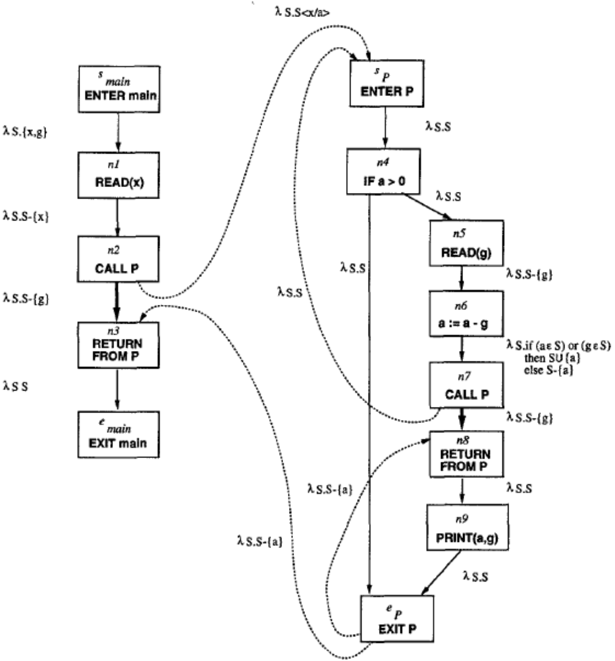
IFDS Details [Reps: 1995]

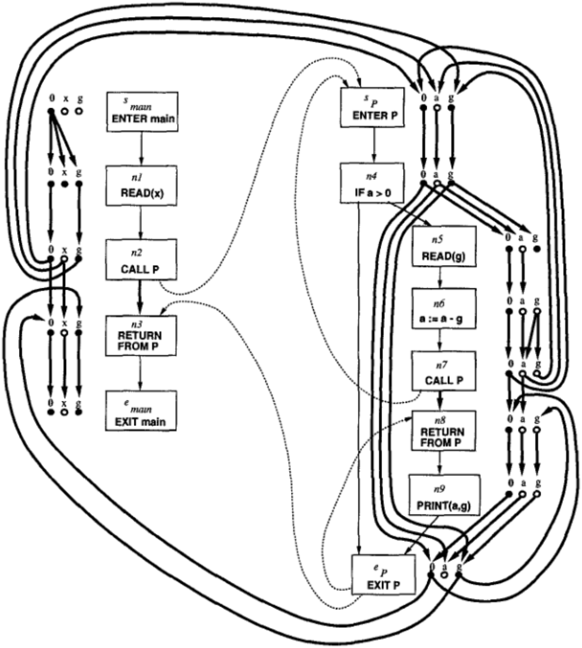
- Constructing Supergraph G*
- Flow function is represented as a graph with 2(D+1) nodes, D is the number of all elements in a dataflow solution, its edge represents a binary relation

Definition 3.1. The ***representation relation of f***, $R_f \subseteq (D \cup \{\mathbf{0}\}) \times (D \cup \{\mathbf{0}\})$, is a binary relation (*i.e.*, graph) defined as follows:

$$
\begin{aligned}
R_f =_{df} \quad & \{ (\mathbf{0}, \mathbf{0}) \} \\
\cup \; & \{ (\mathbf{0}, y) \mid y \in f(\emptyset) \} \\
\cup \; & \{ (x, y) \mid y \in f(\{x\}) \text{ and } y \notin f(\emptyset) \}. \qquad \square
\end{aligned}
$$




λ S.S<x/a>
s main
ENTER main
λ S.{x,g}
n1
READ(x)
λ S.S-{x}
n2
CALL P
λ S.S-{g}
n3
RETURN FROM P
λ S S
e main
EXIT main
s P
ENTER P
λ S.S
n4
IF a > 0
λ S.S
n5
READ(g)
λ S.S-{g}
n6
a := a - g
λ S.if (a ε S) or (g ε S) then SU{a} else S-{a}
n7
CALL P
λ S.S-{g}
n8
RETURN FROM P
λ S.S
n9
PRINT(a,g)
λ S.S
e P
EXIT P
λ S.S
λ S.S
λ S.S-{a}
λ S.S-{a}



s_{main} ENTER main
$n1$ READ(x)
$n2$ CALL P
$n3$ RETURN FROM P
e_{main} EXIT main
s_P ENTER P
$n4$ IF a > 0
$n5$ READ(g)
$n6$ a := a - g
$n7$ CALL P
$n8$ RETURN FROM P
$n9$ PRINT(a,g)
e_P EXIT P
0 x g
0 a g

IFDS Tabulation Algorithm [Reps: 1995]

- keep a worklist of Path Edges (suffixes of valid paths)
- build set of Summary Edges (side effects of a procedure call)
- result = meet over realizable paths

See Ben Greenman's slides for example

Procedural Summaries

When call p is encountered in context C, with input D, check if procedure summary for p in context C exists.

- If not, process p in context C with input D
- If yes, with input D' and output E'
- If D' $\sqsubseteq$ D, then use E'
- if D' $\not\sqsubseteq$ D, then process p in context C with input D'$\sqcap$D (merge the dataflow)
- If output of p in context C changes then may need to reprocess anything that called it
- Need to take care with recursive calls

References and Further Reading

- Precise Interprocedural Dataflow Analysis via Graph Reachability by Tom Reps et al.
- Program Analysis via Graph Reachability by Tom Reps
- IFDS presentation by Ben Greenmen
- Harvard CS252r Lecture Note, Interprocedural Analysis by Stephen Chong
- ESP: path-sensitive program verification in polynomial time