Interprocedural Analysis

Extend intraprocedural analyses to work across calls

+ avoid making conservative assumptions about:
  - effect of callee
  - inputs to procedure
+ no (direct) code increase

– doesn't eliminate direct costs of call
– may not be as effective as inlining at cutting indirect costs

Interprocedural analysis algorithm #1: supergraph

Given call graph and CFG's of procedures,
create single CFG ("control flow supergraph") by

- connecting call sites to entry nodes of callees
- connecting return nodes of callees back to calls

+ simple
+ intraprocedural analysis algorithms work on larger graph
+ decent effectiveness
  (but not as good as inlining)

– speed?
– separate compilation?

Interprocedural analysis algorithm #2: summaries

Compute summary info for each procedure

- can summarize effect/result of calling procedure for callers
- can summarize effect/input of callers for procedure
- can do both

Store summaries in database
Use summaries when compiling & optimizing procedures later

For simple summaries:

+ concise
+ compute, use summaries quickly
+ separate compilation practical

– less precise analysis

Examples of callee summaries

MOD

- the set of variables possibly modified by a call to a proc

USE

- the set of variables possibly read by a call to a proc

MOD-BEFORE-USE

- the set of variables definitely modified before use

Kinds of summaries

Flow-insensitive summaries can be computed without regard to control flow

+ calculated in linear time
– limited kinds of information (e.g. MAY only)

Flow-sensitive summaries must take control flow into account

– may require iterative dfa
+ more precise info possible

Computing summaries

One summary can depend on other summaries

- e.g. summarizing effect of procedure containing calls

Compute summaries bottom-up

What about recursion?
What about program changes?

Interprocedural Constant Propagation

[Callahan, Cooper, Kennedy, & Torczon 86]

Goal: for each procedure, for each formal, identify whether all calls of procedure pass a particular constant to the formal

- e.g. stride argument passed to LINPACK library routines

Sets up lattice-theoretic framework for solving problem

- store const-prop domain element for each formal
  - initialize all formals to T
- worklist-based algorithm to find interprocedural fixpoint:

```
worklist := {Main};
while worklist ≠ ∅ do
  proc := remove_any(worklist);
  process(proc);
end
process(proc) {
  forall call sites c in proc do
    compute c's actuals from proc's formals
    c's callee's formals meet= c's actuals
      if changed, push callee on worklist
}
```


Jump functions

How to quickly compute info at `c`'s actuals from `proc`'s formals?

Define *jump functions* to relate actual parameter at a call site to formal parameters of enclosing procedure

Different degrees of sophistication:

- **all-or-nothing**:
  only if actual is an intraprocedural constant
- **pass-through**:
  also, if formal a constant, then actual a constant
- **symbolic interpretation**:
  do full intraprocedural constant propagation

Define similar jump functions for procedure results, too

- push callers on worklist if procedure's result info changes

Results

"We have not yet tried the system on a large variety of programs, so it would be premature to report any empirical evidence about the value of interprocedural constant propagation."

A contrary position, with experimental data

Richardson & Ganapathi [89]

Studied interprocedural analysis of Pascal programs optimized for RISC uniprocessors

- interprocedural USE, MOD, ALIASES summaries
- found little benefit (< 2%)

Studied simple link-time inlining

- inline after optimization
- respectable benefits (~ 10%)