## 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

Given call graph and CFG's of procedures,
create single CFG 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

Compute summary info for each procedure, independently
Store summaries in database
Use summary at each call site or procedure entry

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

– less precise analysis



## Examples

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

- initialize all formals to T
- worklist-based algorithm to find interprocedural fixpoint:

```
push Main on worklist
while worklist non-empty do
  P := remove_any(worklist);
  process(P);
end
process(P) {
  forall call sites in P do
    update callee's formals by
      meeting with actuals
    if changed, push callee on worklist
  end
}
```



## Jump functions

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

Different degrees of sophistication:

- all-or-nothing: intraprocedural constant
- pass-through: if formal a constant, then actual a constant
- symbolic interpretation: slice of procedure for actual

Define jump functions for procedure results, too

## 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%)