## Summary functions

Idea: generalize a summary of callee into a function

- take info at call site (**calling context**) as argument
- compute info after call site as result

Also called **context-sensitive** or **polyvariant** interprocedural analysis

Total function: handles all possible call site info

- compute once for callee, e.g. bottom-up
- reuse for all callers
- can be expensive to compute/represent precise total function

Partial function: handles only subset of possible call site infos, e.g. those actually occurring in a program

- compute on demand when encountering new call sites, top-down
- can be easier to represent partial functions precisely

Critical question: how to define calling context?



## Interprocedural Constant Propagation

[Callahan, Cooper, Kennedy, & Torczon, PLDI 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

- a kind of total summary function for callers
- 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

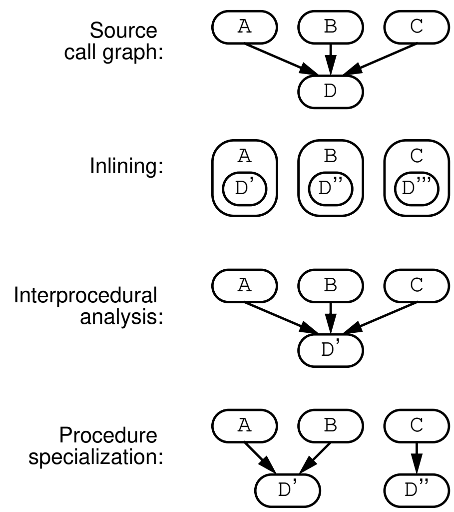
- inline after optimization
- respectable benefits (~ 10%)



## Procedure specialization

A.k.a. procedure cloning, customization

Halfway between inlining and interprocedural analysis, similar to context-sensitive interprocedural analysis






## Abstract process

Given set of call sites of procedure `P`
e.g. $\{c_1, c_2, c_3, c_4, c_5\}$

Partition into equivalence classes of “similar” call sites
e.g. $\{\{c_1, c_2\}, \{c_3, c_4\}, \{c_5\}\}$

Copy `P` for each class, change calls accordingly

Do interprocedural analysis on changed call graph

Versus inlining:
+ less code explosion
+ works in presence of recursion

Versus interprocedural analysis:
+ better optimization of caller and callee

Versus context-sensitive interprocedural analysis:
+ better optimization of callee

Question: how to do partitioning?