Info about finals week

- Final will be given out Monday Dec 10, and will be due Friday Dec 14 at 5pm
  - I will give you last years final, so you can see what kind of questions will be on there
- Final project presentations: Tuesday Dec 11
- Final project report due Thursday Dec 13

From last time

```
main() {         f() {                          g() {
   g();             g();                           if(isLocked()) {
   f();             if (...) { main(); }              unlock;
   lock;         }                                 } else { lock; }
   unlock;                                      }
}
```

```
     main                 f                   g
    ↓    ↑             ↓     ↑             ↓     ↑
    u    ∅             ∅     ∅             ∅     ∅
    "    "             "     "             u     "
    "    "             "     "             "     l
    "    "             l     "             "     "
    "    "             "     "           {u,l}   "
    "    "             "     "             "   {u,l}
    "    "           {u,l} {u,l}           "     "
  {u,l}{u,e}           "     "             "     "
```

What went wrong?

- We merged info from two call sites of g()
- Solution: summaries that keep different contexts separate
- What is a context?

Approach #1 to context-sensitivity

- Keep information for different call sites separate
- In this case: context is the call site from which the procedure is called

Example again

```
main() {         f() {                          g() {
   g();             g();                           if(isLocked()) {
   f();             if (...) { main(); }              unlock;
   lock;         }                                 } else { lock; }
   unlock;                                      }
}
```

```
     main                 f                   g
```

Example again

```
main() {         f() {                          g() {
   g();             g();                           if(isLocked()) {
   f();             if (...) { main(); }              unlock;
   lock;         }                                 } else { lock; }
   unlock;                                      }
}
```

```
     main                 f                   g
```

How should we change the example?

- How should we change our example to break our context sensitivity strategy?

```
main() {
   g();
   f();
   lock;
   unlock;
}

f() {
   g();
   if (...) {
      main();
   }
}

g() {
   if(isLocked()) {
      unlock;
   }
   else {
      lock;
   }
}
```

Answer

```
main() {
   h();
   f();
   lock;
   unlock;
}

f() {
   h();
   if (...) {
      main();
   }
}

h() { g() }

g() {
   if(isLocked()) {
      unlock;
   }
   else {
      lock;
   }
}
```

In general

- Our first attempt was to make the context be the immediate call site
- Previous example shows that we may need 2 levels of the stack
  - the context for an analysis of function f is: call site L_1 where f was called from AND call site L_2 where f's caller was called from
- Can generalize to k levels
  - k-length call strings approach of Sharir and Pnueli
  - Shiver's k-CFA

Approach #2 to context-sensitivity

Approach #2 to context-sensitivity

- Use dataflow information at call site as the context, not the call site itself

Using dataflow info as context

```
main() {
   g();
   f();
   lock;
   unlock;
}

f() {
   g();
   if (...) { main(); }
}

g() {
   if(isLocked()) {
      unlock;
   } else { lock; }
}
```

main

f

g

Transfer functions

- Our pairs of summaries look like functions from input information to output information
- We call these transfer functions
- Complete transfer functions
  - contain entries for all possible incoming dataflow information
- Partial transfer functions
  - contain only some entries, and continually refine during analysis

Top-down vs. bottom-up

- We've always run our interproc analysis top down: from main, down into procs
- For data-based context sensitivity, can also run the analysis bottom-up
  - analyze a proc in all possibly contexts
  - if domain is distributive, only need to analyze singleton sets

Bottom-up example

```
main() {        f() {                       g() {
   g();            g();                        if(isLocked()) {
   f();            if (...) { main(); }           unlock;
   lock;        }                              } else { lock; }
   unlock;                                  }
}
```

main	f	g
⊥	⊥	⊥
"	"	u → l l → u
"	u → l l → u	"
u → u l → e	"	"
"	u → {l,e} l → u	"
u → u l → e		

Top-down vs. bottom-up

- What are the tradeoffs?

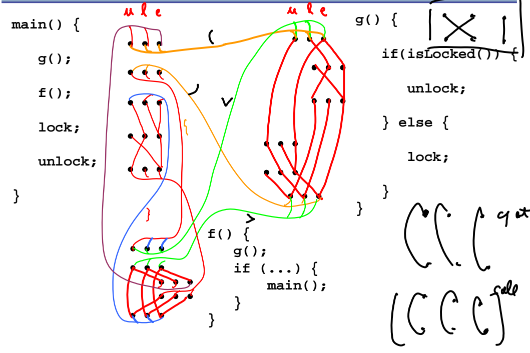
Top-down vs. bottom-up

- What are the tradeoffs?
  - In top-down, only analyze procs in the context that occur during analysis, whereas in bottom-up, may do useless work analyzing proc in a data context never used during analysis
  - However, top-down requires analyzing a given function at several points in time that are far away from each other. If the entire program can't fit in RAM, this will lead to unnecessary swapping. On the other hand, can do bottom-up as one pass over the call-graph, one SCC at a time. Once a proc is analyzed, it never needs to be reloaded in memory.
  - top-down better suited for infinite domains

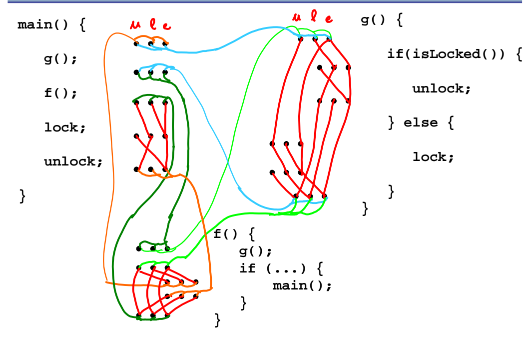
Reps Horwitz and Sagiv 95 (RHS)

- Another approach to context-sensitive interprocedural analysis
- Express the problem as a graph reachability query
- Works for distributive problems

Reps Horwitz and Sagiv 95 (RHS)



Reps Horwitz and Sagiv 95 (RHS)



Procedure specialization

- Interprocedural analysis is great for callers
- But for the callee, information is still merged

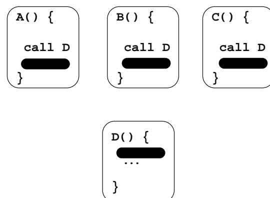
```
main() {
    x := new A(...);
    y := x.g();
    y.f();

    x := new A(...);
    y := x.g();
    y.f();

    x := new B(...);
    y:= x.g();
    y.f();
}

// g too large to inline
g(x) {
   x.f();
   // lots of code
   return x;
}

// but want to inline f
f(x@A) { ... }
f(x@B) { ... }
```

Procedure specialization

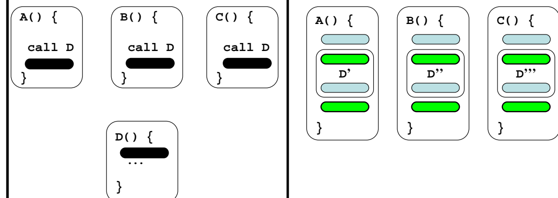
- Specialize g for each dataflow information
- “In between” inlining and context-sensitve interproc

```
main() {
    x := new A(...);
    y := x.g1();
    y.f();

    x := new A(...);
    y := x.g1();
    y.f();

    x := new B(...);
    y:= x.g2();
    y.f();
}

g1(x) {
    x.f(); // can now inline
    // lots of code
    return x;
}
g2(x) {
    x.f(); // can now inline
    // lots of code
    return x;
}

// but want to inline f
f(x@A) { ... }
f(x@B) { ... }
```

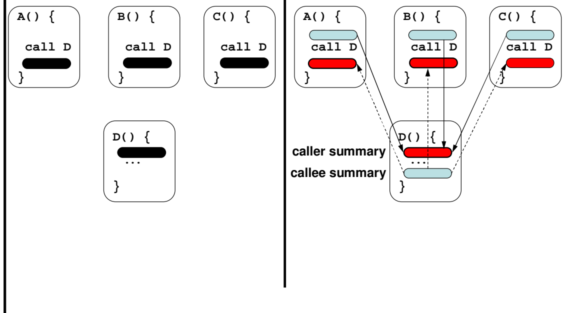
Recap using pictures



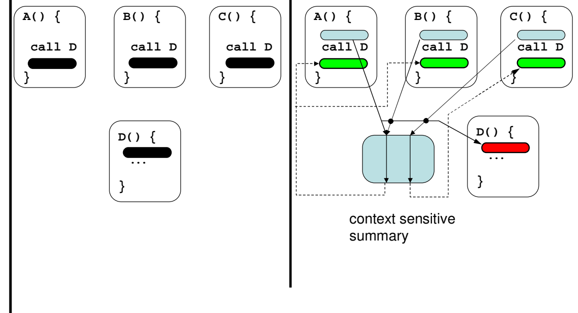
Inlining



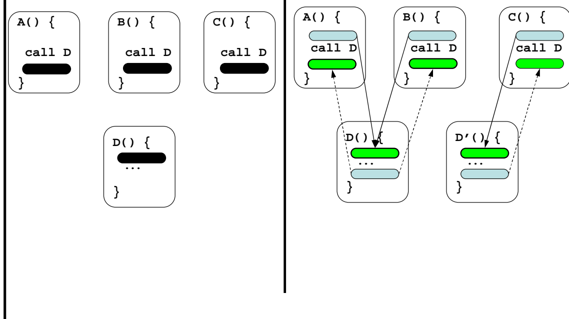
Context-insensitive summary



Context sensitive summary



Procedure Specialization



Comparison

	Caller precision	Callee precision	Code bloat
Inlining			
context-insensitive interproc			
Context sensitive interproc			
Specialization			

Comparison

	Caller precision	Callee precision	code bloat
Inlining	J , because contexts are kept separate	J , because contexts are kept separate	L may be large if we want to get the best precision
context-insensitive interproc	K , because contexts are merged	K , because contexts are merged	J none
Context sensitive interproc	J , because of context sensitive summaries	K , because contexts are still merged when optimizing callees	J none
Specialization	J , contexts are kept separate	J , contexts are kept separate	K Some, less than inlining

Summary on how to optimize function calls

- Inlining
- Tail call optimizations
- Interprocedural analysis using summaries
  - context sensitive
  - context insensitive
- Specialization

Cutting edge research

- Making interprocedural analysis run fast
  - Many of the approaches as we have seen them do not scale to large programs (eg millions of lines of code)
  - Trade off precision for analysis speed
- Optimizing first order function calls
- Making inlining effective in the presence of dynamic dispatching and class loading