Data flow analysis

Abstract Syntax Trees

Control Flow Graph

```
x = a*b;
y = a*b;
while (y > a+b) {
    a = a+1;
x = a*b;
}
```

AST stopped at statement/expression level for brevity
Choosing a representation

- Control flow graph is more general
- AST allows for more efficient algorithms
  - but new programming constructs require changing the algorithm
    - e.g., continue, break, switch, try-catch-finally, goto
    - program transformations may not leave the program in AST form
    - bytecode/machine code isn’t in AST form
    - although you may be able to recover it

Data flow analysis

- A framework for proving facts about a program
  - reasoning about lots of little facts
  - little or no interaction between facts
  - based on all paths through program
    - including infeasible paths
  - e.g., which assignments to \( x \) can be seen at this read of \( x \)?

Reaching definitions

- Each assignment to a variable is a definition
- \( \text{def}(v) \) represents the set of all definitions of \( v \)
- Assume all variables are scalars
  - no pointers or arrays
Gen and Kill

• Gen(S) = facts that are true after S, regardless of the facts true before S
• Kill(S) = facts that aren’t true after S just because they were true before S
  – but might be true after S
• Out(S) = Gen(S) union (In(S) – Kill(S))

Initial conditions

• Out(Entry) needs to be separately defined

• What is appropriate for reaching definitions?

For reaching definitions

• Gen( d: v = exp ) = { d }
• Kill( d: v = exp ) = defs(v)
Computing In(S)

- If S has one predecessor P, In(S) = Out(P)
- Otherwise,
  - In(S) = meet \( p \in \text{Pred}(S) \), Out(P)
- The meet function defines how to combine alternatives
- For reaching definitions, meet = union

Iterative Solution

- For control flow graphs with cycles, can’t directly solve the equations
  - compute final answer for values in terms of other final values already known
- Use iterative solution
  - Can compute dataflow values in any order
    - some orders are more efficient than others
  - Computation will converge to right answer

Initial Value

- For iterative solution
  - might need Out(S) before we get a chance to compute In(S)
- Need an initial value for Out(S) of all statements other than Entry
Control Flow Graph

More control flow programs

Available expressions

An expression $e$ is available at point $p$ if on all paths to $p$, $e$ must have been computed and since that computation, none of the variables in $e$ have been modified — i.e., computation of $e$ here would be redundant.
Backwards problems

- Not all problems are computing from Entry towards exit
- Backwards problems start at Exit, and are computed backwards
- \( \text{In}(S) = \text{Gen}(S) \cup (\text{Out}(S) - \text{Kill}(S)) \)
- \( \text{Out}(S) = \bigcap_{F \in \text{Suc}(S)} \text{In}(F) \)

Backwards problems

- Live variables
  - which variables might be read before they are overwritten or discarded
- Very busy expressions
  - expressions that are guaranteed to be evaluated before any variable used in computing the expression is redefined

Constant Propagation

- Known constant values for variables
Questions

- Does it terminate?
- Does it compute a valid answer?

Definitions

- Meet function: $\sqcap$
- Meet function is commutative and associative
  - $x \sqcap x = x$
- Unique bottom $\bot$ and top $\top$ element
  - $x \sqcap \bot = \bot$
  - $x \sqcap \top = x$

Ordering

- $x \sqsubseteq y$ if and only if $x \sqcap y = x$
- A function $f$ is monotone if for all $x$ and $y$,
  - $x \sqsubseteq y$ implies $f(x) \sqsubseteq f(y)$
Lattice example

```
000
010
100
101
110
111
```

meet is bit-vector logical and

Relating to data flow analysis

- Top is value to initialize non-entry nodes to
  - the identity element for the meet function
- If node function is monotone
  - each re-evaluation of a node moves down the lattice, if it moves at all
- If height of lattice is finite, must terminate

Is it accurate?

- We want the meet over all paths solution

- $\text{MOP}(B) = \text{meet}_{p \in \text{Path}(\text{Entry},B)} f_p(\text{Init})$
  - note that Paths can be infinite if there are loops

- As good as we can do given the framework
- Iterative analysis computes Maximum Fixed Point solution
  - largest solution, ordered by $\sqsubseteq$, that is a fixed point of the iterative computation
  - bottom is also a fixed point, but often not maximal
Is MOP correctly conservative?

- \( \text{MOP}(B) = \bigcap_{p \in \text{Path}(\text{Entry}, B)} f_p(\text{Init}) \)
- Let \( \text{AlmostTruth}(B) = \bigcap_{p \in \text{FeasiblePath}(\text{Entry}, B)} f_p(\text{Init}) \)
- Let \( \text{Bogus}(B) = \bigcap_{p \in \text{InfeasiblePath}(\text{Entry}, B)} f_p(\text{Init}) \)
- \( \text{MOP}(B) = \text{AlmostTruth}(B) \cap \text{Bogus}(B) \)
  - \( \text{MOP}(B) \subseteq \text{AlmostTruth}(B) \)

---

Distributive functions

- \( x \subseteq y \) implies \( f(x) \subseteq f(y) \)
- Monotone implies
  - \( f(x \sqcap y) \subseteq f(x) \sqcap f(y) \)
- \( f \) is distributive if and only if
  - \( f(x \sqcap y) = f(x) \sqcap f(y) \)
  - doing meet early doesn’t cause any reduction in precision

---

Restatement of monotone

- A function \( f \) is monotone if for all \( x \) and \( y \),
  - \( x \subseteq y \) implies \( f(x) \subseteq f(y) \)
- Prove \( f(x \sqcap y) \subseteq f(x) \sqcap f(y) \)
- By definition, \( x \subseteq y \) if and only if \( x \sqcap y = x \)
- Prove \( f(x \sqcap y) \sqcap f(x) \sqcap f(y) = f(x \sqcap y) \)
We know \( x \cap y \subseteq x \) since \( f \) is monotone, \( f(x \cap y) \subseteq f(x) \) which means \( f(x \cap y) \cap f(x) = f(x \cap y) \) and \( f(x \cap y) \cap f(y) = f(x \cap y) \).

\[
f(x \cap y) \cap f(x) \cap f(y) = f(x \cap y) = f(x \cap y)
\]

**MeetOverAllPaths** (\( d_n \)) \( = f_c(f_a(Entry_{out})) \cap f_b(Entry_{out})) \)

**MaximalFixedPoint** (\( d_n \)) \( = f_c(f_a(Entry_{out}) \cap f_b(Entry_{out}))) \)

---

**Distributive problems**

- For a distributive problem
  - you can push transfer functions over meets without causing any reduction in accuracy
- Which problems are distributive?
  - reaching definitions, very busy expressions, live variables, available expressions
- Which are not?
  - most formulations of constant propagation
Constant propagation

Entry

\[ x = 1 \quad x = -1 \]

\[ y = x \times x \]

All Gen/Kill problems are distributive

- If \( \text{Out}_S = \text{Gen}_S \cup \text{In}_S - \text{Kill}_S \)
- Problem is distributive
  - left at exercise for the reader
  - and/or exam question

Are all problems monotone?

- No, you have to be careful
- Consider constant propagation of truth values
  - What is the rule for \( \text{if } x \text{ then } y \text{ else } z \)
Basic Blocks

• When doing dataflow analysis for real
  – don’t iterate through basic block and store
    in/out values for each statement
  – instead, store one in/out value for entire basic
    block
  – compose all of the transfer functions

Order matters

• For forward problems, visit nodes in
  reverse postorder
  – head visited before tail except for back edges
  – expected # of iterations = nesting depth
• For backwards problems
  – compute reverse postorder on reversed graph

ContextInsensitive Analysis

• Analysis we’ve done so far depends on context
  and statement order
  – e.g., S1; S2 ≠ S2; S1
• Difficult to make context sensitive analysis scale
  to 100,000’s of lines of code
  – let alone millions
• Context insensitive analysis simply combines
  information from statements
  – e.g., which variables are modified