COMP 181

Lecture 21

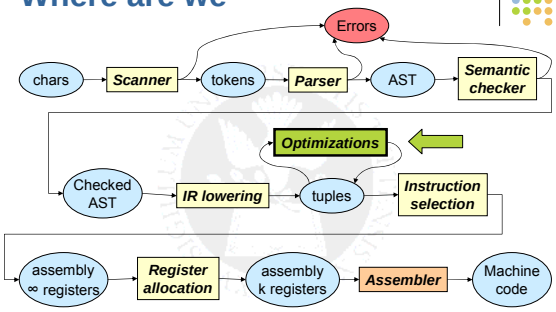
More program analysis

November 21, 2006

Prelude

- What animal is this?
  - Turkey (duh)
- What national honor was almost bestowed on turkeys?
  - National bird
- What annual ritual is going on in this picture?
  - President "pardons" a turkey
  - Origin unknown
  - Last year: "Marshmallow" and "Yam" were pardoned

Where are we



Liveness across instructions

- How is liveness determined?
  - All variables that I uses are live before I
    *Called the **uses** of I*
  - All variables live after I are also live before I, unless I writes to them
    *Called the **defs** of I*
- Mathematically:

$$in[I] = (out[i] - def[I]) \cup use[I]$$

in[I] = {b}
`a = b + 2`

in[I] = {y,z}
`x = 5`
out[I] = {x,y,z}

Flow of information

- Equation:

$$in[I] = (out[i] - def[I]) \cup use[I]$$

- Notice: information flows ***backwards***
  - Need out[] sets to compute in[] sets
  - Propagate information up
- Many problems are ***forward***
  Common sub-expressions, constant propagation, others

Live1
`x = y+1`
Live2
`y = 2*z`
Live3
`if (d)`
Live4

Control flow

- Rule: A variable is live at end of block B if it is live at the beginning of *any* of the successors
  - Characterizes all possible executions
  - ***Conservative***: some paths may not actually happen
- Mathematically:

$$out[B] = \bigcup_{B' \in succ(B)} in[B']$$

- Again: information flows backwards

System of equations

- Put parts together:

$$in[I] = (out[i] - def[I]) \cup use[I]$$
$$out[B] = \bigcup_{B' \in succ(B)} in[B']$$

Often called a system of ***Dataflow Equations***

- Defines a system of equations (or constraints)
  - Consider equation instances for each instruction and each basic block
  - What happens with loops?
    - Circular dependences in the constraints
    - Is that a problem?

Solving the problem

- Iterative solution:
  - Start with empty sets of live variables
  - Iteratively apply constraints
  - Stop when we reach a fixed point

For all instructions $in[I] = out[i] = \varnothing$

Repeat

For each instruction I

$in[I] = (out[i] - def[I]) \cup use[I]$

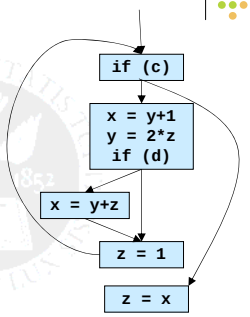
For each basic block B

$out[B] = \bigcup_{B' \in succ(B)} in[B']$

Until no new changes in sets

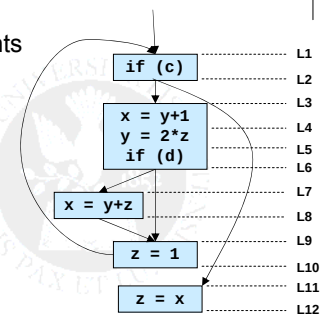
Example

- Steps:
  - Set up live sets for each program point
  - Instantiate equations
  - Solve equations



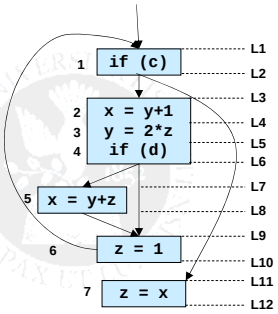
Example

- Program points



Example

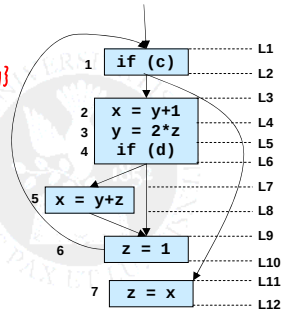
Stmt	Defs	Uses
1		c
2	x	y
3	y	z
4		d
5	x	y, z
6	z	
7	z	x



Example

$$in[I] = (out[i] - def[I]) \cup use[I]$$
$$out[B] = \bigcup_{B' \in succ(B)} in[B']$$

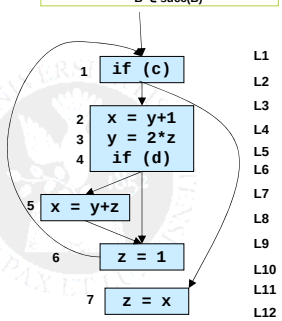
L1 = L2 ∪ {c}
L2 = L3 ∪ L11
L3 = (L4 - {x}) ∪ {y}
L4 =
L5 =
L6 = L7 ∪ L9
L7 =
L8 =
L9 =
L10 =
L11 =
L12 = { }



Example

in[l] = (out[i] – def[l]) ∪ use[l]
out[B] = ∪ in[B']
B' ∈ succ(B)

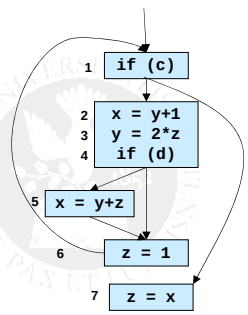
L1 = L2 ∪ {c}
L2 = L3 ∪ L11
L3 = (L4 – {x}) ∪ {y}
L4 = (L5 – {y}) ∪ {z}
L5 = L6 ∪ {d}
L6 = L7 ∪ L9
L7 = (L8 – {x}) ∪ {y,z}
L8 = L9
L9 = L10 – {z}
L10 = L1
L11 = (L12 – {z}) ∪ {x}
L12 = {}



L1
L2
L3
L4
L5
L6
L7
L8
L9
L10
L11
L12

Example

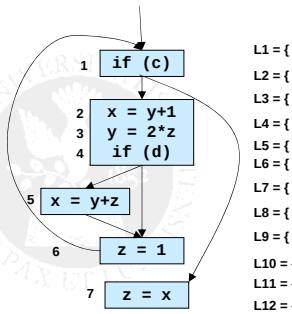
L1 = L2 ∪ {c}
L2 = L3 ∪ L11
L3 = (L4 – {x}) ∪ {y}
L4 = (L5 – {y}) ∪ {z}
L5 = L6 ∪ {d}
L6 = L7 ∪ L9
L7 = (L8 – {x}) ∪ {y,z}
L8 = L9
L9 = L10 – {z}
L10 = L1
L11 = (L12 – {z}) ∪ {x}
L12 = {}



L1 = { x,y,z,d,c }
L2 = { x,y,z,d }
L3 = { y,z,d }
L4 = { z,d }
L5 = { y,z,d }
L6 = { y,z }
L7 = { y,z }
L8 = { }
L9 = { }
L10 = { }
L11 = { x }
L12 = { }

Example

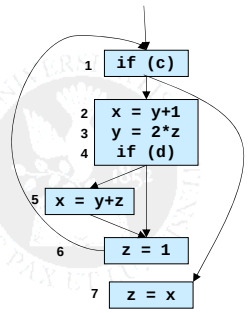
L1 = L2 ∪ {c}
L2 = L3 ∪ L11
L3 = (L4 – {x}) ∪ {y}
L4 = (L5 – {y}) ∪ {z}
L5 = L6 ∪ {d}
L6 = L7 ∪ L9
L7 = (L8 – {x}) ∪ {y,z}
L8 = L9
L9 = L10 – {z}
L10 = L1
L11 = (L12 – {z}) ∪ {x}
L12 = {}



L1 = { }
L2 = { }
L3 = { }
L4 = { }
L5 = { }
L6 = { }
L7 = { }
L8 = { }
L9 = { }
L10 = { }
L11 = { }
L12 = { }

Example

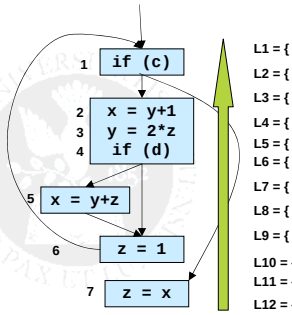
L1 = L2 ∪ {c}
L2 = L3 ∪ L11
L3 = (L4 – {x}) ∪ {y}
L4 = (L5 – {y}) ∪ {z}
L5 = L6 ∪ {d}
L6 = L7 ∪ L9
L7 = (L8 – {x}) ∪ {y,z}
L8 = L9
L9 = L10 – {z}
L10 = L1
L11 = (L12 – {z}) ∪ {x}
L12 = {}



L1 = { x, y, z, c, d }
L2 = { x, y, z, d }
L3 = { y, z, d }
L4 = { z, d }
L5 = { y, z, d }
L6 = { y, z }
L7 = { y, z }
L8 = { }
L9 = { x, y }, c, d
L10 = { x, y, z, c, d }
L11 = { x }
L12 = { }

Example

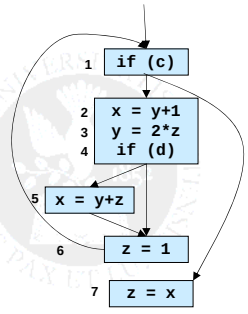
L1 = L2 ∪ {c}
L2 = L3 ∪ L11
L3 = (L4 – {x}) ∪ {y}
L4 = (L5 – {y}) ∪ {z}
L5 = L6 ∪ {d}
L6 = L7 ∪ L9
L7 = (L8 – {x}) ∪ {y,z}
L8 = L9
L9 = L10 – {z}
L10 = L1
L11 = (L12 – {z}) ∪ {x}
L12 = {}



L1 = { x, y, z, c, d }
L2 = { x, y, z, d }
L3 = { y, z, d }
L4 = { z, d }
L5 = { y, z, d }
L6 = { y, z }
L7 = { y, z }
L8 = { }
L9 = { }
L10 = { }
L11 = { x }
L12 = { }

Example

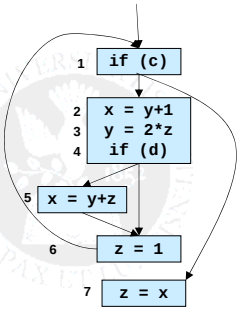
L1 = L2 ∪ {c}
L2 = L3 ∪ L11
L3 = (L4 – {x}) ∪ {y}
L4 = (L5 – {y}) ∪ {z}
L5 = L6 ∪ {d}
L6 = L7 ∪ L9
L7 = (L8 – {x}) ∪ {y,z}
L8 = L9
L9 = L10 – {z}
L10 = L1
L11 = (L12 – {z}) ∪ {x}
L12 = {}



L1 = { x, y, z, c, d }
L2 = { x, y, z, c, d }
L3 = { y, z, c, d }
L4 = { x, z, c, d }
L5 = { x, y, z, c, d }
L6 = { x, y, z, c, d }
L7 = { y, z, c, d }
L8 = { x, y, c, d }
L9 = { x, y, c, d }
L10 = { x, y, z, c, d }
L11 = { x }
L12 = { }

Example

L1 = L2 ∪ {c}
L2 = L3 ∪ L11
L3 = (L4 – {x}) ∪ {y}
L4 = (L5 – {y}) ∪ {z}
L5 = L6 ∪ {d}
L6 = L7 ∪ L9
L7 = (L8 – {x}) ∪ {y,z}
L8 = L9
L9 = L10 – {z}
L10 = L1
L11 = (L12 – {z}) ∪ {x}
L12 = {}



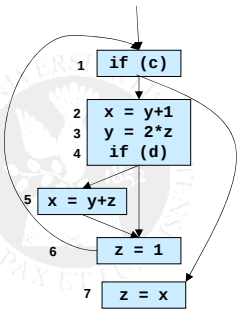
L1 = { x, y, z, c, d }
L2 = { x, y, z, c, d }
L3 = { y, z, c, d }
L4 = { x, z, c, d }
L5 = { x, y, z, c, d }
L6 = { x, y, z, c, d }
L7 = { y, z, c, d }
L8 = { x, y, c, d }
L9 = { x, y, c, d }
L10 = { x, y, z, c, d }
L11 = { x }
L12 = { }

Questions

- Why does this terminate?
- Why does this compute the right answer?
- How could we modify this for dead-code elimination?

Example

L1 = L2 ∪ {c}
L2 = L3 ∪ L11
L3 = (L4 – {x}) ∪ {y}
L4 = (L5 – {y}) ∪ {z}
L5 = L6 ∪ {d}
L6 = L7 ∪ L9
L7 = (L8 – {x}) ∪ {y,z}
L8 = L9
L9 = L10 – {z}
L10 = L1
L11 = (L12 – {z}) ∪ {x}
L12 = {}



L1 = { x, y, z, c, d }
L2 = { x, y, z, c, d }
L3 = { y, z, c, d }
L4 = { x, z, c, d }
L5 = { x, y, z, c, d }
L6 = { x, y, z, c, d }
L7 = { y, z, c, d }
L8 = { x, y, c, d }
L9 = { x, y, c, d }
L10 = { x, y, z, c, d }
L11 = { x }
L12 = { }

Generalization

- Dataflow analysis
  - A common framework for such analysis
  - Computes information at each program point
  - Conservative: characterizes all possible program behaviors
- Methodology
  - Describe the information (e.g., live variable sets) using a structure called a ***lattice***
  - Build a system of equations based on:
    - How each statement affects information
    - How information flows between basic blocks
  - Solve the system of constraints

Lattices

- A lattice is a kind of ***partial order***
  - A set of elements P
  - A partial order relation ⊑
    - Reflexive
    - Anti-symmetric
    - Transitive
  - Called a "partial order" because not all elements are comparable
- Used to represent analysis information

Lower and upper bounds

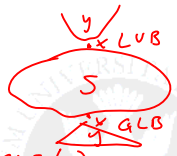
P, ⊑

S ⊆ P

x ∈ P is a lower bound for S if x ⊑ y ∀y ∈ S

x ∈ P is a upper bound y ⊑ x ∀y ∈ S

LUB and GLB



$x \in P$ GLB(S)

x is Lower bound S

$y \sqsubseteq x$ for any lower bound y of S

$x \in P$ LUB

x is an upper bound

$x \sqsubseteq y$ for any upper bound y of S

Lattices

$(L, \sqsubseteq)$

any finite subset S of L has a unique LUB and GLB

two operations on lattice elements

meet $x \sqcap y = GLB(\{x, y\})$

join $x \sqcup y = LUB(\{x, y\})$

Complete lattices

$(L, \sqsubseteq)$

Top $\top$ LUB(L)

Bottom $\bot$ GLB(L)

for all $x \in L$ $x \sqsubseteq \top$

$\bot \sqsubseteq x$

$(L, \sqsubseteq, \sqcap, \top)$

Properties of meet and join

Associative

$(x \sqcap y) \sqcap z = x \sqcap (y \sqcap z)$

Commutative

$x \sqcap y = y \sqcap x$

Idempotent

$x \sqcap x = x$

Example

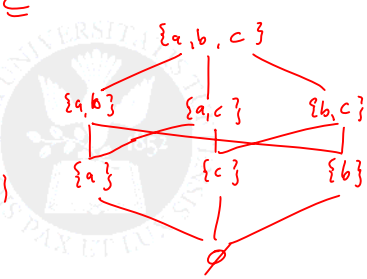
- S = {a, b, c }
- P = power set of S x, y
- Partial order as set inclusion $\sqsubseteq \equiv \subseteq$
  - Reflexive $x \sqsubseteq x$
  - Anti-symmetric $x \sqsubseteq y$ and $y \sqsubseteq x$
  - Transitive
- What about LUB and GLB?

$x \sqcap y = z$ s.t. $x \supseteq z$, $y \supseteq z$ $x \; y \equiv x \cap y$

$x \sqcup y$

Graphically

- Partial order $\subseteq$
- Meet $\cap$
- Join $\cup$
- Top $\{a, b, c\}$
- Bottom $\emptyset$



Reversed Lattice

- Partial order $\supseteq$
- Meet $\cup$
- Join $\cap$
- Top $\emptyset$
- Bottom $\{a,b,c\}$

In dataflow analysis

- Example: live variables

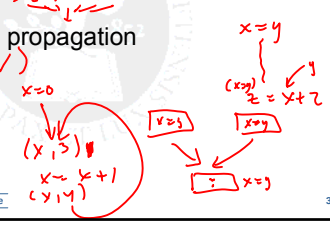
  $P = 2^V$

  $\sqcap = \cup$

- Example: copy propagation

  $P = 2^{(V \times V)}$

  $\sqcap = \cap$



Transfer functions

- Capture the effect of a statement on lattice values
- Forward: out = F(in)
- Backward: in = F(out)
- Monotonicity: for all A, B

  $A \sqsubseteq B \Rightarrow F(A) \sqsubseteq F(B)$

  or $A \sqsubseteq F(A)$

Merge operation

- Combine information on different paths

Next time

- Later today: new programming assignment

  (And yes, you will have two weeks to work on it)
- Memory management and garbage collection
- Happy Thanksgiving!