Data Flow Analysis II

15-817A Model Checking and Abstract Interpretation

HAPPY GROUNDHOG DAY!

Agenda

- Recalling last lecture
- Analysis: Very Busy Expressions
- {**forward**,**backward**}x{**may**,**must**} typology
- Analysis: Live Variables
- Analysis: Available Expressions
- Where do we go from here?

Agenda

- **Recalling last lecture**
- Analysis: Very Busy Expressions
- {**forward**,**backward**}x{**may**,**must**} typology
- Analysis: Live Variables
- Analysis: Available Expressions
- Where do we go from here?

Recall: Last Lecture

- Recall: Feynman-공 Method for data flow analysis
- Recall: Our programming language
- Recall: Control Flow Graphs (CFGs)
- Recall: Reaching Definitions
- Recall: Reaching Definition Analysis

Recall: Last Lecture

- Recall: Semilattice, Complete Lattice
- Recall: Monotone Functions, ACC, and their significance
- Recall: General Algorithm

Today: Cosmetic Changes

Let s be a statement:

- succ(s) = {immediate successor statements of s}
- Pred(s) = {immediate predecessor statements of s}
- In(s) = flow at program point just before executing s
- Out(s) = flow at program point just after executing s
- $In(s) = \bigcap_{s' \in pred(s)} Out(s')$ **(Must)**
- $Out(s) = Gen(s) \cup (In(s) - Kill(s))$ **(Forward)**
- Note these are also called transfer functions

Gen(s) = *set of facts true after s that weren't true before s*

Kill(s) = *set of facts no longer true after s*

Agenda

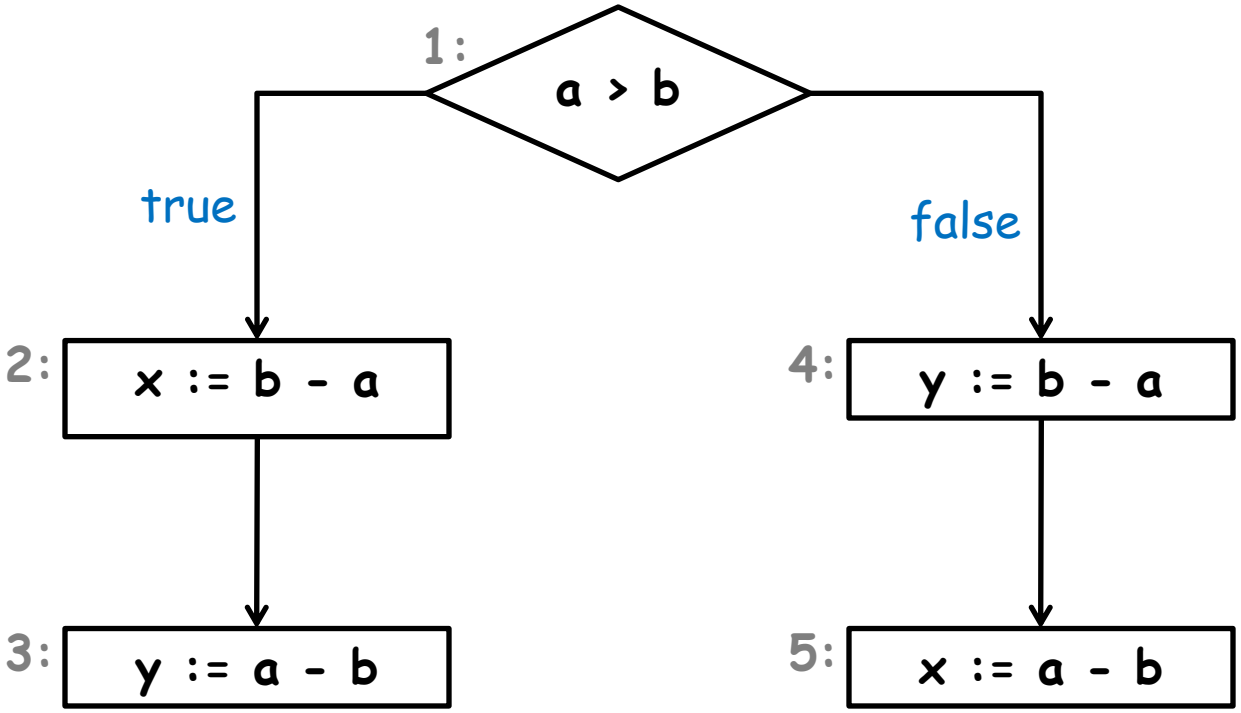
- Recalling last lecture
- **Analysis: Very Busy Expressions**
- {**forward**,**backward**}x{**may**,**must**} typology
- Analysis: Live Variables
- Analysis: Available Expressions
- Where do we go from here?

Very Busy Expressions: Definitions

An expression is *very busy* at the exit from a label if,
no matter what path is taken from the label,
the expression must always be used
before any of the variables occurring in it are redefined.

For each program point, which expressions *must* be very busy
at the exit from the point?

Very Busy Expressions: CFG



Agenda

- Recalling last lecture
- Analysis: Very Busy Expressions
- **{forward,backward}x{may,must} typology**
- Analysis: Live Variables
- Analysis: Available Expressions
- Where do we go from here?

General Iterative Data Flow Analysis Algorithm (Forward)

```
// Boundary Condition
Out[Entry] = V_Entry
// Initialization for iterative algorithm
For each basic block B
        Out[B] = Top

// iterate
while(Changes to any Out[], In[] occur) {
        For each basic block B {
                In[B] = meet(Out[p_0], ... Out[p_1])
                Out[B] = f_B(In[B])
        }
}
```

Forward Data Flow (General Case)

Out(s) = Top for all statements s

W := { all statements } (worklist)

Repeat

  Take s from W

    temp := $f_s(\sqcap_{s' \in pred(s)}$ Out(s')) (f_s monotonic *transfer fn*)

    if (temp != Out(s)) {

      Out(s) := temp

      W := W $\cup$ succ(s)

    }

until W = $\emptyset$

Forward Data Flow (General Case)

Out(s) = Top for all statements s

W := { all statements } (worklist)

Repeat

  Take s from W

    temp := $f_s(\sqcap_{s' \in pred(s)} Out(s'))$ (f_s monotonic *transfer fn*)

    if (temp != Out(s)) {

      Out(s) := temp

      W := W ∪ succ(s)

    }

until W = ∅

Forward vs. Backward

Out(s) = Top for all s
W := { all statements }
repeat
    Take s from W
    temp := $f_s(\sqcap_{s' \in pred(s)}$ Out(s′))
    if (temp != Out(s)) {
      Out(s) := temp
      W := W ∪ succ(s)
    }
until W = ∅

In(s) = Top for all s
W := { all statements }
repeat
    Take s from W
    temp := $f_s(\sqcap_{s' \in succ(s)}$ In(s′))
    if (temp != In(s)) {
      In(s) := temp
      W := W ∪ pred(s)
    }
until W = ∅

Agenda

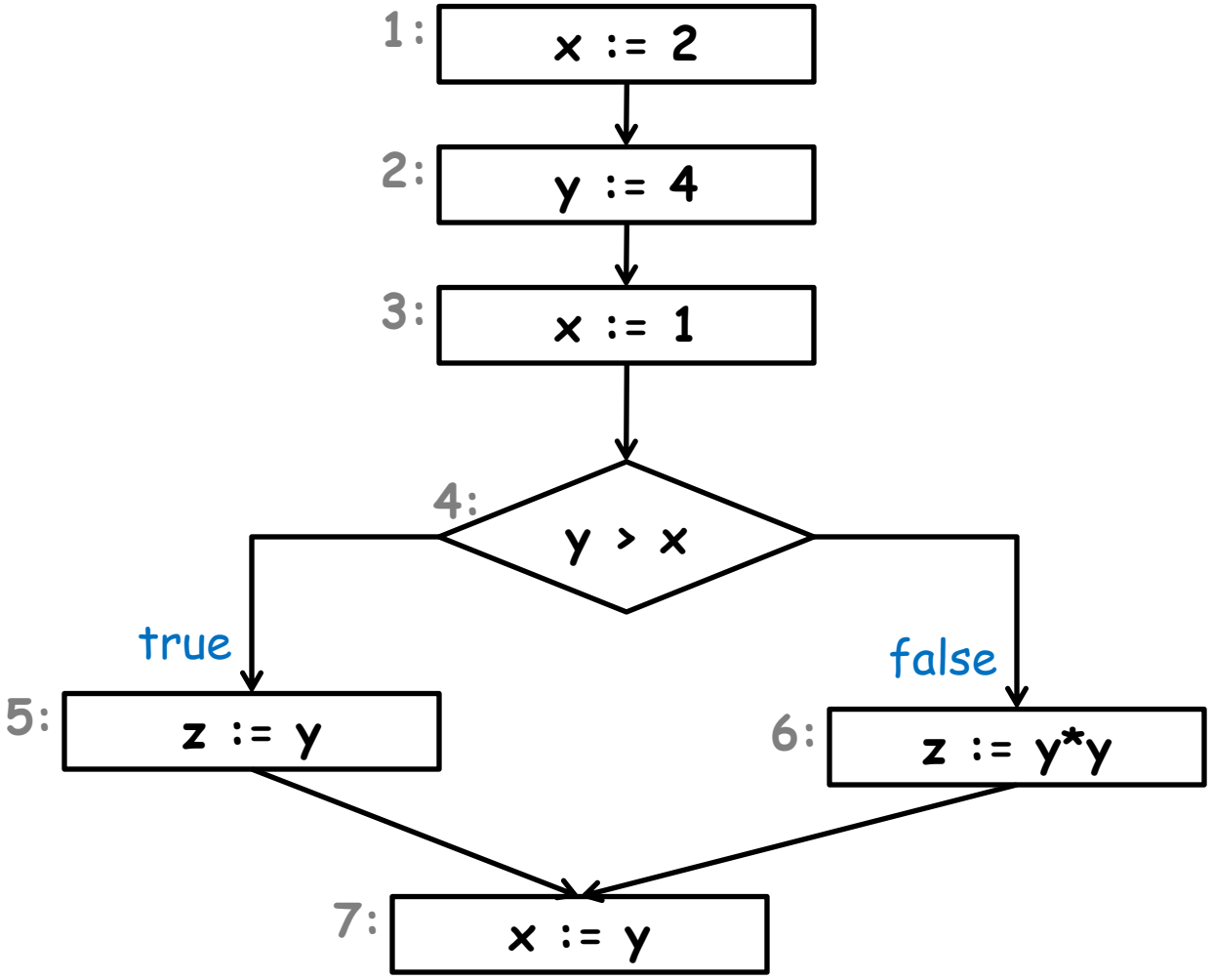
- Recalling last lecture
- Analysis: Very Busy Expressions
- {**forward**,**backward**}x{**may**,**must**} typology
- **Analysis: Live Variables**
- Analysis: Available Expressions
- Where do we go from here?

Live Variables: Definitions

A variable is *live* at the exit from a label if
there exists a path from the label to a use of the variable that does not re-define the variable.

For each program point, which variables *may* be live at the exit from the point?

Live Variables:
CFG



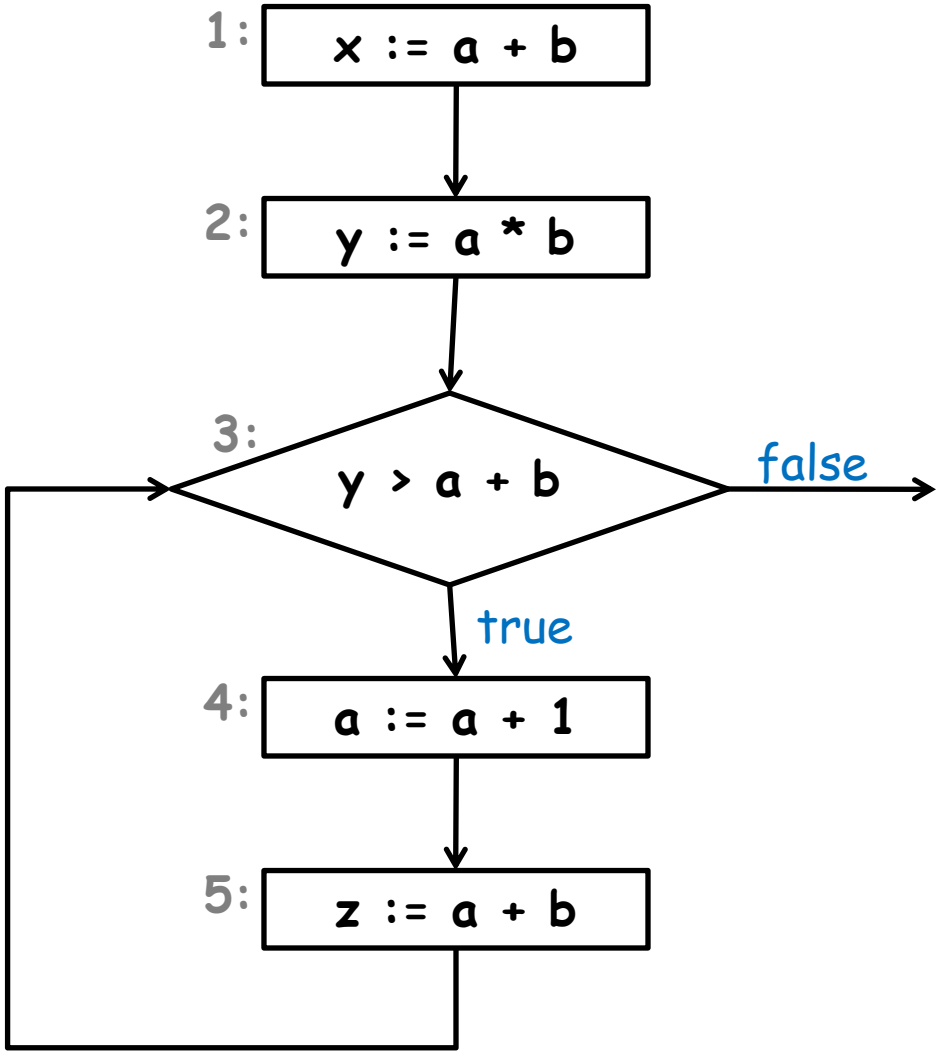
Agenda

- Recalling last lecture
- Analysis: Very Busy Expressions
- {**forward**,**backward**}x{**may**,**must**} typology
- Analysis: Live Variables
- **Analysis: Available Expressions**
- Where do we go from here?

Available Expressions: Definition

For each program point,
which expressions *must* have already been computed,
and not later modified,
on all paths to the program point.

Available Expressions: CFG



Agenda

- Recalling last lecture
- Analysis: Very Busy Expressions
- {**forward**,**backward**}x{**may**,**must**} typology
- Analysis: Live Variables
- Analysis: Available Expressions
- **Where do we go from here?**

Next?

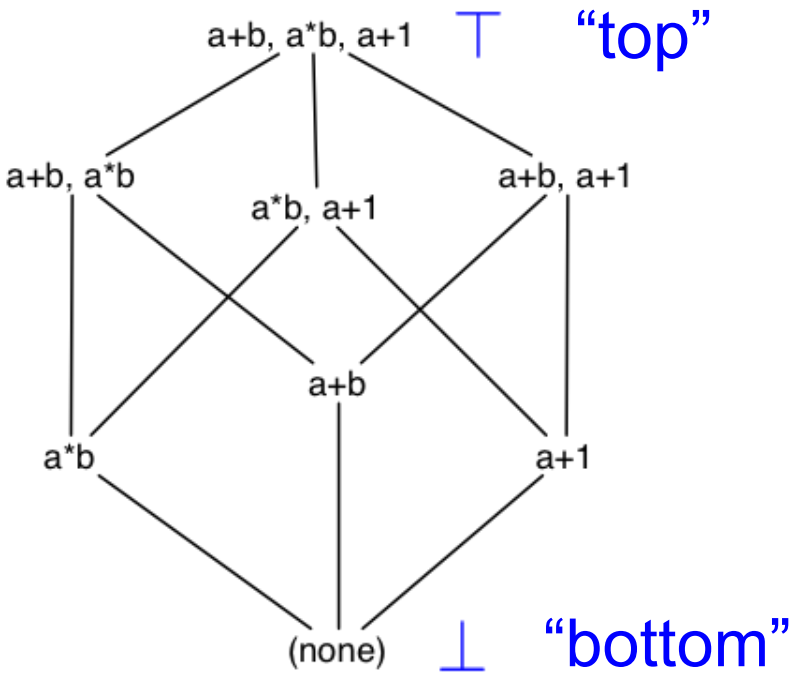
- 2.2: Formal correctness proof (Live Variables)
- Constant Propagation
- 2.5: Interprocedural Analysis
- 2.6: Shape Analysis

(Overflow Slides)

Data Flow Facts and lattices

Typically, data flow facts form a lattice

Example, Available expressions



Partial Orders

- A *partial order* is a pair $(P, \leq)$ such that
  - $\leq \subseteq P \times P$
  - $\leq$ is *reflexive*: $x \leq x$
  - $\leq$ is *anti-symmetric*: $x \leq y$ and $y \leq x$ implies $x = y$
  - $\leq$ is *transitive*: $x \leq y$ and $y \leq z$ implies $x \leq z$

Lattices

- A partial order is a lattice if $\sqcap$ and $\sqcup$ are defined so that
  - $\sqcap$ is the meet or greatest lower bound operation
    - $x \sqcap y \leq x$ and $x \sqcap y \leq y$
    - If $z \leq x$ and $z \leq y$ then $z \leq x \sqcap y$
  - $\sqcup$ is the join or least upper bound operation
    - $x \leq x \sqcup y$ and $y \leq x \sqcup y$
    - If $x \leq z$ and $y \leq z$, then $x \sqcup y \leq z$

Lattices (cont.)

A finite partial order is a lattice if meet and join exist for every pair of elements

A lattice has unique elements bot and top such that

$x \sqcap \bot = \bot \qquad x \sqcup \bot = x$

$x \sqcap \top = x \qquad x \sqcup \top = \top$

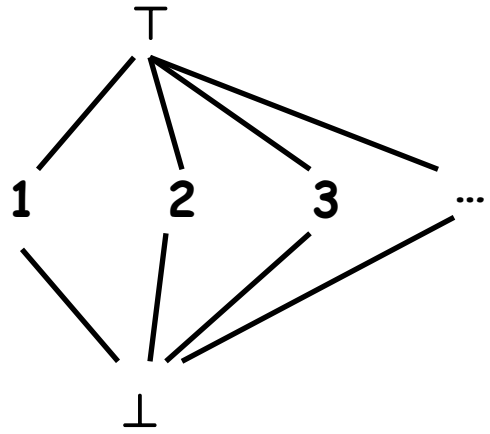
In a lattice

$x \leq y$ iff $x \sqcap y = x$

$x \leq y$ iff $x \sqcup y = y$

Useful Lattices

- $(2^S, \subseteq)$ forms a lattice for any set S.
  - 2^S is the powerset of S (set of all subsets)
- If $(S, \leq)$ is a lattice, so is $(S, \geq)$
  - i.e., lattices can be flipped
- The lattice for constant propagation



Note: order on integers is different from order in lattice

Monotonicity

- A function f on a partial order is monotonic if

$$x \leq y \text{ implies } f(x) \leq f(y)$$

- Easy to check that operations to compute In and Out are monotonic
  - $In(s) = \bigcap_{s' \in \mathbf{pred(s)}} Out(s')$
  - $Temp = Gen(s) \cup (In(s) - Kill(s))$
- Putting the two together
  - $Temp = f_s \left(\bigcap_{s' \in \mathbf{pred(s)}} Out(s')\right)$

Termination -- Intuition

- We know algorithm terminates because
  - The lattice has finite height
  - The operations to compute In and Out are monotonic
  - On every iteration we remove a statement from the worklist and/or move down the lattice.

Lattices (P, ≤)

Available expressions

- P = sets of expressions
- S1 ⊓ S2 = S1 ∩ S2
- Top = set of all expressions

Reaching Definitions

- P = set of definitions (assignment statements)
- S1 ⊓ S2 = S1 ∪ S2
- Top = empty set

Fixpoints -- Intuition

We always start with Top

- Every expression is available, no defns reach this point
- Most optimistic assumption
- Strongest possible hypothesis

Revise as we encounter contradictions

- Always move down in the lattice (with meet)

Result: A greatest fixpoint

Lattices (P, ≤), cont'd

Live variables

- P = sets of variables
- S1 ⊓ S2 = S1 ∪ S2
- Top = empty set

Very busy expressions

- P = set of expressions
- S1 ⊓ S2 = S1 ∩ S2
- Top = set of all expressions

Least vs. Greatest Fixpoints

Dataflow tradition: Start with Top, use meet

- To do this, we need a *meet semilattice with top*
- meet semilattice = meets defined for any set
- Computes greatest fixpoint

Denotational semantics tradition: Start with Bottom, use join

- Computes least fixpoint

Terminology Review

Must vs. May

- (Not always followed in literature)

Forwards vs. Backwards

Flow-sensitive vs. Flow-insensitive

Distributive vs. Non-distributive