Slide 1

Another example: constant prop

- Set D =

```
        in
   X := N
        out
```
$F_{X := N}(in) =$

```
        in
   X := Y op Z
        out
```
$F_{X := Y op Z}(in) =$

1

Slide 2

Another example: constant prop

- Set D = $2^{\{ x \to N \mid x \in Vars \wedge N \in Z \}}$

```
        in
   X := N
        out
```
$F_{X := N}(in) = in - \{ X \to * \} \cup \{ X \to N \}$

```
        in
   X := Y op Z
        out
```
$F_{X := Y op Z}(in) = in - \{ X \to * \} \cup$
$\{ X \to N \mid (Y \to N_1) \in in \wedge$
$(Z \to N_2) \in in \wedge$
$N = N_1 op N_2 \}$

2

Slide 3

Another example: constant prop

```
        in
   X := *Y
        out
```
$F_{X := *Y}(in) =$

```
        in
   *X := Y
        out
```
$F_{*X := Y}(in) =$

3

Slide 4

Another example: constant prop

```
        in
   X := *Y
        out
```
$F_{X := *Y}(in) = in - \{ X \to * \}$
$\cup \{ X \to N \mid \forall Z \in may\text{-}point\text{-}to(Y) .$
$(Z \to N) \in in \}$

```
        in
   *X := Y
        out
```
$F_{*X := Y}(in) = in - \{ Z \to * \mid Z \in may\text{-}point(X) \}$
$\cup \{ Z \to N \mid Z \in must\text{-}point\text{-}to(X) \wedge$
$Y \to N \in in \}$
$\cup \{ Z \to N \mid (Y \to N) \in in \wedge$
$(Z \to N) \in in \}$

4

Slide 5

Another example: constant prop

```
        in
   *X := *Y + *Z
        out
```
$F_{*X := *Y + *Z}(in) =$

```
        in
   X := G(...)
        out
```
$F_{X := G(...)}(in) =$

5

Slide 6

Another example: constant prop

```
        in
   *X := *Y + *Z
        out
```
$F_{*X := *Y + *Z}(in) = F_{a := *Y; b := *Z; c := a + b; *X := c}(in)$

```
        in
   X := G(...)
        out
```
$F_{X := G(...)}(in) = \emptyset$

6

Another example: constant prop


```
        in
s: if (...)
out[0]    out[1]
```

```
in[0]      in[1]
  merge
       out
```
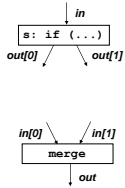
7

Lattice

- $(D, \sqsubseteq, \bot, \top, \sqcup, \sqcap) =$
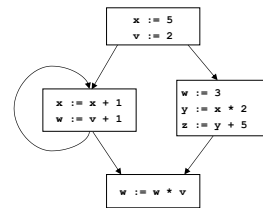
8

Lattice

- $(D, \sqsubseteq, \bot, \top, \sqcup, \sqcap) =$
  $(2^A, \supseteq, A, \emptyset, \cap, \cup)$
  where $A = \{ x \rightarrow N \mid x \in \text{Vars} \land N \in Z \}$
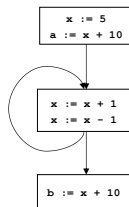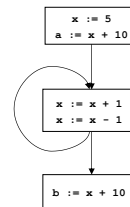
9

Example


```
x := 5
v := 2
```

```
x := x + 1        w := 3
w := v + 1        y := x * 2
                  z := y + 5
```

```
w := w * v
```

10

Another Example


```
x := 5
a := x + 10
```

```
x := x + 1
x := x - 1
```

```
b := x + 10
```

11

Another Example starting at top


```
x := 5
a := x + 10
```

```
x := x + 1
x := x - 1
```

```
b := x + 10
```

12

2

Back to lattice

- $(D, \sqsubseteq, \bot, \top, \sqcup, \sqcap) =$
  $(2^A, \supseteq, A, \emptyset, \cap, \cup)$
  where $A = \{ x \rightarrow N \mid x \in Vars \wedge N \in Z \}$

- What's the problem with this lattice?

13

Back to lattice

- $(D, \sqsubseteq, \bot, \top, \sqcup, \sqcap) =$
  $(2^A, \supseteq, A, \emptyset, \cap, \cup)$
  where $A = \{ x \rightarrow N \mid x \in Vars \wedge N \in Z \}$

- What's the problem with this lattice?

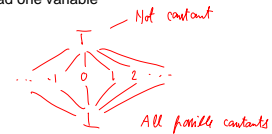- Lattice is infinitely high, which means we can't guarantee termination

14

Better lattice

- Suppose we only had one variable

15

Better lattice

- Suppose we only had one variable

- $D = \{ \bot, \top \} \cup Z$
- $\forall i \in Z \,.\, \bot \sqsubseteq i \wedge i \sqsubseteq \top$
- height = 3

16

For all variables

- Two possibilities
- Option 1: Tuple of lattices
- Given lattices $(D_1, \sqsubseteq_1, \bot_1, \top_1, \sqcup_1, \sqcap_1) \ldots (D_n, \sqsubseteq_n, \bot_n, \top_n, \sqcup_n, \sqcap_n)$ create:

  tuple lattice $D^n =$

17

For all variables

- Two possibilities
- Option 1: Tuple of lattices
- Given lattices $(D_1, \sqsubseteq_1, \bot_1, \top_1, \sqcup_1, \sqcap_1) \ldots (D_n, \sqsubseteq_n, \bot_n, \top_n, \sqcup_n, \sqcap_n)$ create:

  tuple lattice $D^n = ((D_1 \times \ldots \times D_n), \sqsubseteq, \bot, \top, \sqcup, \sqcap)$ where
  $\bot = (\bot_1, \ldots, \bot_n)$
  $\top = (\top_1, \ldots, \top_n)$
  $(a_1, \ldots, a_n) \sqcup (b_1, \ldots, b_n) = (a_1 \sqcup_1 b_1, \ldots, a_n \sqcup_n b_n)$
  $(a_1, \ldots, a_n) \sqcap (b_1, \ldots, b_n) = (a_1 \sqcap_1 b_1, \ldots, a_n \sqcap_n b_n)$
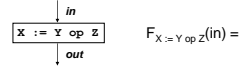 height = height(D_1) + ... + height(D_n)

18

For all variables

- Option 2: Map from variables to single lattice
- Given lattice $(D, \sqsubseteq_1, \bot_1, \top_1, \sqcup_1, \sqcap_1)$ and a set V, create:

  map lattice $V \to D = (V \to D, \sqsubseteq, \bot, \top, \sqcup, \sqcap)$

$$\bot = \lambda v \to \bot_1$$
$$\top = \lambda v \to \top_1$$
$$m_1 \sqcup m_2 = \lambda v \to m_1(v) \sqcup_1 m_2(v)$$
$$m_1 \sqsubseteq m_2 \iff \forall v.\ m_1(v) \sqsubseteq_1 m_2(v)$$

19

Back to example

$$F_{X := Y \text{ op } Z}(in) =$$

20

Back to example

$$F_{X := Y \text{ op } Z}(in) = in\ [\ X \to in(Y)\ \widehat{op}\ in(Z)\ ]$$
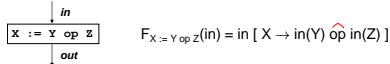
where $a\ \widehat{op}\ b =$

$\widehat{op}$	$\bot$	d_1	$\top$
$\bot$	$\bot$	$\bot$	$\top$
d_2	$\bot$	$d_1 \text{ op } d_2$	$\top$
$\top$	$\top$	$\top$	$\top$

21

General approach to domain design

- Simple lattices:
  - boolean logic lattice
  - powerset lattice
  - incomparable set: set of incomparable values, plus top and bottom (eg const prop lattice)
  - two point lattice: just top and bottom
- Use combinators to create more complicated lattices
  - tuple lattice constructor
  - map lattice constructor

22

May vs Must

- Has to do with definition of computed info

- Set of $x \to y$ must-point-to pairs
  - if we compute $x \to y$, then, then during program execution, x must point to y
- Set of $x \to y$ may-point-to pairs
  - if during program execution, it is possible for x to point to y, then we must compute $x \to y$

23

May vs must

	May	Must
most optimistic (bottom)		
most conservative (top)		
safe		
merge		

24

4

May vs must

	May	Must
most optimistic (bottom)	empty set	full set
most conservative (top)	full set	empty set
safe	overly big	overly small
merge	$\cup$	$\cap$

25

Common Sub-expression Elim

- Want to compute when an expression is available in a var
- Domain:

26

Common Sub-expression Elim

- Want to compute when an expression is available in a var
- Domain:

$$S = \{ X \to E \mid X \in Var, \, E \in Expr \}$$

$$D = 2^S$$
$$\bot = S$$
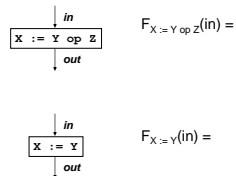$$\top = \emptyset$$
$$\sqcup = \cap$$

27

Flow functions

$F_{X := Y \, op \, Z}(in) =$

$F_{X := Y}(in) =$

28

Flow functions

$F_{X := Y \, op \, Z}(in) = in - \{ X \to * \}$
$\qquad - \{ * \to ... X ... \} \cup$
$\qquad \{ X \to Y \, op \, Z \mid X \neq Y \wedge X \neq Z \}$

$F_{X := Y}(in) = in - \{ X \to * \}$
$\qquad - \{ * \to ... X ... \} \cup$
$\qquad \{ X \to E \mid Y \to E \in in \}$

29

Example


```
x := read()
v := a + b
```
```
x := x + 1
w := x + 1
```
```
w := x + 1
a := w
v := a + b
```
```
z := x + 1
t := a + b
```

30

5
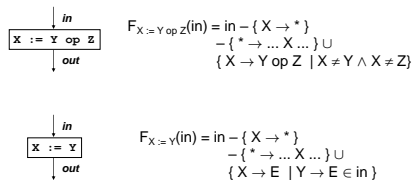
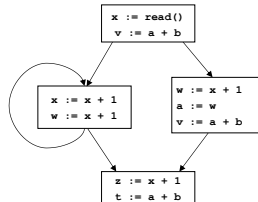## Direction of analysis

- Although constraints are not directional, flow functions are
- All flow functions we have seen so far are in the forward direction
- In some cases, the constraints are of the form
  in = F(out)
- These are called backward problems.
- Example: live variables
  – compute the set of variables that may be live

31

Live Variables

- A variable is live at a program point if it will be used before being redefined
- A variable is dead at a program point if it is redefined before being used

32

Example: live variables

- Set D =
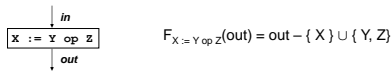- Lattice: $(D, \sqsubseteq, \bot, \top, \sqcup, \sqcap)$ =

33

Example: live variables

- Set D = 2^{Vars}
- Lattice: $(D, \sqsubseteq, \bot, \top, \sqcup, \sqcap) = (2^{Vars}, \subseteq, \emptyset, Vars, \cup, \cap)$

$$\begin{array}{c} \downarrow in \\ \boxed{\texttt{x := Y op Z}} \\ \downarrow out \end{array} \qquad F_{X := Y \, op \, Z}(out) =$$
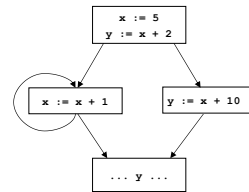
34

Example: live variables

- Set D = 2^{Vars}
- Lattice: $(D, \sqsubseteq, \bot, \top, \sqcup, \sqcap) = (2^{Vars}, \subseteq, \emptyset, Vars, \cup, \cap)$

$$\begin{array}{c} \downarrow in \\ \boxed{\texttt{x := Y op Z}} \\ \downarrow out \end{array} \qquad F_{X := Y \, op \, Z}(out) = out - \{ X \} \cup \{ Y, Z\}$$

35

Example: live variables

```
         x := 5
      y := x + 2

 x := x + 1      y := x + 10

         . . . y . . .
```

36

Slide 37

Example: live variables
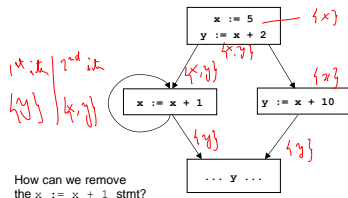

```
                    x := 5      {x}
                    y := x + 2
                       {x,y}
        1st it | 2nd it
        {y}    | {x,y}
        {x,y}      {x}
    x := x + 1      y := x + 10
        {y}           {y}
              . . . y . . .    {y}

How can we remove
the x := x + 1 stmt?
```

Slide 38

Revisiting assignment


```
        in
  x := Y op Z
        out
```

$F_{X := Y \text{ op } Z}(\text{out}) = \text{out} - \{ X \} \cup \{ Y, Z \}$

Slide 39

Revisiting assignment


```
        in
  x := Y op Z
        out
```
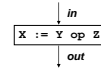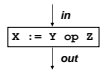
$F_{X := Y \text{ op } Z}(\text{out}) = \text{out} - \{ X \} \cup \{ Y, Z \}$

$$\text{out} - \{x\} \cup$$
$$x \notin \text{out}? \ \emptyset : \{Y, Z\}$$

Slide 40

Theory of backward analyses

- Can formalize backward analyses in two ways
- Option 1: reverse flow graph, and then run forward problem
- Option 2: re-develop the theory, but in the backward direction

Slide 41

Precision

- Going back to constant prop, in what cases would we lose precision?

Slide 42

Precision

- Going back to constant prop, in what cases would we lose precision?

```
x := 5              if (p) {          if (...) {
if (<expr>) {          x := 5;           x := -1;
    x := 6          } else            } else
}                      x := 4;           x := 1;
... x ...           }                 }
                    ...               y := x * x;
where <expr> is     if (p) {          ... y ...
equiv to false         y := x + 1
                    } else {
                       y := x + 2
                    }
                    ... y ...
```

Precision

- The first problem: Unreachable code
  - solution: run unreachable code removal before
  - the unreachable code removal analysis will do its best, but may not remove all unreachable code
- The other two problems are path-sensitivity issues
  - Branch correlations: some paths are infeasible
  - Path merging: can lead to loss of precision

MOP: meet over all paths

- Information computed at a given point is the meet of the information computed by each path to the program point

```
if (...) {
    x := -1;
} else
    x := 1;
}
y := x * x;
... y ...
```

MOP

- For a path p, which is a sequence of statements $[s_1, ..., s_n]$, define: $F_p(in) = F_{s_n}(...F_{s_1}(in) ...)$
- In other words: $F_p = F_{s_1} \circ \cdots \circ F_{s_n}$
- Given an edge e, let paths-to(e) be the (possibly infinite) set of paths that lead to e
- Given an edge e, $MOP(e) = \bigsqcup_{p \in paths\text{-}to(e)} F_p(\perp)$

- For us, should be called JOP (ie: join, not meet)

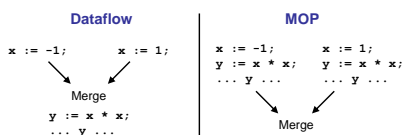## MOP vs. dataflow

- MOP is the "best" possible answer, given a fixed set of flow functions
  - This means that MOP $\sqsubseteq$ dataflow at edge in the CFG
- In general, MOP is not computable (because there can be infinitely many paths)
  - vs dataflow which is generally computable (if flow fns are monotonic and height of lattice is finite)
- And we saw in our example, in general, MOP $\neq$ dataflow

MOP vs. dataflow

- However, it would be great if by imposing some restrictions on the flow functions, we could guarantee that dataflow is the same as MOP. What would this restriction be?

```
        Dataflow              |              MOP
x := -1;        x := 1;       |    x := -1;        x := 1;
                              |    y := x * x;     y := x * x;
        Merge                 |    ... y ...       ... y ...
y := x * x;                   |
... y ...                     |              Merge
```

MOP vs. dataflow

- However, it would be great if by imposing some restrictions on the flow functions, we could guarantee that dataflow is the same as MOP. What would this restriction be?
- Distributive problems. A problem is distributive if:

  $\forall$ a, b . $F(a \sqcup b) = F(a) \sqcup F(b)$

- If flow function is distributive, then MOP = dataflow

Summary of precision

- Dataflow is the basic algorithm
- To basic dataflow, we can add path-separation
  - Get MOP, which is same as dataflow for distributive problems
  - Variety of research efforts to get closer to MOP for non-distributive problems
- To basic dataflow, we can add path-pruning
  - Get branch correlation
- To basic dataflow, can add both:
  - meet over all feasible paths

49