DATA FLOW ANALYSIS [ASU1e Ch. 10.5–6] [ALSU2e Ch. 9.2-4] [Muchnick Ch. 8]

Conservative approximation to global information on data flow properties that are relevant for optimizations

→ MAY-problems vs. MUST-problems

Examples:

- Constant Propagation Analysis

  Has *var* always the same constant value at this point?

- Reaching Definitions

  Which definitions of *var* may be relevant for this use?

– local (BB)

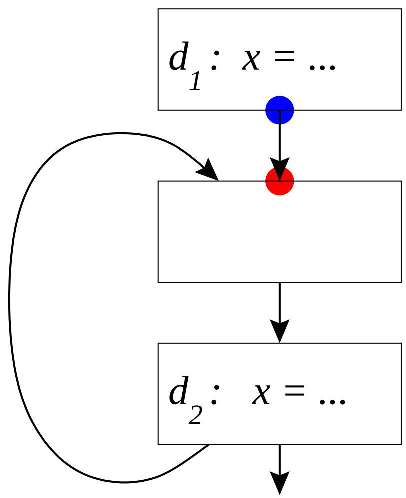
– global (CFG) using "effects" of entire BB's (summary info)

forward vs. backward, iterative vs. interval-based vs. structured...

– interprocedural

Example: Reaching Definitions

Definition d of variable v: $d: v \leftarrow \ldots$



d reaches a point p in CFG
if there is a path $d \rightarrow^* p$ in CFG (excl. d, p)
that contains no kill of d (= reassignment of v)

NB: Whether a specific definition d *actually* reaches a specific program point p is undecidable in the formal sense!

(program behavior may e.g. depend on run-time input)

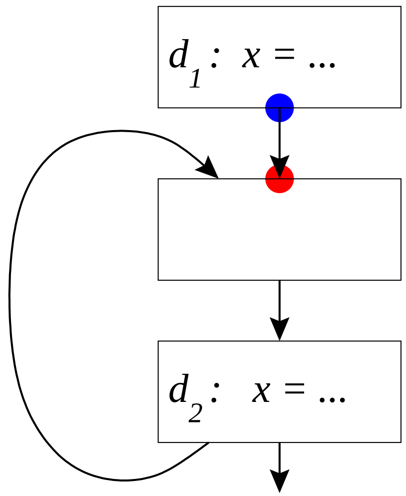
$\rightarrow$ conservative approximation

MAY-REACH or MUST-REACH, depending on the application.

Example: Reaching Definitions (cont.)

Definition d of variable v: $d: v \leftarrow \ldots$

d reaches a point p in CFG
if there is a path $d \rightarrow^* p$ in CFG (excl. d, p)
that contains no kill of d (= reassignment of v)



Summarize the effect of each basic block (which can be analyzed locally):

- A basic block B *generates* (contains) some definitions
- A basic block B *kills all* definitions d' that write *any* variable v defined in B.
- A definition d that is not killed by a basic block B is *preserved* by B.

Background: Bitvector representation of sets

Given: Finite global set (universe) U

Any subset $S \subseteq U$ can be represented as a bitvector b_S
with $b_S[i] = 1$ iff the ith element of U is in S.

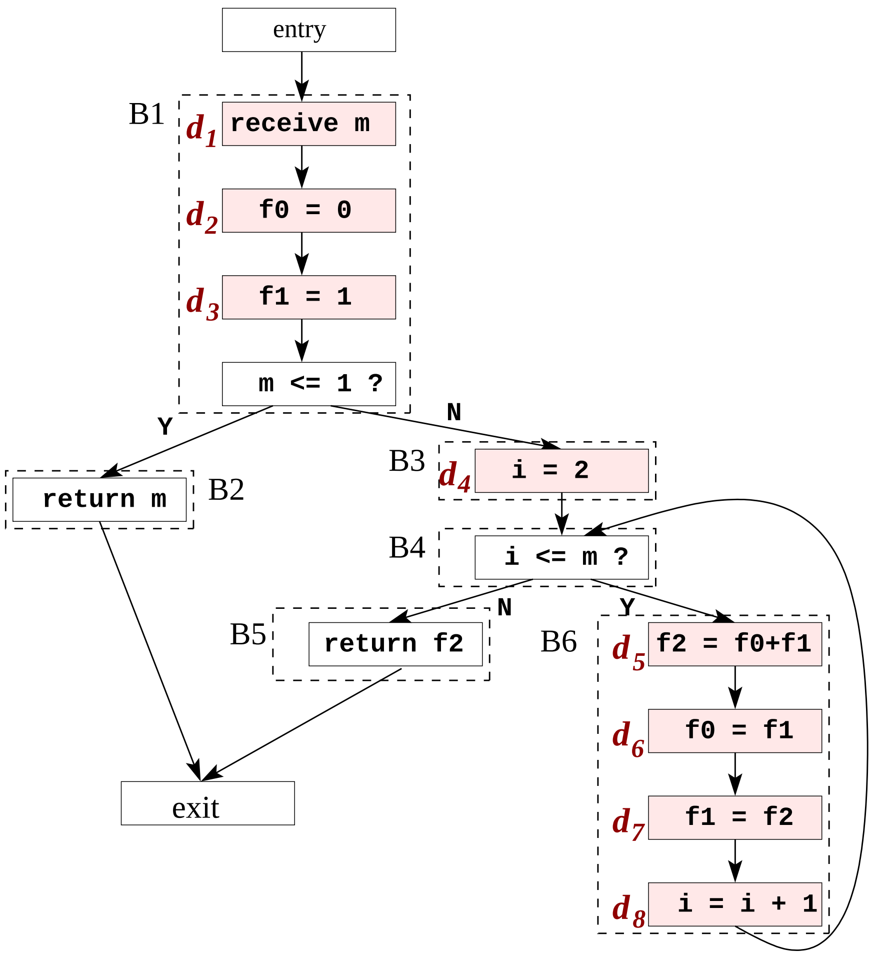
Example:

$U = \{a, b, c, d, e, f, g, h\}$
$S = \{a, d, e\}$ has bitvector representation $b_S = \langle 10011000 \rangle$.

If clear from the context, we simplify the notation, using S for b_S:
$S = \langle 10011000 \rangle$.

Here: Consider bitvector representation of sets of definitions
i.e., the universe U = the set of all definitions in the program
= set of all CFG nodes (e.g. MIR statements) writing to some variable.

Example (cont.): Bitvector Representation of Definitions; GEN sets



Bit	Definition (generated)	Basic block
1	d_1 of `m` in node 1	B1
2	d_2 of `f0` in node 2	
3	d_3 of `f1` in node 3	
4	d_4 of `i` in node 4	B3
5	d_5 of `f2` in node 8	B6
6	d_6 of `f0` in node 9	
7	d_7 of `f1` in node 10	
8	d_8 of `i` in node 11	

$$GEN(B1) = \{d_1, d_2, d_3\} = \langle 11100000 \rangle$$

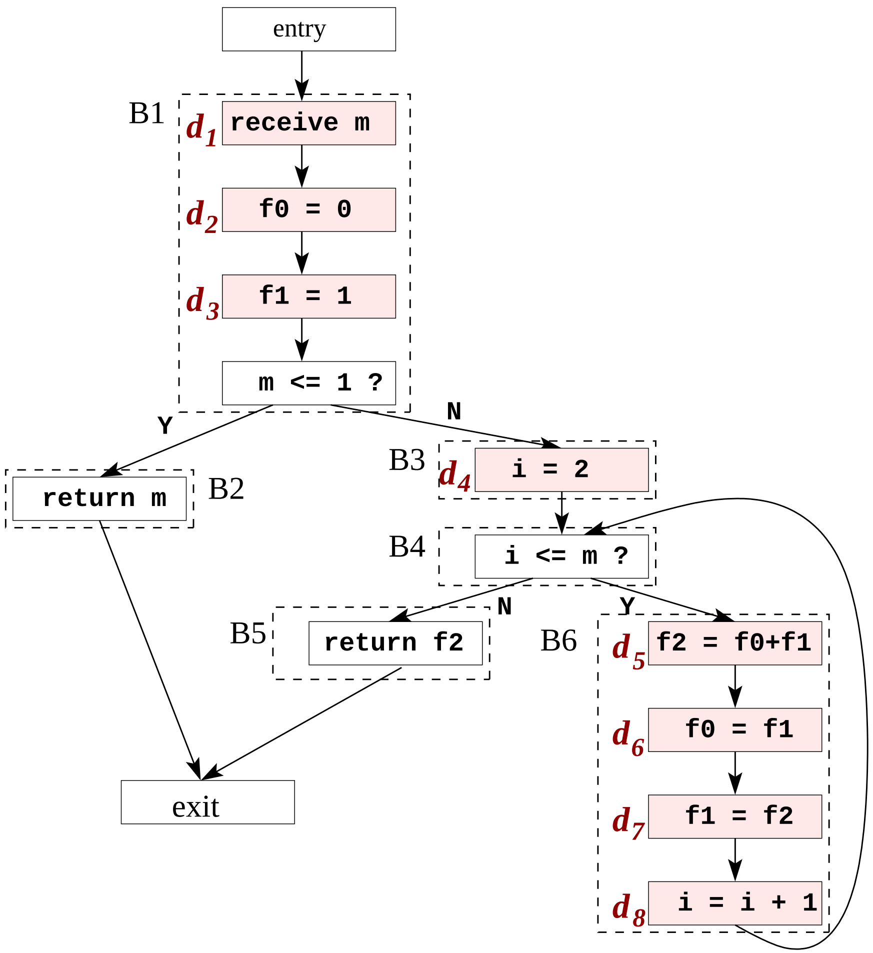
$$GEN(B3) = \{d_4\} = \langle 00010000 \rangle$$

$$GEN(B6) = \{d_5, d_6, d_7, d_8\} = \langle 00001111 \rangle$$

$$GEN(Bi) = \{\} = \langle 00000000 \rangle$$

for $i \neq B1, B3, B6$

Example (cont.): Bitvector Representation of Definitions; *KILL* sets



Bit	Definition (generated)	Basic block
1	d_1 of `m` in node 1	B1
2	d_2 of `f0` in node 2	
3	d_3 of `f1` in node 3	
4	d_4 of `i` in node 4	B3
5	d_5 of `f2` in node 8	B6
6	d_6 of `f0` in node 9	
7	d_7 of `f1` in node 10	
8	d_8 of `i` in node 11	

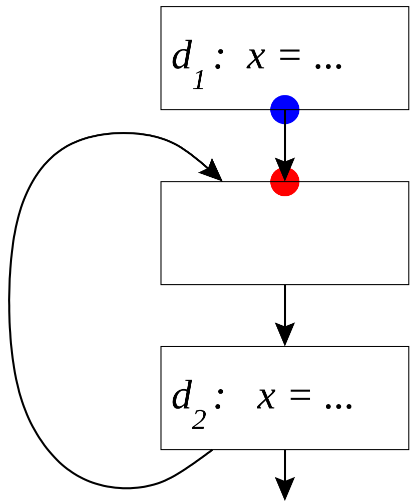
$$KILL(B1) = \{d_1, d_2, d_3, d_6, d_7\} = \langle 11100110 \rangle$$

$$KILL(B3) = \{d_4, d_8\} = \langle 00010001 \rangle$$

$$KILL(B6) = \{d_2, d_3, d_4, d_5, d_6, d_7, d_8\} = \langle 01111111 \rangle$$

$$KILL(Bi) = \{\} = \langle 00000000 \rangle \text{ for } i \neq B1, B3, B6$$

Example: Reaching definitions with bitvector representation



$RDin(B) = \langle 00100010 \rangle$ (1 = def. reaches entry of B)

Certainly, $RDin(\boxed{\text{entry}}) = \langle 00000000 \rangle$

$RDin(B)$ for $B \neq \boxed{\text{entry}}$?

Effect of a node B in CFG on definitions d reaching it:

described by 2 sets $GEN(B)$, $KILL(B)$:

$GEN(B) = \langle 11100000 \rangle$ (1 iff B generates this definition)

$KILL(B) = \langle 11100110 \rangle$ (1 iff B kills this definition)

$RDout(B) = \langle 111??00? \rangle$ (1 = def. reaches end of B, ? = bit as in $RDin(B)$)

Example: $RDin(B) = \langle 10001101 \rangle$ and effect of B as above

$\Longrightarrow RDout(B) = \langle 11101001 \rangle$

Example (cont.): Reaching Definitions — Dataflow Equations

Flow functions — Effect of a basic block B on any $RDin(B)$:

Set equation: $$RDout(B) = GEN(B) \cup (RDin(B) - KILL(B)) \quad \forall B$$

Bitvector equation: $$RDout(B) = GEN(B) \vee (RDin(B) \wedge \overline{KILL(B)}) \quad \forall B$$

Effect of joining control flow paths:

Set equation: $$RDin(B) = \bigcup_{P \in Pred(B)} RDout(P) \quad \forall B$$ (for MUST-REACH: $\bigcap$)

Bitvector equation: $$RDin(B) = \bigvee_{P \in Pred(B)} RDout(P) \quad \forall B$$ (for MUST-REACH: $\bigwedge$)

Reaching Definitions is a forward flow problem:

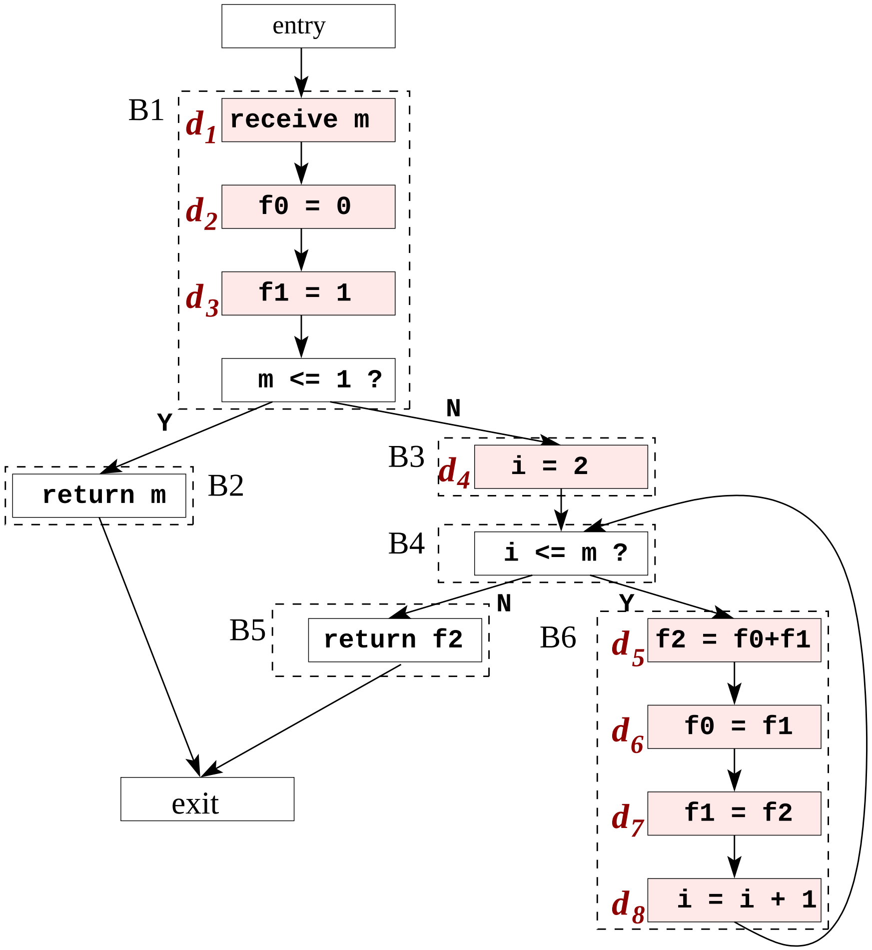
- BB flow functions specify outgoing property as function of ingoing
- Information propagates through CFG in direction from entry towards exit

Iterative computation of Reaching Definitions

Algorithm: (Fixed-point iteration)

- For MAY-Reach we initialize
  $RDin(\boxed{\text{entry}}) = \{\} = \langle 00000000 \rangle$,
  $RDin(B) = \{\} = \langle 00000000 \rangle$
  for all other B

- Iterate,
  applying the equations
  to $RDin(B)$, $RDout(B)$ for all B
  until no more changes occur.



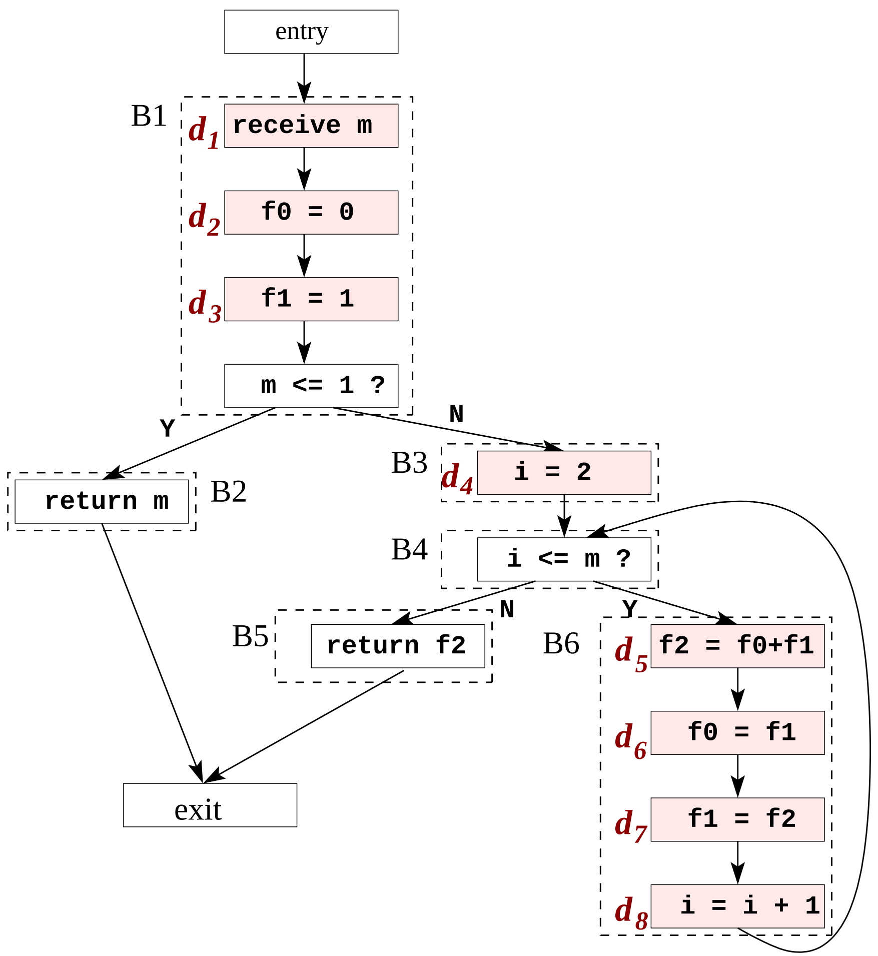
Example: see whiteboard

Why does this work?

Example (cont.): Iterative computation of Reaching Definitions

First iteration:

$RDin(\boxed{\text{entry}}) = \langle 00000000 \rangle$
$RDout(\boxed{\text{entry}}) = \langle 00000000 \rangle$
$RDin(B1) = \langle 00000000 \rangle$
$RDout(B1) = \langle 11100000 \rangle$ — changed!
$RDin(B2) = \langle 11100000 \rangle$
$RDout(B2) = \langle 11100000 \rangle$
$RDin(B3) = \langle 11100000 \rangle$
$RDout(B3) = \langle 11110000 \rangle$
$RDin(B4) = \langle 11110000 \rangle$
$RDout(B4) = \langle 11110000 \rangle$
$RDin(B5) = \langle 11110000 \rangle$
$RDout(B5) = \langle 11110000 \rangle$
$RDin(B6) = \langle 11110000 \rangle$
$RDout(B6) = \langle 10001111 \rangle$
$RDin(\boxed{\text{exit}}) = \langle 11110000 \rangle$
$RDout(\boxed{\text{exit}}) = \langle 11110000 \rangle$



Example (cont.): Iterative computation of Reaching Definitions

Second iteration:

$RDin(\boxed{\text{entry}}) = \langle 00000000 \rangle$

$RDout(\boxed{\text{entry}}) = \langle 00000000 \rangle$

$RDin(B1) = \langle 00000000 \rangle$

$RDout(B1) = \langle 11100000 \rangle$

$RDin(B2) = \langle 11100000 \rangle$

$RDout(B2) = \langle 11100000 \rangle$

$RDin(B3) = \langle 11100000 \rangle$

$RDout(B3) = \langle 11110000 \rangle$

$RDin(B4) = \langle 11111111 \rangle$ — changed!

$RDout(B4) = \langle 11111111 \rangle$

$RDin(B5) = \langle 11111111 \rangle$

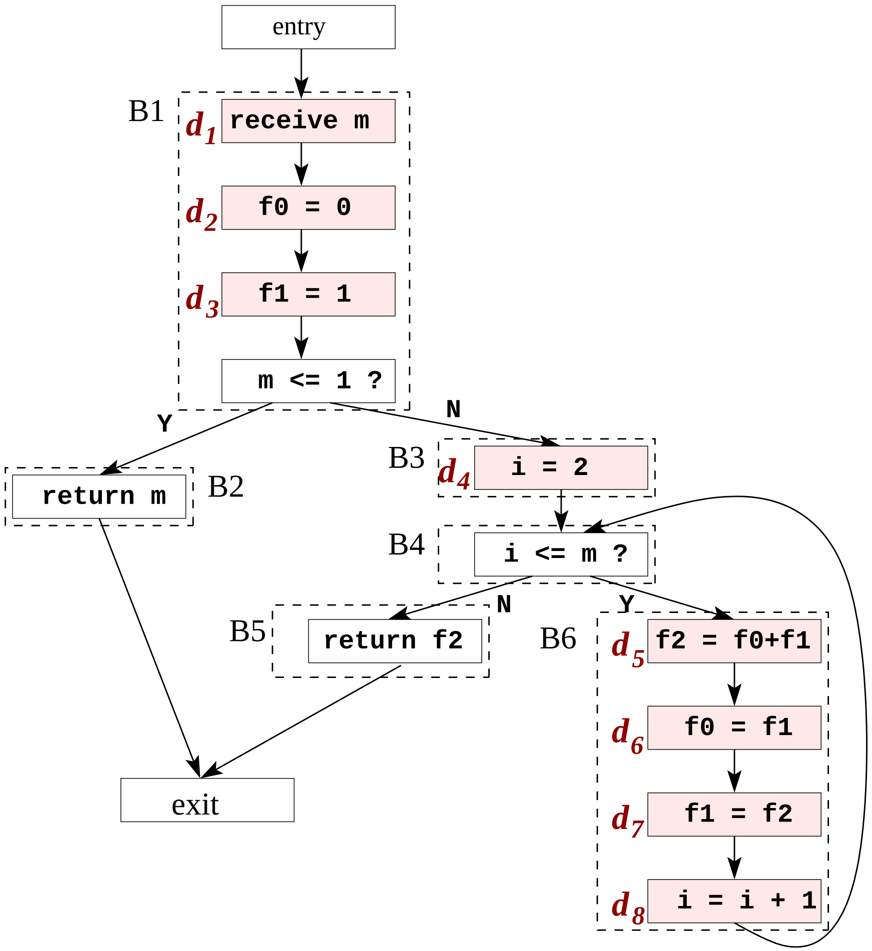
$RDout(B5) = \langle 11111111 \rangle$

$RDin(B6) = \langle 11111111 \rangle$

$RDout(B6) = \langle 10001111 \rangle$

$RDin(\boxed{\text{exit}}) = \langle 11111111 \rangle$

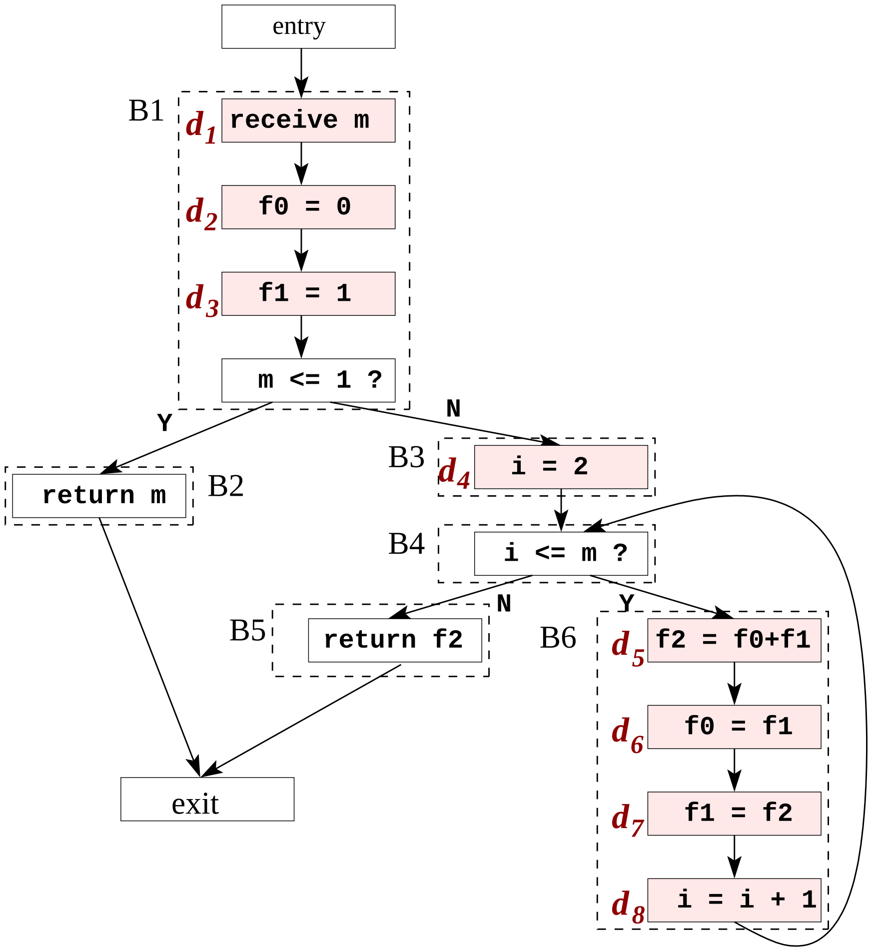
$RDout(\boxed{\text{exit}}) = \langle 11111111 \rangle$



Example (cont.): Iterative computation of Reaching Definitions

Third iteration:

$RDin(\boxed{\text{entry}}) = \langle 00000000 \rangle$
$RDout(\boxed{\text{entry}}) = \langle 00000000 \rangle$
$RDin(B1) = \langle 00000000 \rangle$
$RDout(B1) = \langle 11100000 \rangle$
$RDin(B2) = \langle 11100000 \rangle$
$RDout(B2) = \langle 11100000 \rangle$
$RDin(B3) = \langle 11100000 \rangle$
$RDout(B3) = \langle 11110000 \rangle$
$RDin(B4) = \langle 11111111 \rangle$
$RDout(B4) = \langle 11111111 \rangle$
$RDin(B5) = \langle 11111111 \rangle$
$RDout(B5) = \langle 11111111 \rangle$
$RDin(B6) = \langle 11111111 \rangle$
$RDout(B6) = \langle 10001111 \rangle$
$RDin(\boxed{\text{exit}}) = \langle 11111111 \rangle$
$RDout(\boxed{\text{exit}}) = \langle 11111111 \rangle$



No more change — done!

Why does this work?

Underlying theory:

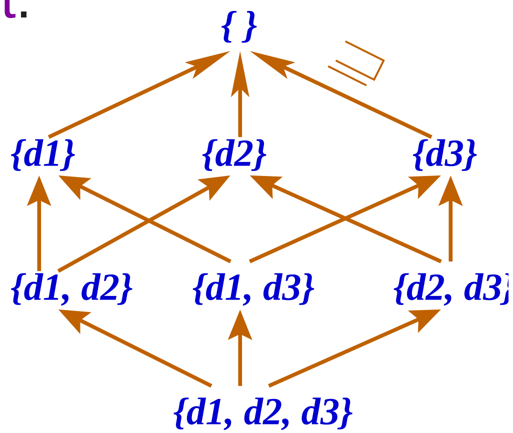
- Posets, least upper bounds, semilattices, lattices
- Monotone flow functions
- Data flow analysis framework
- Meet-over-all-paths
- Convergence theorems for iterative data flow analysis

Posets

A relation $\sqsubseteq$ on a set L defines a partial order on L
if, for all x, y and z in L,

1. $x \sqsubseteq x$ (reflexive),
2. If $x \sqsubseteq y$ and $y \sqsubseteq x$ then $x = y$ (antisymmetric), and
3. If $x \sqsubseteq y$ and $y \sqsubseteq z$ then $x \sqsubseteq z$ (transitive).

The pair $(L, \sqsubseteq)$ is called a poset or partially ordered set.



Notation: $x \sqsubset y$ iff $x \sqsubseteq y$ and $x \neq y$.

Example: $L = 2^S$ for a set S, $\sqsubseteq$ = $\supseteq$

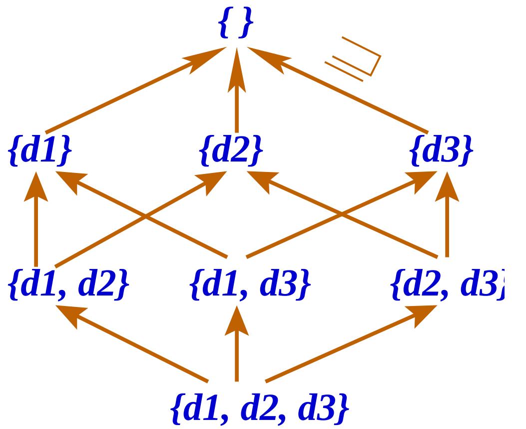
Interpretation in data flow analysis: $x \sqsubseteq y$ means "x is not more precise than y"

Least upper bound, greatest lower bound

Given poset $(L, \sqsubseteq)$.

A greatest lower bound (glb) of any two elements $x, y \in L$
is an element $g \in L$ such that

1. $g \sqsubseteq x$,
2. $g \sqsubseteq y$, and
3. for any $z \in L$ with $z \sqsubseteq x$ and $z \sqsubseteq y$, $z \sqsubseteq g$.



Example: For $(2^S, \supseteq)$, glb is set union ($\cup$).

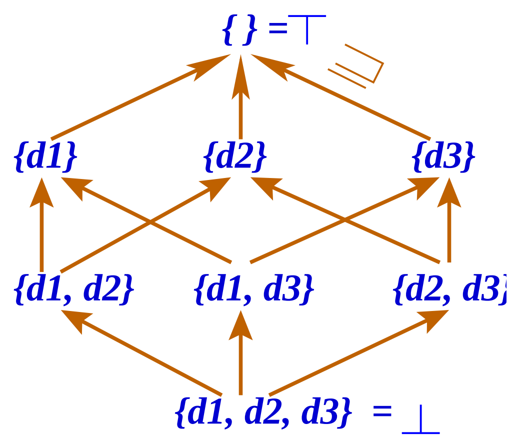
Analogously: Least upper bound (lub).

A poset $(L, \sqsubseteq)$ where any two elements in L have a greatest lower bound in L (i.e., closedness under glb) is a necessary condition for a semilattice.

Semilattice

A semilattice $(L, \sqcap)$
consists of a set L and a binary meet operator $\sqcap$
such that for all $x, y \in L$,

1. $x \sqcap x = x$ (meet is idempotent),
2. $x \sqcap y = y \sqcap x$ (meet is commutative),
3. $x \sqcap (y \sqcap z) = (x \sqcap y) \sqcap z$ (meet is associative),



and there is a top element $\top \in L$ such that

4. for all $x \in L$, $\top \sqcap x = x$.

Optionally, a semilattice may also have a bottom element $\bot \in L$ with

for all $x \in L$, $\bot \sqcap x = \bot$.

Example 1: $(2^S, \cup)$ is a semilattice with $\top = \{\}$ and $\bot = S$.

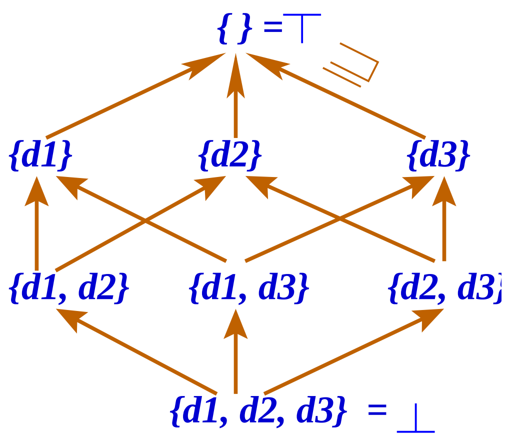
Example 2: $(2^S, \cap)$ is a semilattice with $\top = S$ and $\bot = \{\}$.

Semilattice and partial order

A semilattice $(L, \sqcap)$
implicitly defines a partial order $\sqsubseteq$
where, for all $x, y \in L$,

$$x \sqsubseteq y \text{ iff } x \sqcap y = x.$$

The glb is just the $\sqcap$ operator.



Example 1: $(2^S, \cup)$ implicitly defines partial order $\supseteq$.

Example 2: $(2^S, \cap)$ implicitly defines partial order $\subseteq$.

Note: $\bot \sqsubseteq x \sqsubseteq \top$ for all $x \neq \top$, $x \neq \bot$.

Interpretation: $\top$ is most precise information, $\bot$ is most imprecise information.

Lattice

Lattice $(L, \sqcap, \sqcup)$

- set L of values
- meet operation $\sqcap$, join operation $\sqcup$ where

(1) for all $x, y \in L$ ex. unique $z, w \in L$: $x \sqcap y = z$, $x \sqcup y = w$ (closedness)

(2) for all $x, y \in L$: $x \sqcap y = y \sqcap x$, $x \sqcup y = y \sqcup x$ (commutativity)

(3) for all $x, y, z \in L$: $(x \sqcap y) \sqcap z = x \sqcap (y \sqcap z)$, $(x \sqcup y) \sqcup z = x \sqcup (y \sqcup z)$ (associativity)

(4) there are two unique elements of L:

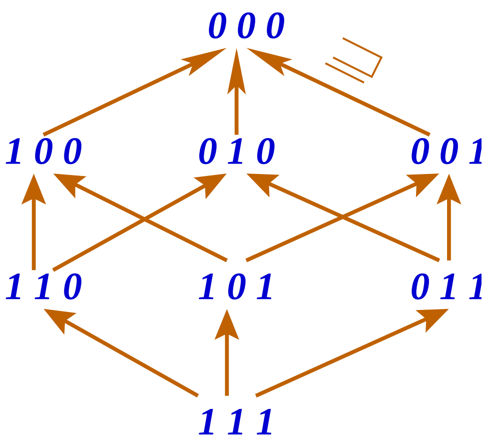
$\top$ "top": $\forall x \in L,\ x \sqcup \top = \top$

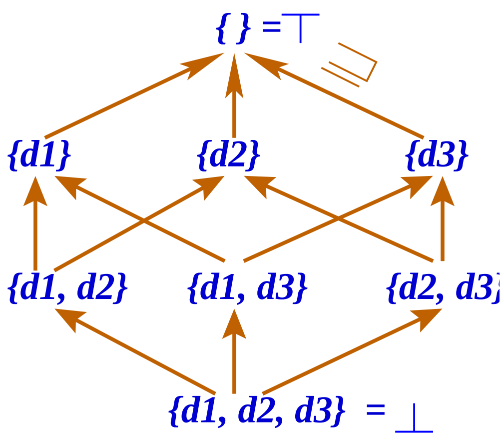
$\bot$ "bottom": $\forall x \in L,\ x \sqcap \bot = \bot$

(5) often also distributivity of $\sqcap$, $\sqcup$ given

Example: Bitvector Lattice

Bitvector lattice: $L = BV^3$, $\sqcap$ = union/bitwise OR, $\sqcup$ = inters./bitwise AND





001 $\sqcap$ 101 = 101
001 $\sqcup$ 101 = 001

partial order $\sqsubseteq$:

$x \sqsubseteq y$ iff $x \sqcap y = x$
(transitive, antisymmetric, reflexive)

for all x: $\bot \sqsubseteq x \sqsubseteq \top$

meet $x \sqcap y$: follow paths in L from x, y downwards until they meet
(greatest lower bound w.r.t. $\sqsubseteq$)

join $x \sqcup y$: follow paths in L from x, y upwards until they join
(least upper bound w.r.t. $\sqsubseteq$)

Lattices: Monotonicity, Height; Termination

$f: L \rightarrow L$

is monotone iff $\forall x, y \in L$: $x \sqsubseteq y \Rightarrow f(x) \sqsubseteq f(y)$

Example:

$f: BV^3 \rightarrow BV^3$ with $f(\langle x_1 x_2 x_3 \rangle) = (x_1 1 x_3)$ for all $x_1, x_2, x_3 \in BV$ is monotone.

$g: BV^1 \rightarrow BV^1$ with $g(\langle 0 \rangle) = \langle 1 \rangle$ and $g(\langle 1 \rangle) = \langle 0 \rangle$ is not monotone.

Height of $(L, \sqcap, \sqcup)$

= length of longest strictly ascending chain in L

= max. n: $\exists x_1, x_2, ..., x_n \in L$ with $\bot = x_1 \sqsubseteq x_2 \sqsubseteq ... \sqsubseteq x_n = \top$

Example:

Height of BV^3 is 4.

Finite height + Monotonicity $\Rightarrow$ Termination of the fixed-point iteration

Flow functions

Flow functions specify the effect of a programming language construct as a mapping $L \rightarrow L$.

E.g., in Reaching Definitions:

BB B_1 generates d_1, d_2, d_3, kills d_1, d_2, d_3, d_6, d_7:

$$F_{B_1}(\langle x_1 x_2 x_3 x_4 x_5 x_6 x_7 x_8 \rangle) = \langle 111 x_4 x_5 00 x_8 \rangle$$

$$F_{B_3}(\langle x_1 x_2 x_3 x_4 x_5 x_6 x_7 x_8 \rangle) = \langle x_1 x_2 x_3 1 x_5 x_6 x_7 0 \rangle$$

$$F_{B_6}(\langle x_1 x_2 x_3 x_4 x_5 x_6 x_7 x_8 \rangle) = \langle x_1 0001111 \rangle$$

$$F_{B_j} = id \text{ for all } j \notin \{1, 3, 6\}$$

Flow functions must be monotone.

(otherwise the fixed-point iteration algorithm could oscillate)

Fixed points

Fixed point of a function $f: L \rightarrow L$

is a $z \in L$ with $f(z) = z$

- Solution to a set of data flow equations
- In general not unique!

Example:

$f: BV \rightarrow BV$ with $f(0) = 0$ and $f(1) = 1$

has 2 fixed points: 0 and 1.

Reaching definitions (see above):

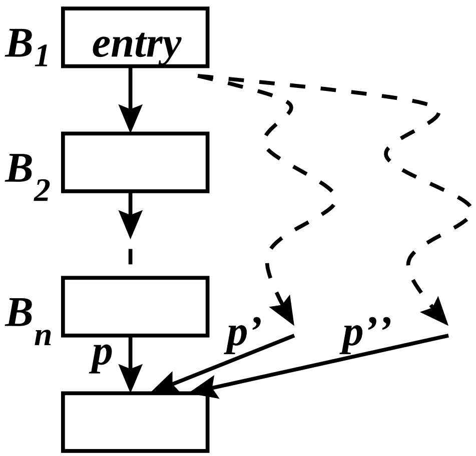
iterate until $f(RDin(B)) = RDin(B)\ \forall B$

where f = composition of all flow functions and equations.

The ideal solution

Ideal solution (IDEAL) to the data flow equations
(for forward problems):

- begin with initial information $Init$ at entry
- apply composition of flow functions along all *really* possible paths from entry to each CFG node B and compose these results by the meet operator:



$$F_p = F_{B_n} \circ \ldots \circ F_{B_1}$$

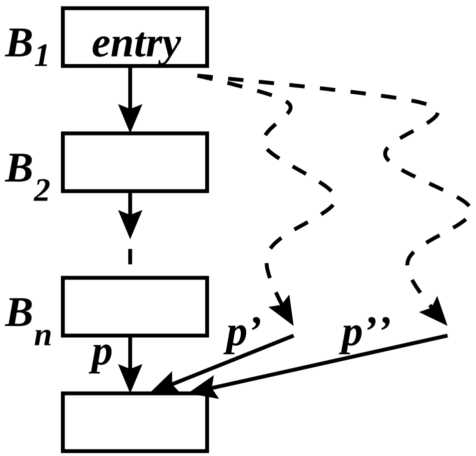
$$IDEAL(B) = \bigsqcap_{p \in Paths(B)} F_p(Init)$$

similarly for backward problems

Meet over all paths (MOP)

Meet-over-all-paths (MOP) solution to data flow equations (for forward problems):

- begin with initial information $Init$ at entry
- apply composition of flow functions along all possible paths from entry to each CFG node B and compose these results by the meet operator:



$$F_p = F_{B_n} \circ \ldots \circ F_{B_1}$$

$$MOP(B) = \bigsqcap_{p \in Paths(B)} F_p(Init)$$

similarly for backward problems

MOP vs. IDEAL

A solution in is *safe* if $in(B) \sqsupseteq IDEAL[B]\ \forall B$

A solution in is *incorrect* if $in(B) \sqsubset IDEAL[B]$ for some B

BUT: $IDEAL$ is statically undecidable!

The exact subset of the paths really taken at run time

is not statically known. E.g., an else branch or loop may never be executed.

$IDEAL(B)$ = Meet over all paths to B possibly taken at run time

$NEVER(B)$:= Meet over all remaining paths to B (never executed)

The most precise solution is $IDEAL(B)$,

but $MOP(B) = IDEAL(B) \sqcap NEVER(B)$,

i.e., $MOP(B) \sqsubseteq IDEAL(B)$.

MOP is the best solution that we could compute statically.

Fixed point solutions of the dataflow equations

Goal: find the maximum fixed point (MFP) solution

(maximal w.r.t. information, i.e., also w.r.t. $\sqsubseteq$, and still safe)

Theorem [Kildall'73]

If all flow functions distributive over $\sqcap$, $\sqcup$

i.e., $\forall x, y,\ f(x \sqcap y) = f(x) \sqcap f(y)$ and $f(x \sqcup y) = f(x) \sqcup f(y)$,

$\Rightarrow$ iterative DFA computes MFP, and MFP = MOP

Theorem [Kam/Ullman'75]

If all flow functions monotone but not necessarily distributive

$\Rightarrow$ iterative DFA computes MFP but not necessarily the MOP solution

Iterative Data Flow Analysis [Kildall'73]

given: CFG $G = (N, E)$, Lattice $(L, \sqcap, \sqcup)$

dataflow equations

$$in(B) = \begin{cases} Init & \text{for } B = \boxed{\text{entry}} \\ \prod_{P \in Pred(B)} out(P) & \text{otherwise} \end{cases}$$

$$out(B) = F_B(in(B))$$

or, by substitution,

$$in(B) = \begin{cases} Init & \text{for } B = \boxed{\text{entry}} \\ \prod_{P \in Pred(B)} F_P(in(P)) & \text{otherwise} \end{cases}$$

$Init$ is usually $\top$ (for $\sqcap$) or $\perp$ (for $\sqcup$)

Iterative DFA: Worklist algorithm (1)

- Implements the fixed-point algorithm above
- Maintain a *worklist* of blocks B
  whose predecessors' in values have changed in the last iteration
- worklist contains initially all BB's (except entry)
- iterate applying the dataflow equations
  until no more changes occur

Observation: maximal effect on forwarding information
if BB's in worklist are processed in topological order

$\rightarrow$ start with reverse postorder
$\rightarrow$ queue as worklist

$\Rightarrow$ $A+2$ iterations for a (sub-)CFG with A back edges [Hecht/Ullman'75]

Iterative DFA: Worklist algorithm (2)

```
Worklist_It ( N, [entry], F, DFin, Init )
    Set<Node> N;
    Node [entry];
    Functions F : Node × L → L;
    Function DFin: Node → L;
    L Init;  // (L, ⊓) is the (semi-)lattice
{
    L totaleffect, effectP;
    List<Node> W ← N − {[entry]};
    DFin([entry]) ← Init;
    for each B ∈ N do
        DFin(B) ← ⊤;
    ...
    ...
    repeat
        B ← W.delete_first_element();
        totaleffect ← ⊤;
        for each P ∈ Pred(B) do
            effectP ← F(P, DFin(P));
            totaleffect ← totaleffect ⊓ effectP;
        if DFin(B) ≠ totaleffect then
            DFin(B) ← totaleffect;
            W ← W ∪ Succ(B);
    until W = ∅;
    return DFin;
}
```

Survey of some data flow problems

classified by:

- information to be computed
- direction of information flow: forward / backward / bidirectional
- lattices used, meanings attached to lattice elements etc.

Reaching Definitions

forward, bitvector (1 bit per definition of a variable)

Available Expressions

forward, bitvector (1 bit per definition of an expression)

Live Variables

backward, bitvector (1 bit per use of a variable)

Survey of some data flow problems (cont.)

Upwards Exposed Uses

backward, bitvector (1 bit per use of a variable)

Copy-Propagation Analysis

forward, bitvector (1 bit per copy assignment)

Constant-Propagation Analysis

forward, ICP^n (or similar)

1 lattice value per def., symbolic execution

Partial Redundancy Analysis

[Morel,Renvoise'81] bidirectional, bitvector (1 bit per expression computation)

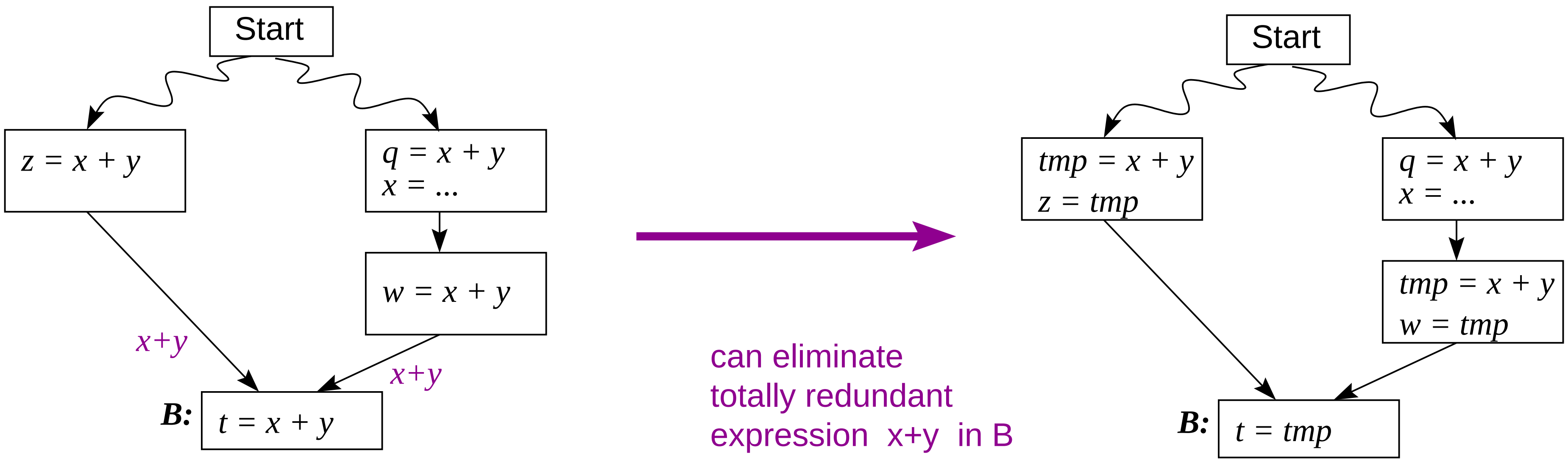
[Knoop/Rüthing/Steffen'92] "Lazy Code Motion"

Available Expressions

An expression, say x+y, is *available* at a point p if:

(1) *every* path from the entry node to p evaluates x+y, and

(2) after the last evaluation prior to reaching p,
there are no subsequent assignments to x or y.

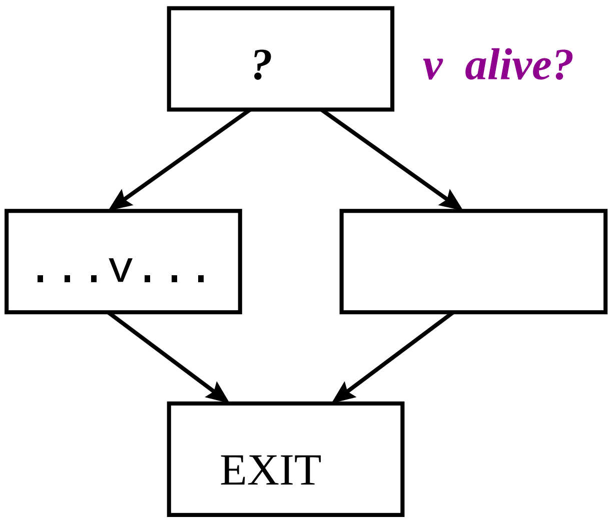


We say that a basic block *kills* expression x+y
if it *may* assign x or y, and does not subsequently recompute x+y.

Live Variables

A variable is live at a program point p
if there is a path from p to any use of v
that does not contain a definition of v.

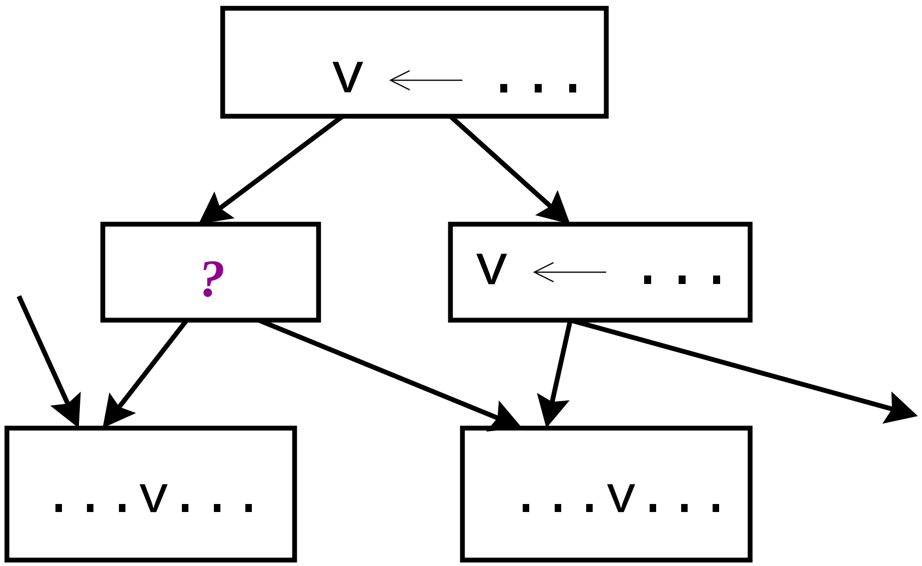
Flow problem: backward, bitvector (1 bit per use of a variable)



Upwards Exposed Uses

A use u of a variable v is upwards exposed at a program point p if there is a path from p to u that does not contain a definition of v.

Flow problem: backward, bitvector (1 bit per use of a variable)

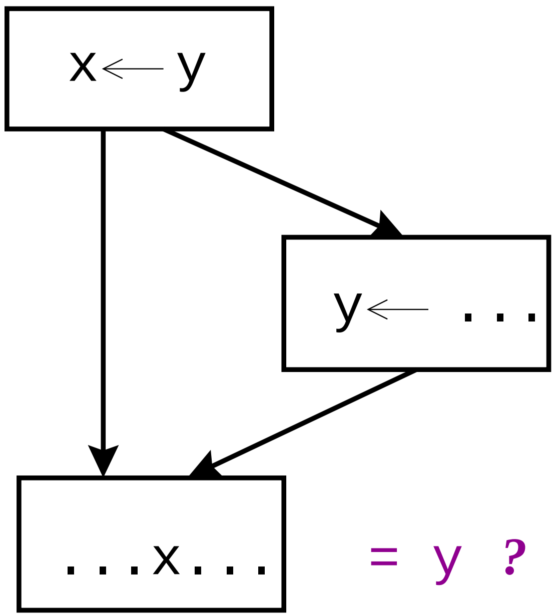


Copy Propagation Analysis

A copy statement $x \leftarrow y$ assigns variable y to v.

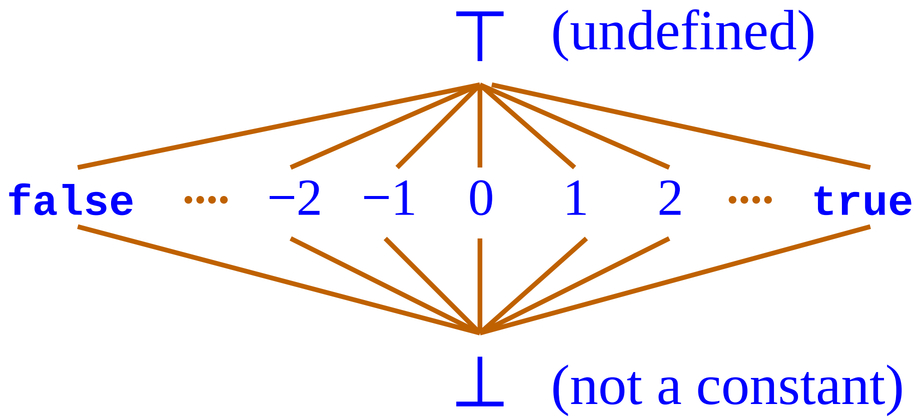
Can we safely replace all occurrences of x by y,
in order to eliminate the copy statement and variable x completely?

Flow problem: forward, bitvector (1 bit per copy assignment)

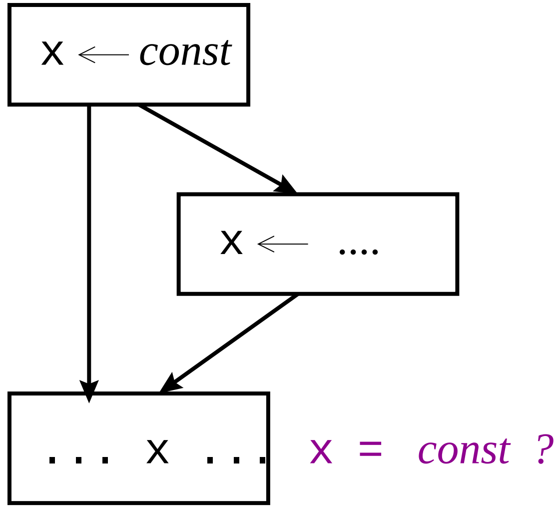


Constant Propagation Analysis

Flow problem: forward analysis, using ICP^n (or similar)
(1 lattice value per definition, symbolic execution)



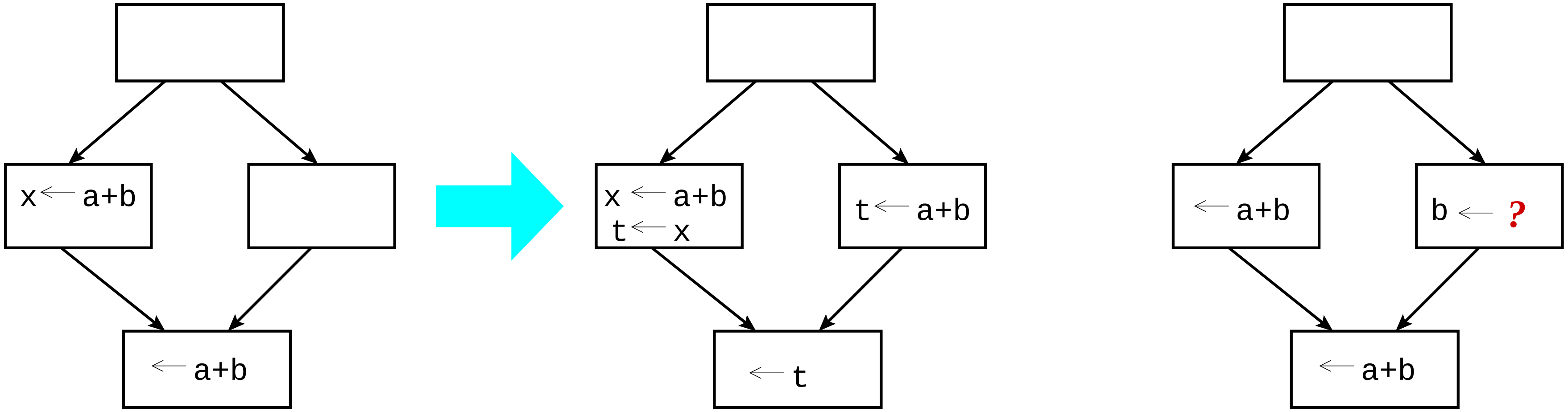
ICP:



Partial Redundancy Elimination

bidirectional,

bitvector: 1 bit per expression computation



[Morel,Renvoise'81] bidirectional, bitvector (1 bit per expression computation)

[Knoop/Rüthing/Steffen'92] "Lazy Code Motion"

[Dhamdhere'02] "PRE made easy"

DU chains, UD chains, Webs

sparse representation of dataflow information about variables:

- DU-chain connects a definition to all uses it may reach
- UD-chain connects a use to all definitions that may reach it

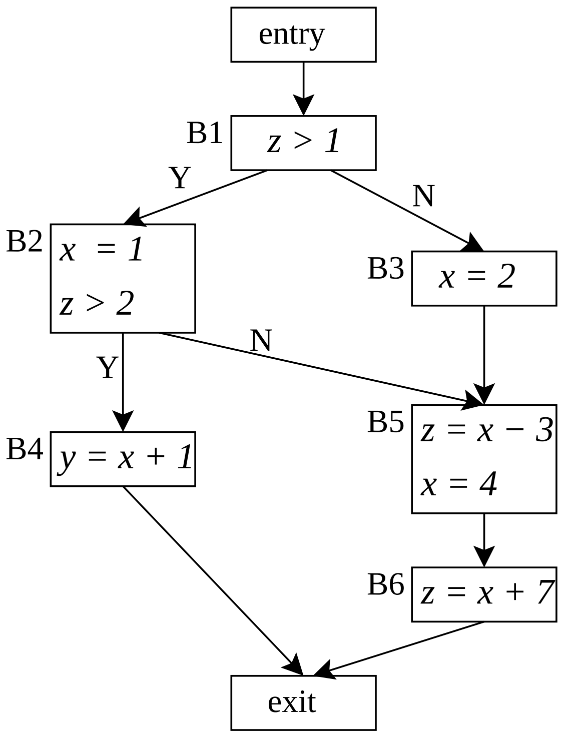
implemented as lists

Web for a variable v
= maximal union of intersecting DU-chains for v

useful in global register allocation (count as one live range)

DU, UD chains are implicitly given in SSA form ($\rightarrow$).

Web Construction Example



5 webs (sets of intersecting DU-chains):

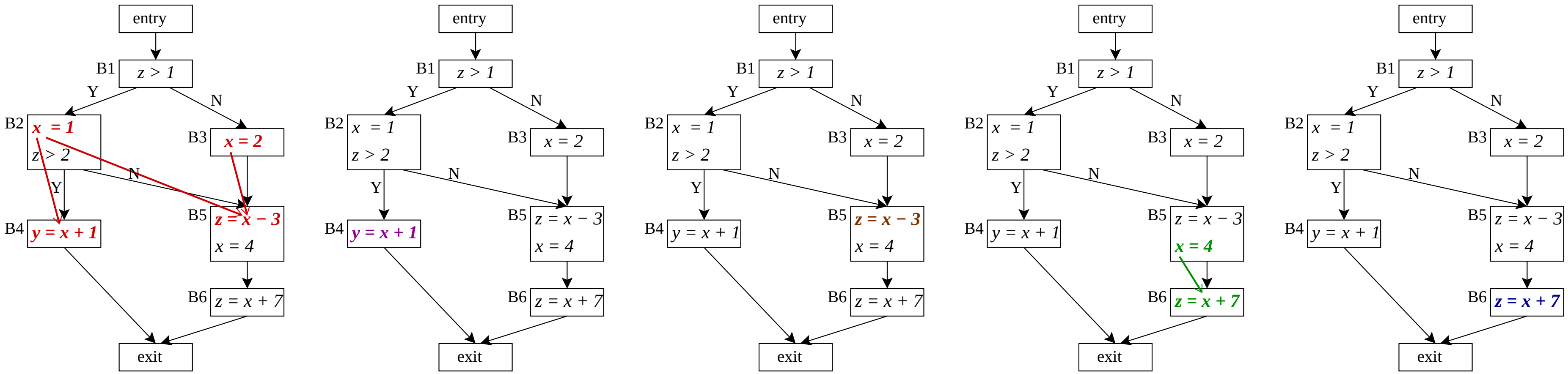
$$\{\ \langle\ \langle x,\ \langle B2,1\rangle\ \rangle,\ \{\ \langle B4,1\rangle,\ \langle B5,1\rangle\ \}\ \rangle,\ \langle\ \langle x,\ \langle B3,1\rangle\ \rangle,\ \{\langle B5,1\rangle\}\ \rangle\ \}$$

$$\{\langle\langle y,\langle B4,1\rangle\rangle,\emptyset\}$$

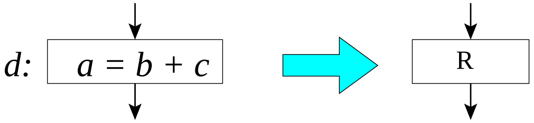
$$\{\langle\langle z,\langle B5,1\rangle\rangle,\emptyset\}$$

$$\{\langle\langle x,\langle B5,2\rangle\rangle,\ \{\langle B6,1\rangle\}\rangle\}$$

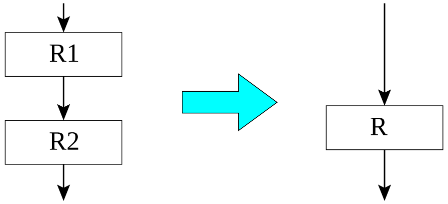
$$\{\langle\langle z,\langle B6,1\rangle\rangle,\emptyset\}$$



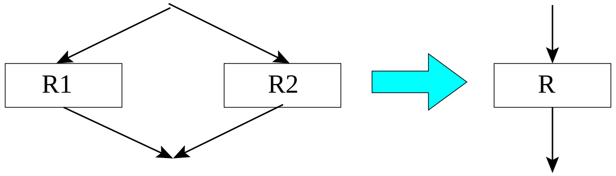
Structural Dataflow Analysis — Example: Reaching Definitions



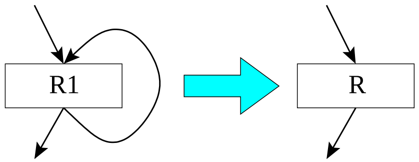
$$GEN(R) = \{d\}$$
$$KILL(R) = \{d_i : d_i \text{ defines } a\}$$
$$RDout(R) = GEN(R) \cup (RDin(R) - KILL(R))$$



$$GEN(R) = GEN(R2) \cup (GEN(R1) - KILL(R2))$$
$$KILL(R) = KILL(R2) \cup (KILL(S1) - GEN(R2))$$
$$RDin(R1) = RDin(R)$$
$$RDin(R2) = RDout(R1)$$
$$RDout(R) = RDout(R2)$$



$$GEN(R) = GEN(R1) \cup GEN(R2)$$
$$KILL(R) = KILL(R1) \cap KILL(R2)$$
$$RDin(R1) = RDin(R)$$
$$RDin(R2) = RDin(R)$$
$$RDout(R) = RDout(R1) \cup RDout(R2)$$



$$GEN(R) = GEN(R1)$$
$$KILL(R) = KILL(R1)$$
$$RDin(R1) = RDin(R) \cup GEN(R1)$$
$$RDout(R) = RDout(R1)$$

Data Flow Analysis: Summary

- Gather global information about data flow properties
- Safe under- / overestimation, depending on intended transformations
- Propagation over the CFG $\rightarrow$ iterative data flow analysis, implemented with the Worklist algorithm
- Lattice theory:
  Monotonicity + Finite height $\Rightarrow$ Termination of fixed-point iteration
- Various data flow problems and methods
- DU / UD chains, webs
- Structural dataflow analysis

Data Flow Analysis, further topics and outlook:

- Further DFA methods (interval / structural analysis)
- Array data flow analysis [Feautrier'91], [Maydan/Hennessy/Lam'91]
- DFA for pointers and heap data structures
- SSA form
- Generators for Data Flow Analyzers,
  e.g. Sharlit [Tjiang/Hennessy'92], PAG [Martin'98]