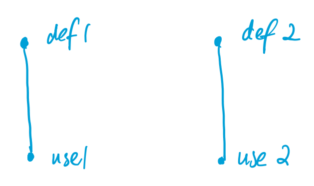
Static Single Assignment

Reaching definitions

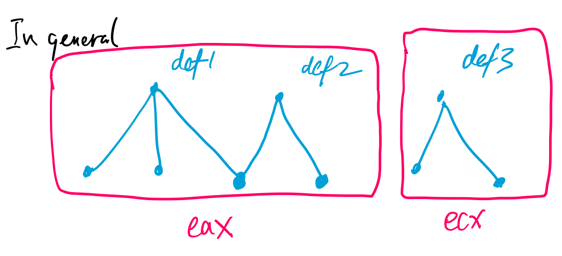
Let's do register allocation for this code:

```
i = 2          // def1 of i
...
i = i + 1      // def2 of i, use1 of i
...
a[i] = 0       // use2 of i
```



can use 2 regs to hold i
⇓
each def has smaller liveness range

In general



Computing reaching definitions: a dataflow analysis

What defs (set of assignment nodes) reach a use?

Conservative: overapproximate reaching defs

Every dataflow analysis is a five-tuple $(L, D, F_n, \sqcap, \top)$

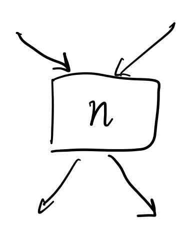
$l \in L$ out[n] is set of defs that reach the out edges of n.

$\sqsubseteq$ $\supseteq$

$\top$ $\emptyset$ height = |DefNodes (G)|

$\sqcap$ $\cup$ D = forward.

$$in[n] = \bigsqcap_{n' \prec n} out[n']$$



$$F_n(l) = gen[n] \cup (l - kill[n]) \qquad out[n] = F_n(in[n])$$

n	gen[n]	kill[n]
$x \leftarrow e$	$\{n\}$	all nodes defining x
everything else	$\emptyset$	$\emptyset$

Renumbering

Step 1: Unify defs that reach a common use (union-find)

for all CFG nodes n
  for all x ∈ uses[n]
    unify all defs of x reaching n (union-find)

Step 2: Rename variables

```
i1 = 2         // {def1} of i
...
i2 = i1 + 1    // {def2} of i, use1 of i
...
a[i2] = 0      // use2 of i
```

Modern compilers represent CFGs in SSA forms. E.g., LLVM uses an IR in SSA form.

SSA: statically, every variable has exactly one def

Easy to convert program to SSA form for straight-line programs

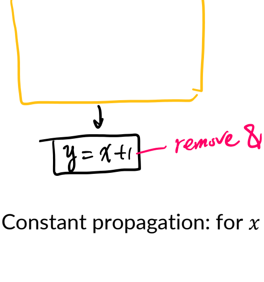
```
i1 = 2
...
i2 = i1 + 1
...
a[i2] = 0
```

Hard if control flow is more complex

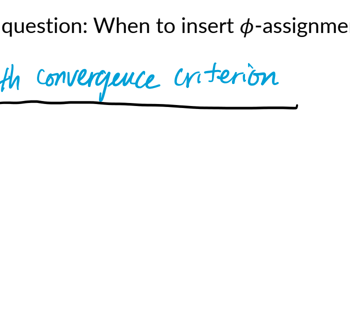
Why SSA?

Many analyses are greatly simplified by SSA.

- Reaching definition analysis



- Dead assignment elimination

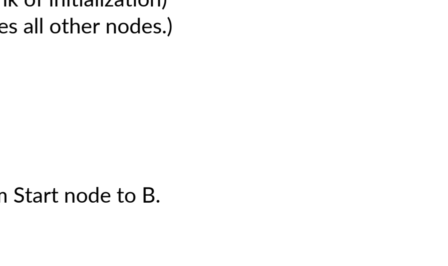


- Constant propagation: for $x \leftarrow c$, replace all uses of x by c

Convert CFG to SSA

Key question: When to insert ϕ-assignments?

Path convergence criterion



Criterion satisfied ⇒ insert $x \leftarrow \phi(x, x, \ldots, x)$ in the beginning of node m

Fine print:
- Start node of CFG considered to define all vars (think of initialization)
  (Simplifies domination analysis: Start node dominates all other nodes.)
- arity of ϕ = # predecessors of node m

Dominators

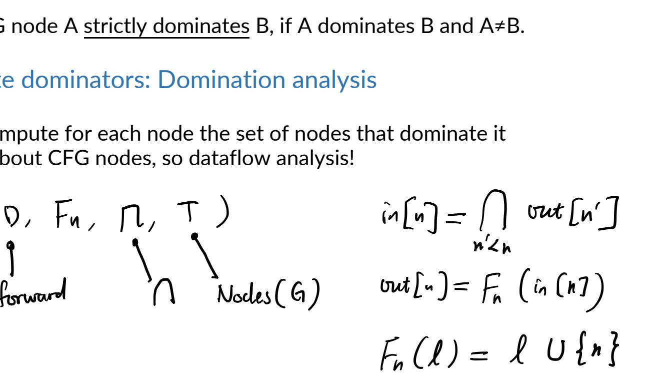
Def/ CFG node A <u>dominates</u> B, if A is on every path from Start node to B.
(To reach B, control must go through A.)

reflexive   A doms A

trans.   A doms B , B doms C ⇒ A doms C

anti-symmetric   A doms B , B doms A ⇒ A = B

A doms C , B doms C ⇒ A doms B ∨ B doms A.



<u>Def/</u> CFG node A <u>strictly dominates</u> B, if A dominates B and A≠B.

Compute dominators: Domination analysis

Want: compute for each node the set of nodes that dominate it
⇒ facts about CFG nodes, so dataflow analysis!

$(L, D, F_n, \sqcap, \top)$ D: forward, $\sqcap$: $\cap$, $\top$: Nodes(G)

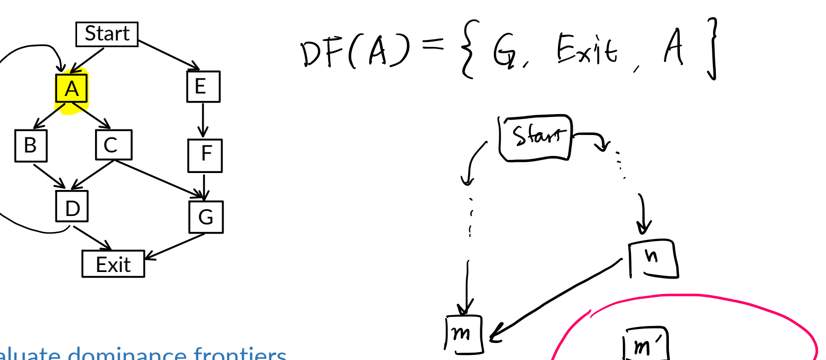
$$in[n] = \bigcap_{n' \prec n} out[n']$$

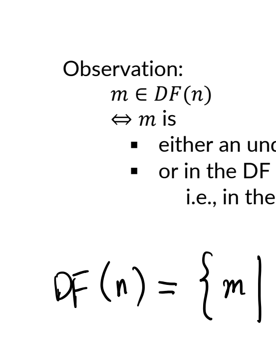
$$out[n] = F_n(in[n])$$

$$F_n(l) = l \cup \{n\}$$

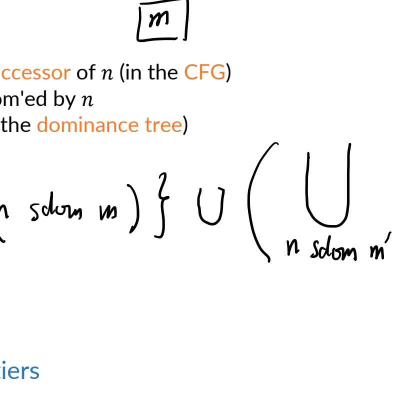
Dominance Frontier

Def/
  A CFG node m is in the <u>dominance frontier of node n</u>,
  if n dominates a predecessor of m, but n does not strictly dominate m.
  ≈ m is the first node on a path from n that is not strictly dominated by n
  ≈ border between dominated nodes and non-dominated nodes





$$DF(A) = \{G, Exit, A\}$$



Evaluate dominance frontiers

Observation:
  $m \in DF(n)$
  ⇔ m is
  - either an undominated, direct successor of n (in the CFG)
  - or in the DF of a node strictly dom'ed by n
    i.e., in the DF of child of n (in the dominance tree)

$$DF(n) = \{ m \mid n \prec m \wedge \neg(n \text{ sdom } m) \} \cup \left(\bigcup_{n \text{ sdom } m'} DF(m') \right)$$

Insert ϕ-assignments on dominance frontiers

n: a non-Start node containing a def of x

$m \in DF(n)$

SSA conversion: "Iterated dominance frontiers"

Let S be the set of nodes defining x.

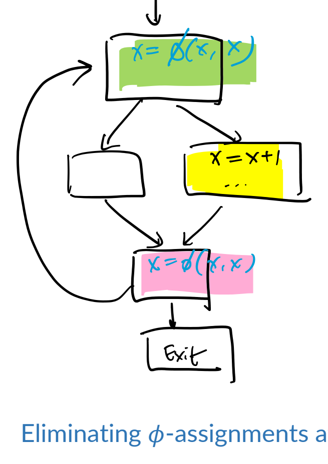
Want:

$$X = \bigcup_{n \in S \cup X} DF(n) \qquad \text{recursive}$$

$$DF_0 = \emptyset$$

$$DF_{i+1} = \bigcup_{n \in S \cup DF_i} DF(n)$$

Ex/ SSA-convert def of x



Eliminating ϕ-assignments at code generation time

ϕ-assignments are fictitious: they should not exist at run time

To translate away $x \leftarrow \phi(x_1, \ldots, x_n)$, insert $x \leftarrow x_i$ at the end of the i-th predecessor block.