Compiler Construction

Single Static Assignment

Preliminary remark

Almost all data flow analysis simplify when variables are defined once.

$\Rightarrow$ No kills in dataflow analysis

Single Static Assignment intuition

> *A program is defined to be in SSA form if each variable is a target of exactly one assignment statement in the program text.*

Idea Behind SSA

- Start with CFG
- Give each definition a fresh name
- Propagate fresh name to subsequent uses

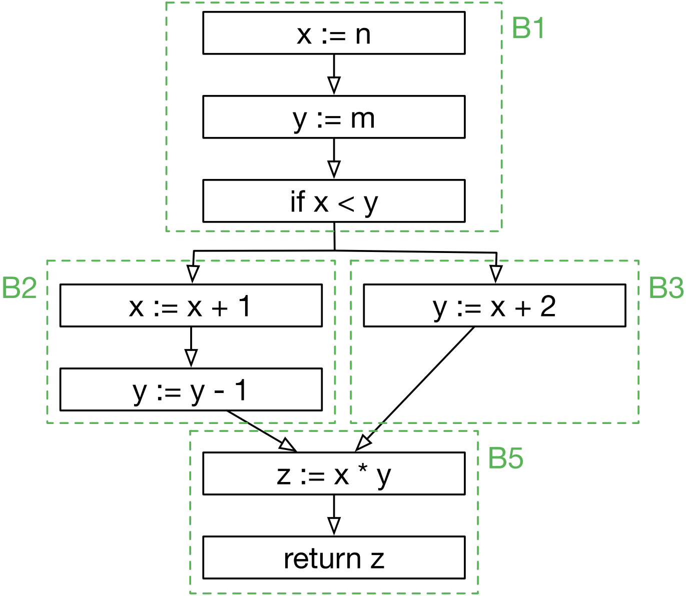
```
x  := n
y  := m
x  := x + y
return x
```

No SSA

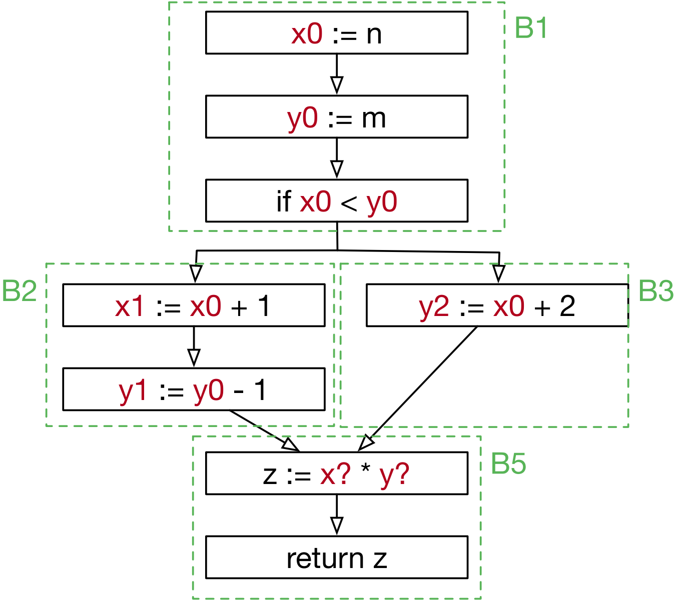
```
x0  := n
y0  := m
x1  := x0 + y0
return x1
```

SSA

Problem with control flow merges (1/2)



Problem with control flow merges (1/2)

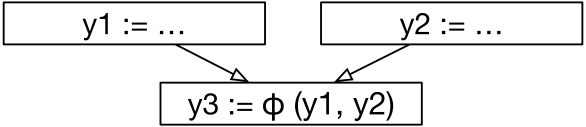


The Solution

- Introduce a notational fiction called a ϕ-function
- This ϕ-function can combine multiple definitions coming from multiple basic blocks

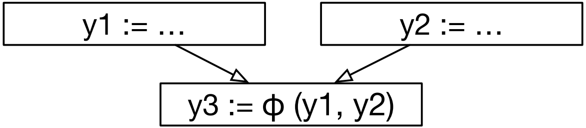
The Solution

- Introduce a notational fiction called a ϕ-function
- This ϕ-function can combine multiple definitions coming from multiple basic blocks



The expression **y3 := ϕ(y1, y2)** means that y3 will hold either the value of y1 or the value of y2 (depending on the execution).

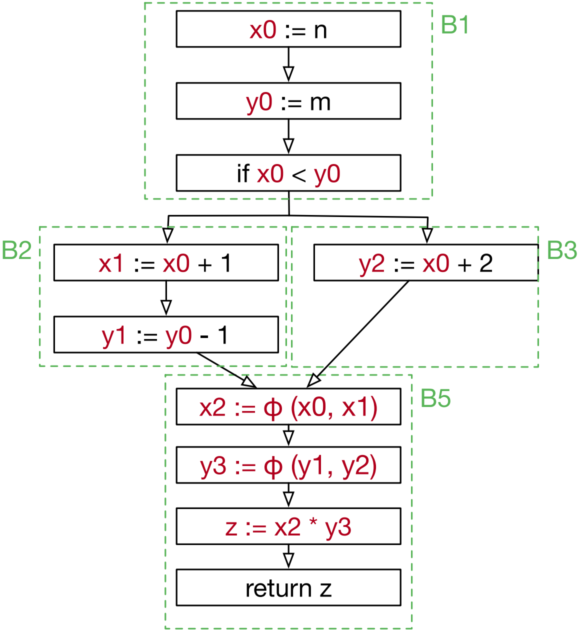
Remark



How does the ϕ-function know which edge was taken?

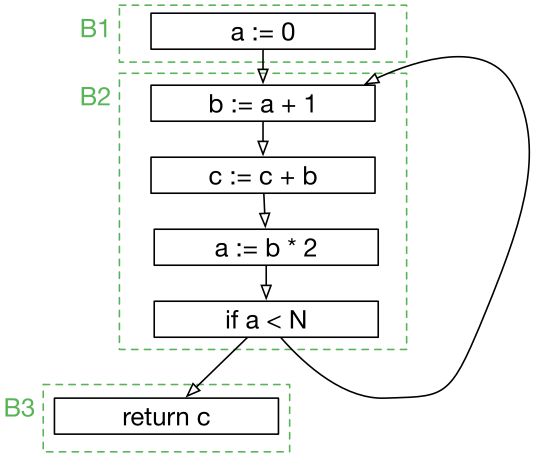
We can "implement" the ϕ-function using a MOVE on each/every incoming edge.

Back to the example

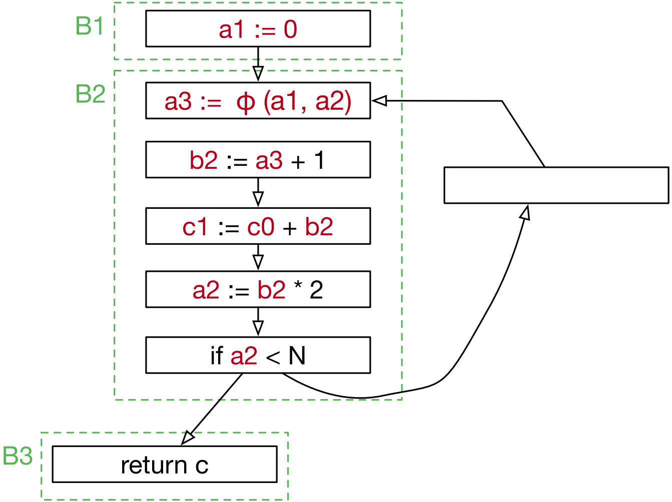


A word on loops (1/2)

How to handle loops



A word on loops (2/2)



CFG to SSA, Naively

1. Insert phi nodes in each basic block except the start node
2. Calculate the dominator tree
3. Traverse the dominator tree in a breadth-first fashion:
   - give each definition of x a fresh index
   - propagate that index to all of the uses

Remarks

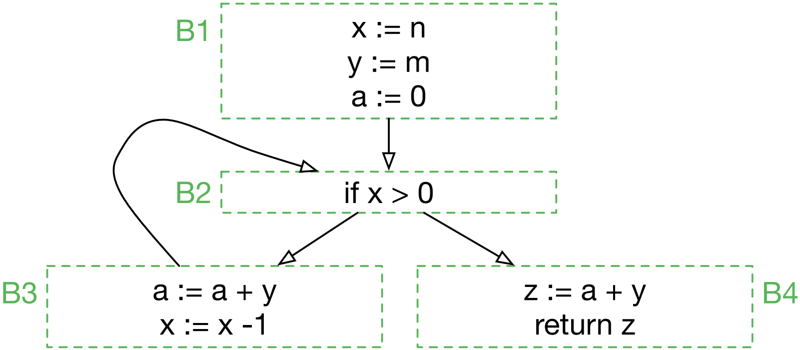
About ϕ-node insertion

Could limit insertion to nodes with more than 1 predecessor

About index-propagation

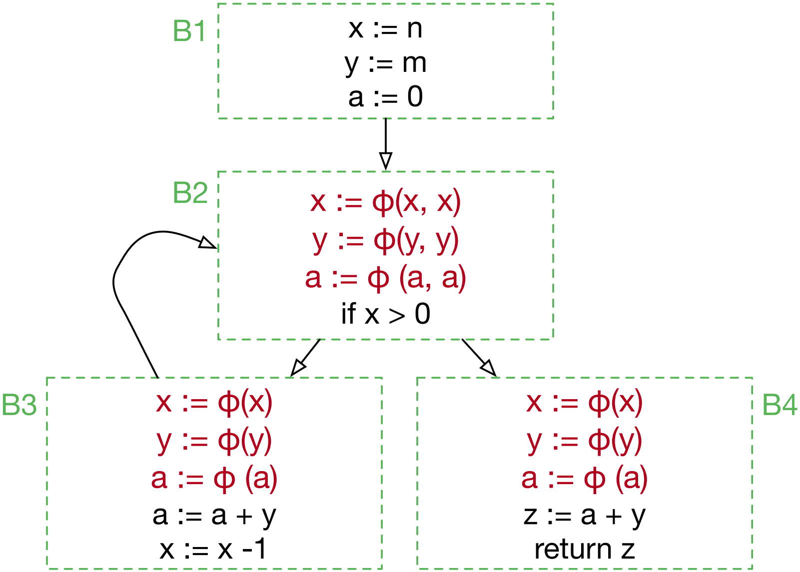
- Propagate to each use of x that is not killed by a subsequent definition.
- Propagate the last definition of x to the successors' phi nodes

Example

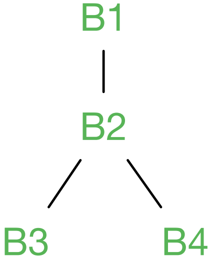


Only basic block are represented for clarity

Insert ϕ-nodes



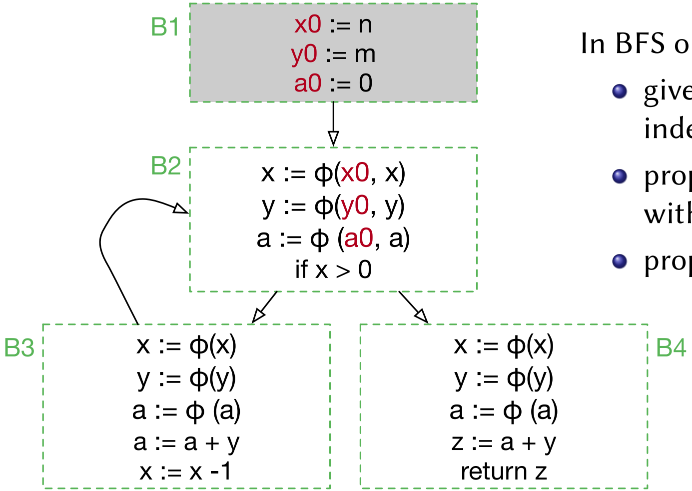
Compute Dominators



A node d dominates a node n if every path of directed edges from the initial state (s_0) to n must go through d. Can be computed with DFS or equations.

$$
\begin{aligned}
\mathrm{D}[s_0] &= \{s_0\} \\
\mathrm{D}[n] &= \{n\} \cup \Big(\bigcap_{p \in \mathrm{pred}[n]} \mathrm{D}[p] \Big)
\end{aligned}
$$

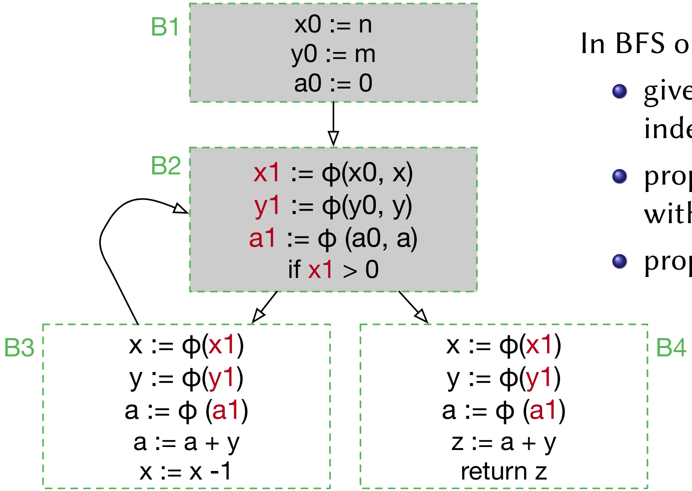
Processing B1



In BFS order:

- give each definition of var a fresh index
- propagate that index to each use within block
- propagate to successor's phi node

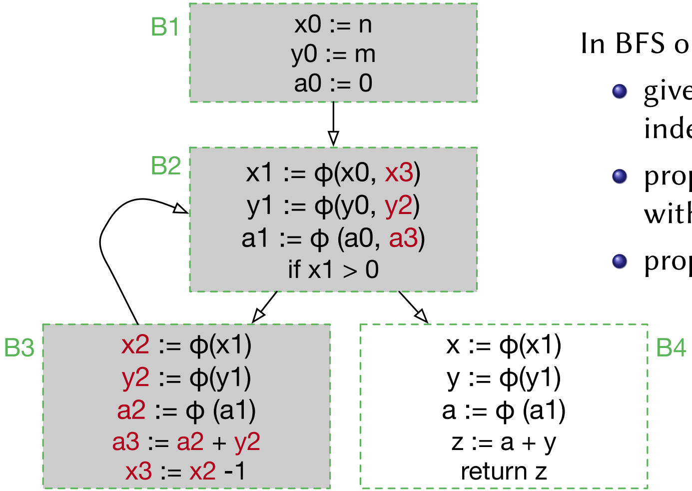
Processing B2



In BFS order:

- give each definition of var a fresh index
- propagate that index to each use within block
- propagate to successor's phi node

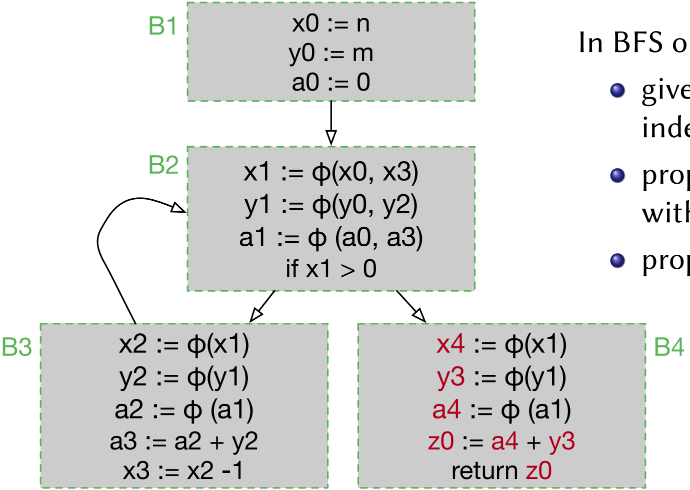
Processing B3



In BFS order:

- give each definition of var a fresh index
- propagate that index to each use within block
- propagate to successor's phi node

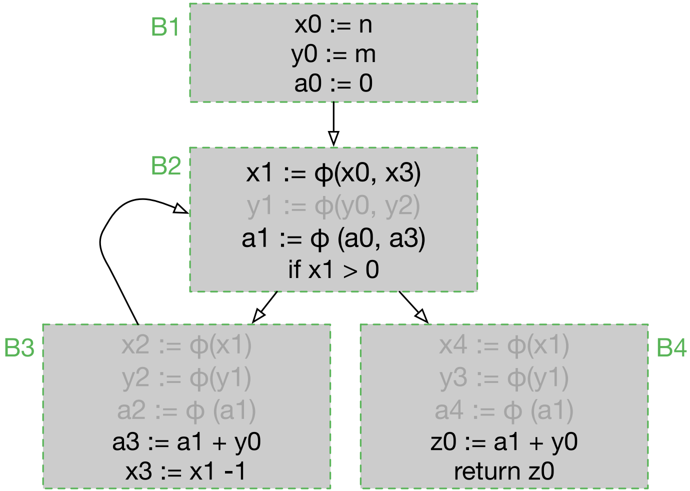
Processing B4



In BFS order:

- give each definition of var a fresh index
- propagate that index to each use within block
- propagate to successor's phi node

Clean up using copy propagation and dead code elimination



Smarter Algorithm for CFG to SSA

Definition

The dominance frontier of n is the set of all nodes w such that

- n dominates a predecessor of w
- n does not strictly dominate w

1. Compute the dominance frontier
2. Use dominance frontier to place phi nodes
   - Whenever block n defines x, put a phi node for x in every block in the dominance frontier of n
3. Do renaming pass using dominator tree

Summary