CSC2/455 Software Analysis and Improvement
Dominators and SSA Form

Sreepathi Pai

Feb 5, 2025

URCS

Outline

Review

Dominator Analysis (DOM)

SSA Form

Postscript

Outline

Data flow analysis framework

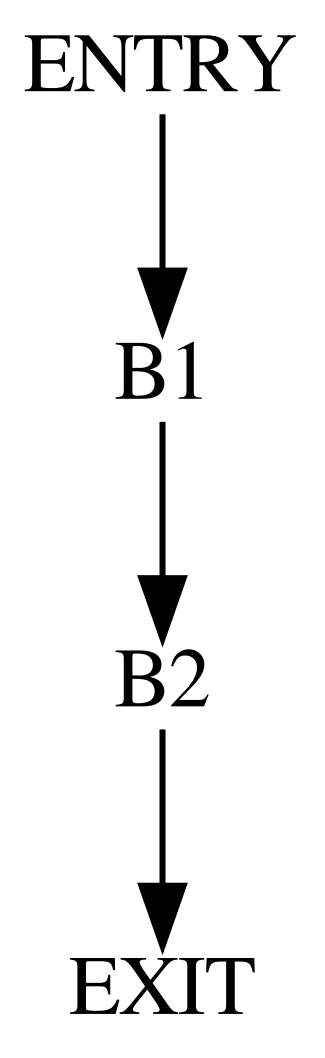
- Live variable analysis
  - "Is there a read of this variable along any path?"
- Reaching Definitions
  - "Which definitions reach this use?"
- Available expressions
  - "Is this expression calculated previously and the result still usable?"
- Very Busy Expressions
  - "Are there expressions that can be precalculated?"
- Iterative data flow analysis
  - GEN, KILL, Transfer functions, Initialization

Outline

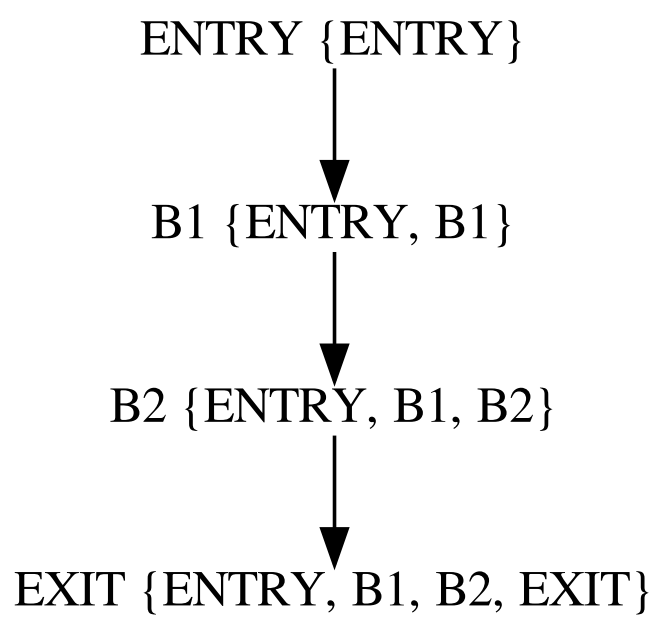
Dominators

- A node n in the CFG dominates a node m iff:
  - n is on all paths from entry to m
  - by definition, a node n always dominates itself
- Dominators are a property of graphs
  - I.e. has nothing to do with code in basic blocks

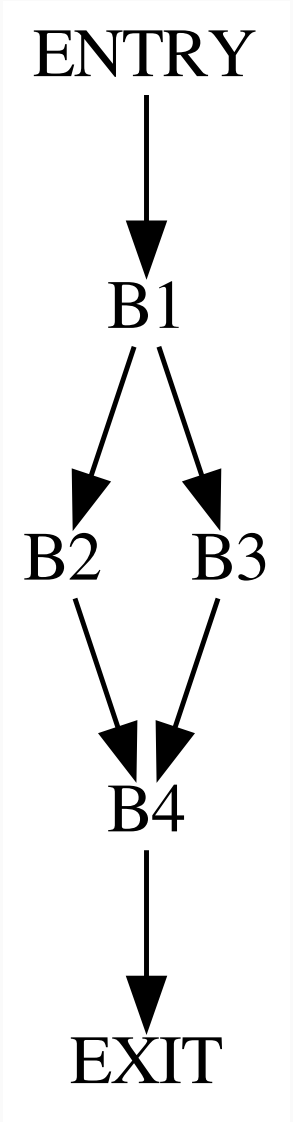
Example 1: Node with single predecessor



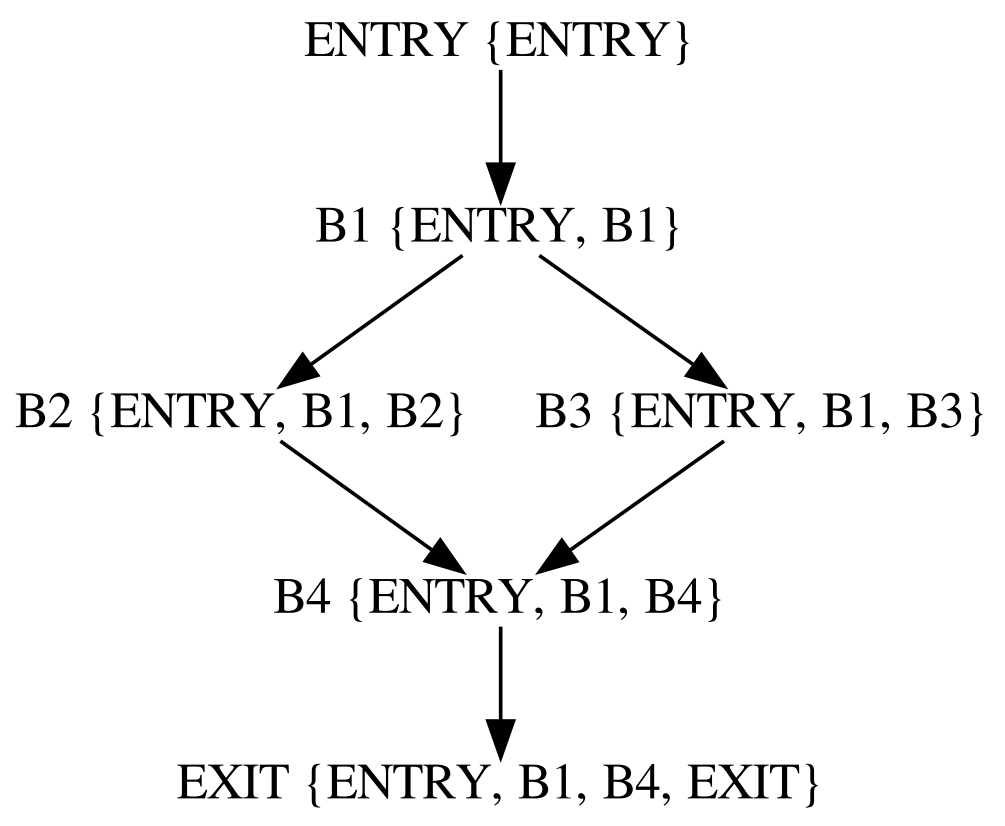
Example 1: Node with single predecessor (Answer)



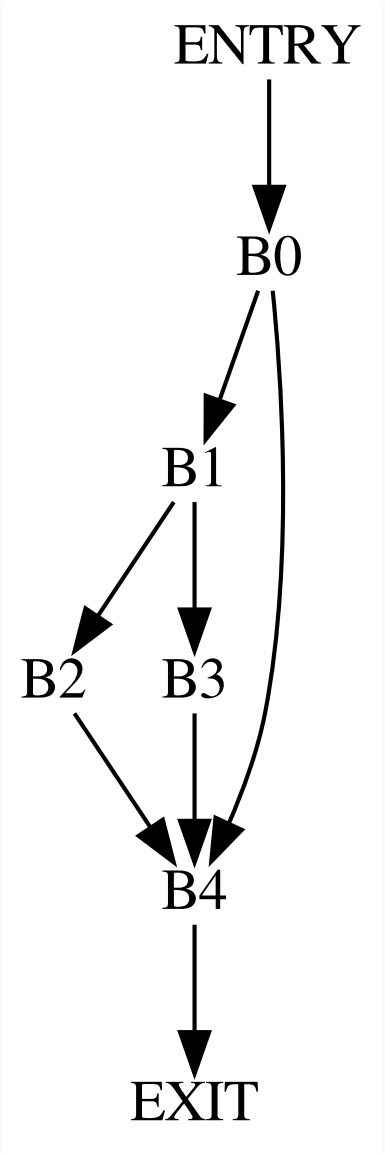
Example 2: Node with multiple predecessors



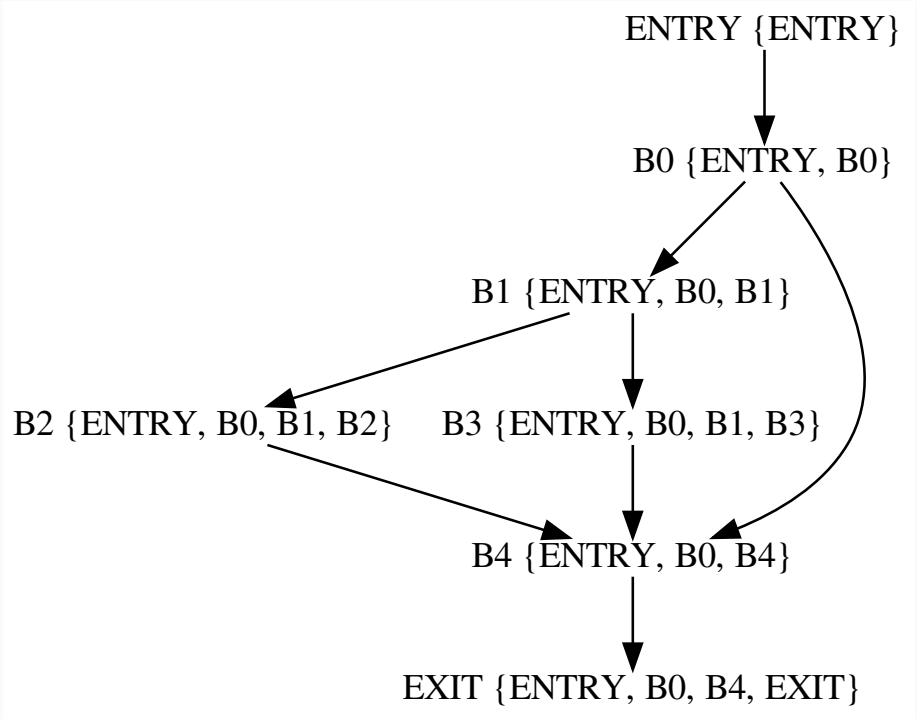
Example 2: Node with multiple predecessors (Answer)



Example 3: Slightly more involved example



Example 3: Slightly more involved example (Answer)



Today

Can we use data flow analysis to identify the dominators of a node?

Data flow analysis setup

- Domain of facts?
- GEN and KILL?
- Direction of analysis?
- Merge operator?
- Initialization?

Data flow analysis Equation

$$\mathrm{DOM}(n) = \{n\} \cup (\cap_{m \in \mathrm{pred}(n)} \mathrm{DOM}(m))$$

- Initialization
  - (for $n \neq \mathrm{ENTRY}$): $\mathrm{DOM}(n) = N$ (where N is the set of all nodes)
  - (for $n = \mathrm{ENTRY}$): $\mathrm{DOM}(n) = \mathrm{ENTRY}$

Related concept: Post-dominators

A node m is post-dominated by a node n iif:

- n appears on every path from m to EXIT.
- n post-dominates itself, by definition

Outline

Static Single Assignment (SSA) Form

- Intermediate Representation
  - Similar to 3 address code
- Each variable only written once
  - Static [in source] Single [once] assignment
- SSA form can be generated from 3 address code
  - Introduce ϕ functions
  - Rename variables

Example 1: Straight-line code

```
y = x + 1;
x = 2;
y = x + y + 2;
```

gets transformed to:

```
y_0 = x_0 + 1
x_1 = 2;
y_1 = x_1 + y_0 + 2;
```

From this example, when should we rename variables?

Example 2: Branches

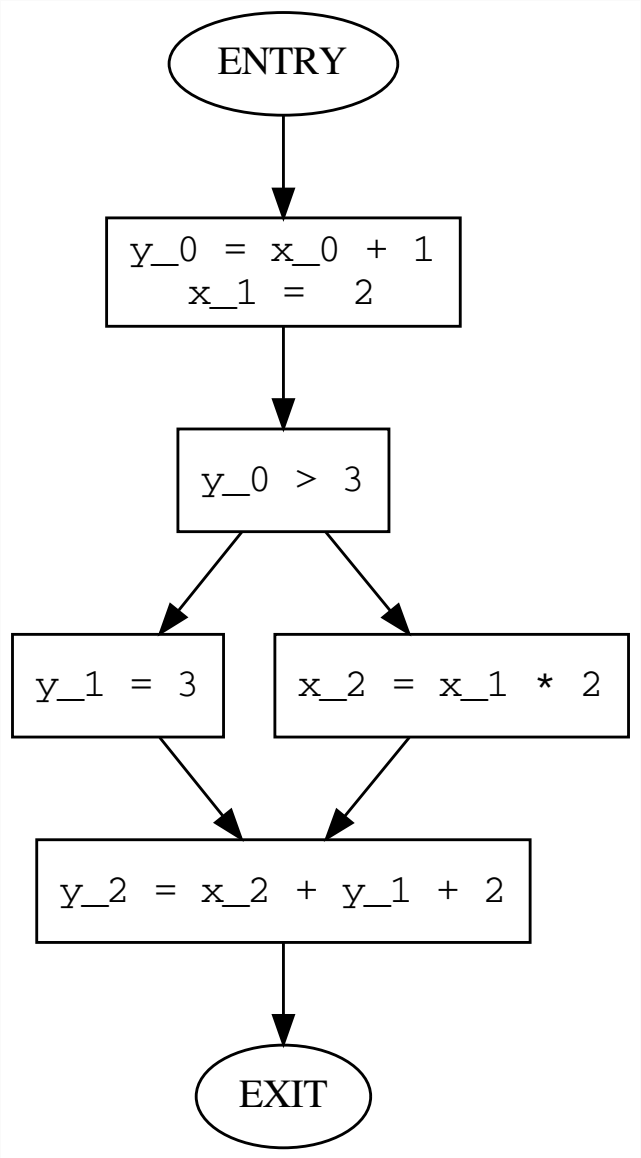
```
y = x + 1;
x = 2;

if(y > 3)
   y = 3;
else
   x = x * 2;

y = x + y + 2;
```

gets transformed to:

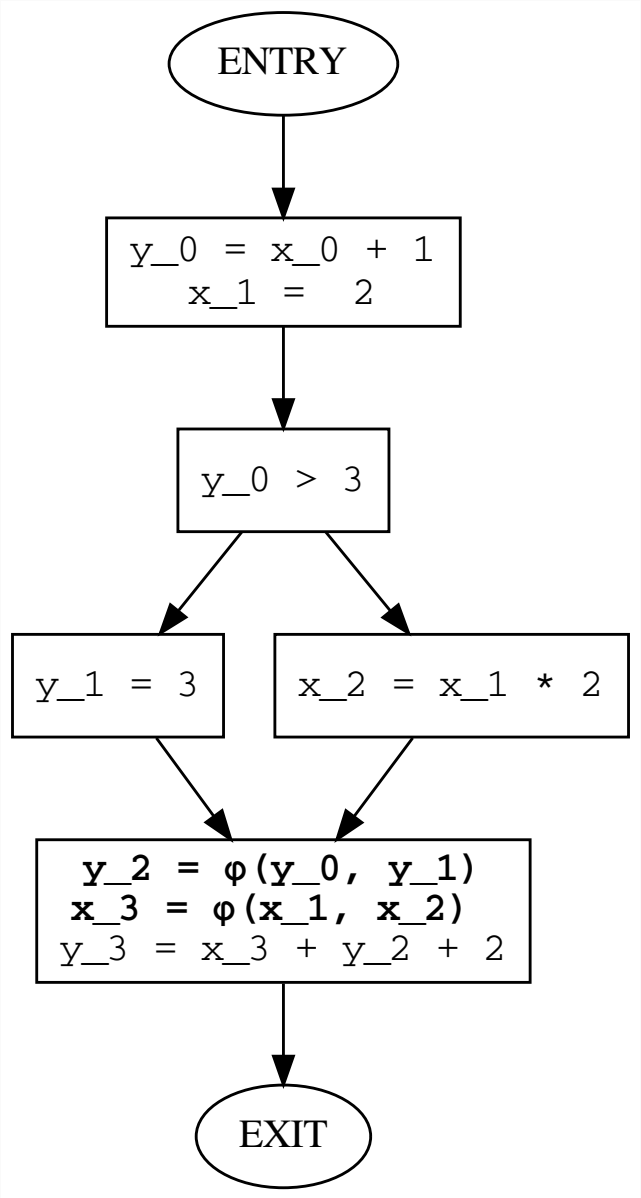
```
y_0 = x_0 + 1
x_1 = 2;

if(y_0 > 3)
   y_1 = 3;
else
   x_2 = x_1 * 2;

y_2 = x_2 + y_1 + 2;
```

Is this renaming correct?

Example 2: The CFG



Example 2: Fix using ϕ functions



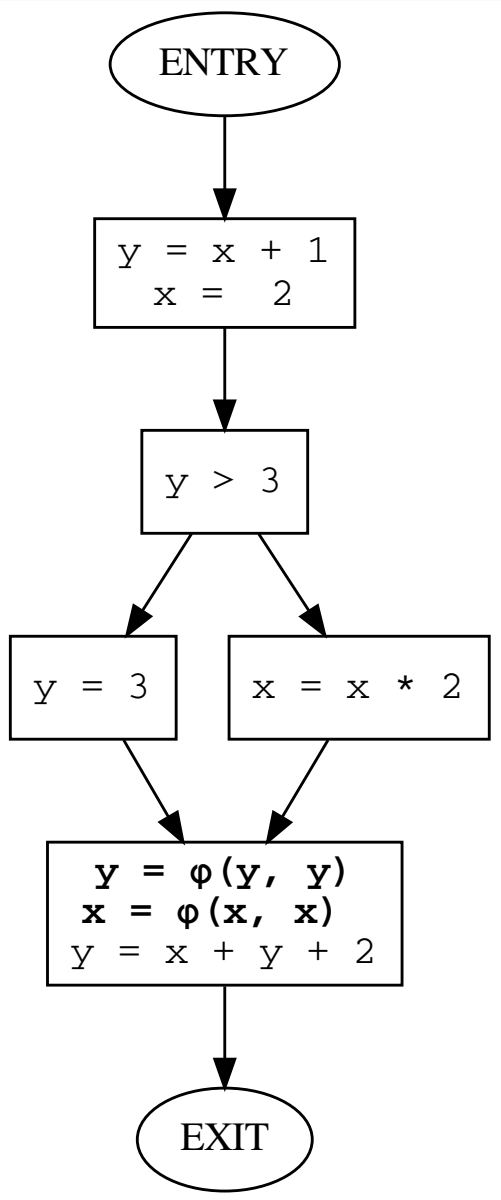
Simple Algorithm for constructing SSA form: 1

- Insert ϕ functions
  - In which nodes of CFG?
  - For which variables?
- Rename variables
  - To what?
  - Helps to think of LHS (definition) renames and RHS (use) renames

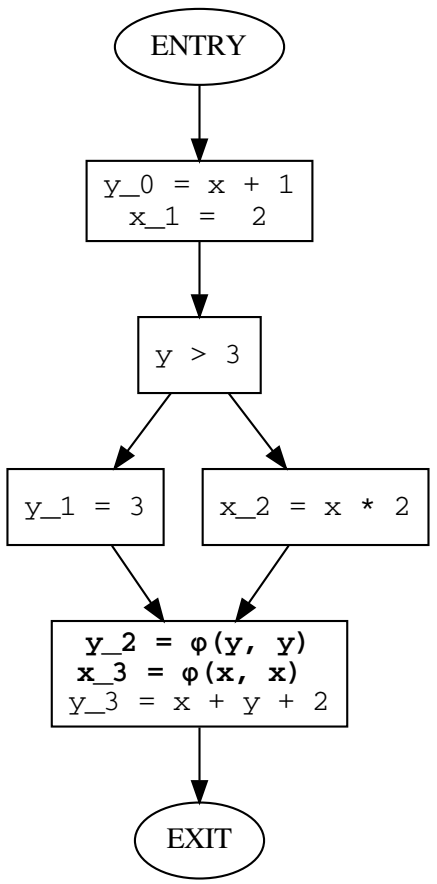
Simple Algorithm for constructing SSA form: 2

- Insert ϕ functions
  - In join nodes, before all other code
  - For *all* variables defined or used in procedure
  - Each ϕ function has one argument per incoming edge
  - Use $y = \phi(y, y)$ form for variable y
- Rename variables
  - To what?
  - Helps to think of LHS (definition) renames and RHS (use) renames

Simple Algorithm for constructing SSA form: 3



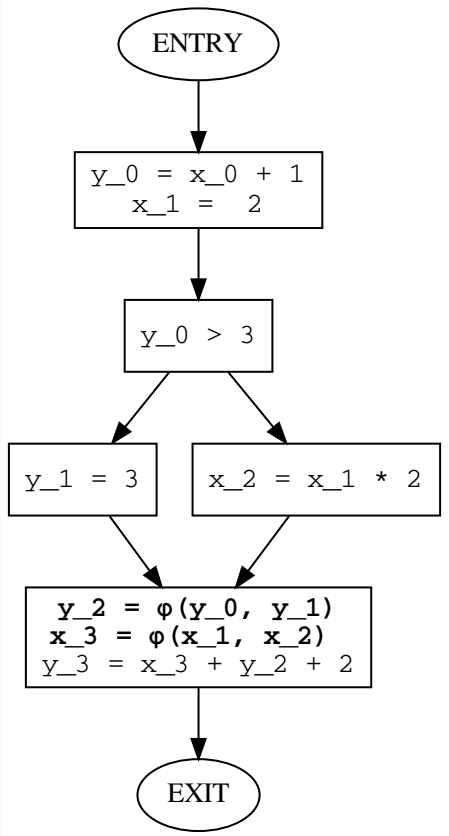
Simple Algorithm for constructing SSA form: Rename LHS



Simple Algorithm for constructing SSA form: Rename RHS

- Note that in SSA form, only one definition reaches a use (except the uses in ϕ)
- The arguments to ϕ are the definitions that reach it

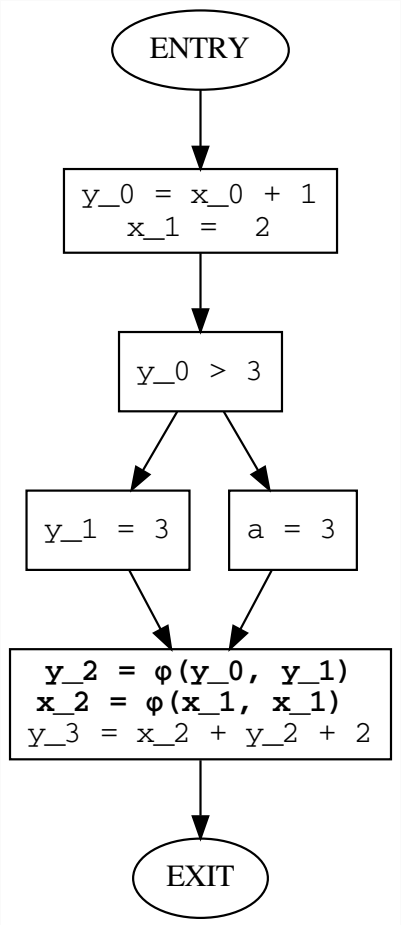
Simple Algorithm for constructing SSA form: Rename RHS



Simple Algorithm for constructing SSA form: Renaming

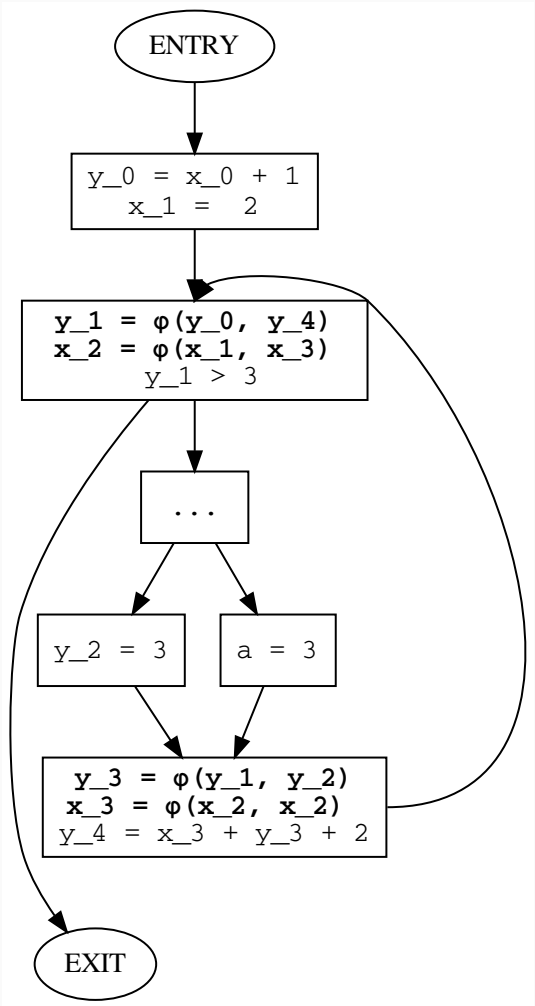
- In actual compilers, renaming LHS and RHS can be done by simply calculating reaching definitions
  - Remember we had to track each definition there too (recall `y#0`)
- This construction is called the *maximal SSA form*
  - Simple to construct
  - Wasteful, can introduce too many ϕ functions (not in our example)

Example: Redundant ϕ functions



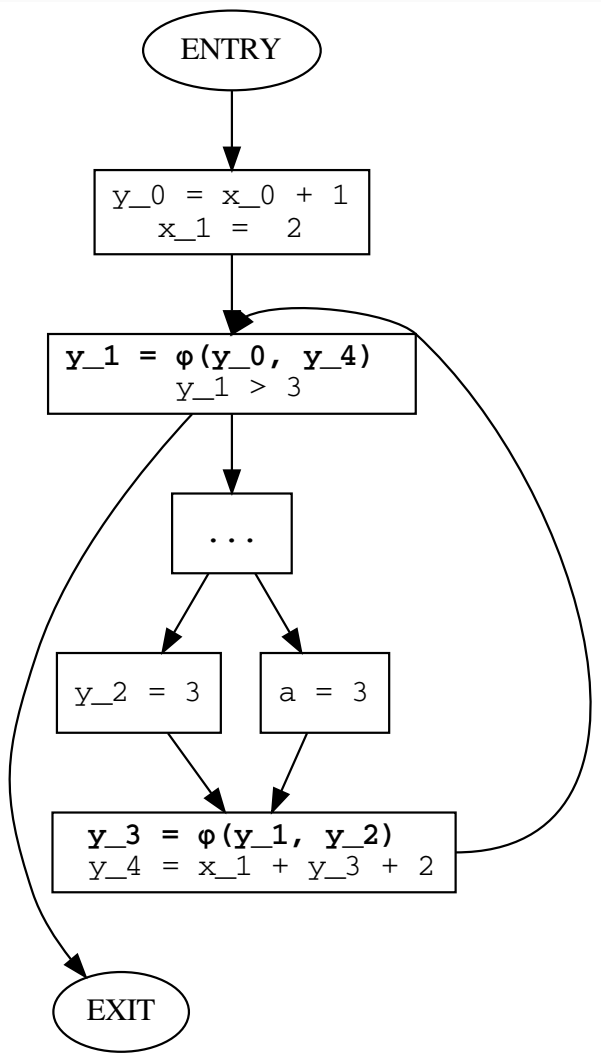
Here, our method constructs a redundant ϕ function for x_2.

Example: Redundant ϕ functions (now with loops)



Here, x_3 is redundant, and its removal makes x_2 redundant.

Example: Non-redundant ϕ functions



This gets rid of the redundant ϕ functions.

Outline

References

- Chapter 9 of Cooper and Turczon
  - Section 9.2.1
  - Section 9.3