CSE 401 – Compilers

Lecture 24: SSA, cont.

Michael Ringenburg

Winter 2013

Reminders

- Project Part 4 due on Friday, March 15.
- There will be a short project report due on Sunday, March 17 – at most **one** late day may be used for the report (if you have any left).
  - One-two pages
  - See posted assignment
- Guest lectures Wednesday and Friday
  - Wednesday: Real world parsing, David Mizell, YarcData (Cray)
  - Friday: Register allocation, Preston Briggs, UW/PNNL

Review: SSA Form

```
if (…)
  a = x;
else
  a = y;
b = a;
```

→

```
if (…)
  a1 = x;
else
  a2 = y;
a3 = Φ(a1, a2);
b1 = a3;
```

- An IR where each variable has exactly one definition
  - Within a basic block, this is easy – simply need to create a new variable (by renumbering) at each definition.
  - At program merge points, add Φ-functions

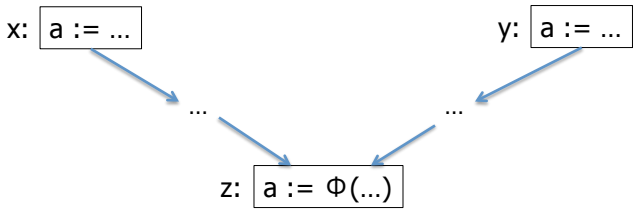
Converting To SSA Form

- Basic idea
  - First, add Φ-functions
  - Then, rename all definitions and uses of variables by adding subscripts
- Renaming is straightforward. Inserting Φ-functions is where things get a little tricky.

Inserting Φ-Functions

- Could simply add Φ-functions for every variable at every join point
- But
  - Wastes way too much space and time
  - Not needed

When to Insert a Φ-Function



- We need a Φ-function for variable a at entry to block z whenever
  - There are blocks x and y, both containing definitions of a, and $x \neq y$
  - There are nonempty paths from x to z and from y to z
  - These paths have no common nodes other than z
    - i.e., this is where the paths first merge

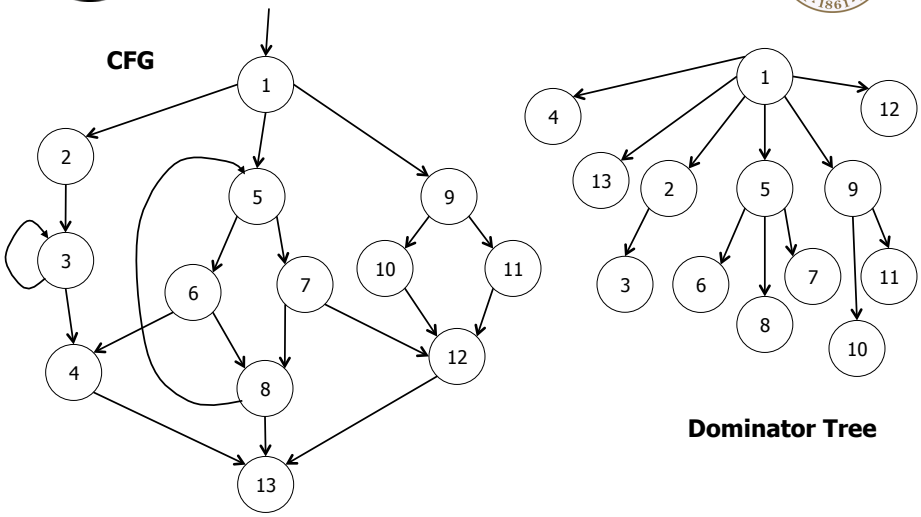
Some Details

- The start node of the control flow graph is considered to define every variable (possibly just to Undefined)
  - Makes following construction simpler
- Each Φ-function itself defines a variable, which may create the need for a new Φ-function.
  - So we need to keep adding Φ-functions until things converge (no more changes).
- How do we do this efficiently?
  - Using a new concept: dominance

Dominators

- Definition
  - A block x *dominates* a block y if and only if every path from the entry of the control-flow graph to y includes x
- By definition, x dominates x
- We can associate a Dom(inator) set with each CFG node
  - The set of all basic blocks that must execute before x
  - | Dom(x) | ≥ 1
- Properties:
  - Transitive: if a dom b and b dom c, then a dom c
  - No cycles, thus can view dominators a tree

Example



Dominators and SSA

- Important property of SSA: definitions must dominate uses
  - In other words, the single assignment must occur prior to any uses of the variable. (Although that single assignment may just be the start node assignment of "Undefined".)
- More specifically:
  - If x := $\Phi(\ldots,x_i,\ldots)$ in block n, then the definition of x_i dominates the i^{th} predecessor of n
  - If x is used in a non-Φ statement in block n, then the definition of x dominates block n

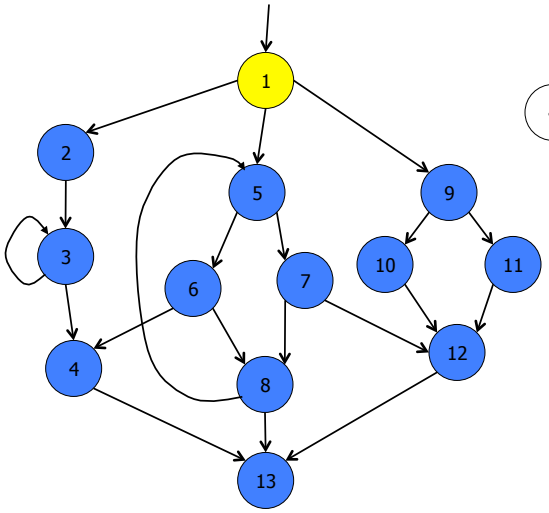
Dominance Frontier (1)

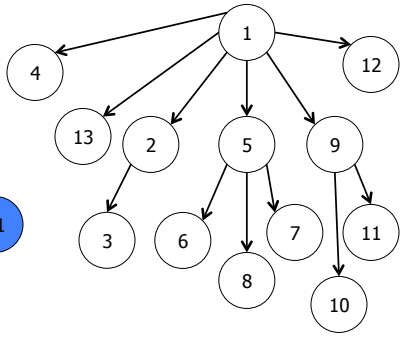
- To get a practical algorithm for placing Φ-functions, we need to avoid looking at all combinations of nodes leading from x to y
- Instead, use the dominator tree in the flow graph.
  - Place merges *just beyond the end of the definitions' dominance*.
    - The first point where they may receive a value from an alternate definition.
  - This follows directly from the previous properties:
    - Φ-function means predecessors are dominated by defs
    - Non Φ usage means dominated by def
  - This is referred to as the *dominance frontier*.

Dominance Frontier (2)

- Definitions
  - x *strictly dominates* y if x dominates y and $x \neq y$
  - The *dominance frontier* of a node x is the set of all nodes w such that x dominates a predecessor of w, but x does not *strictly* dominate w
    - Interestingly, this means that x can be in *it's own dominance frontier!* This can happen if you have a back edge to x (x is the head of a loop).
- Essentially, the dominance frontier is the border between dominated and undominated nodes

Example



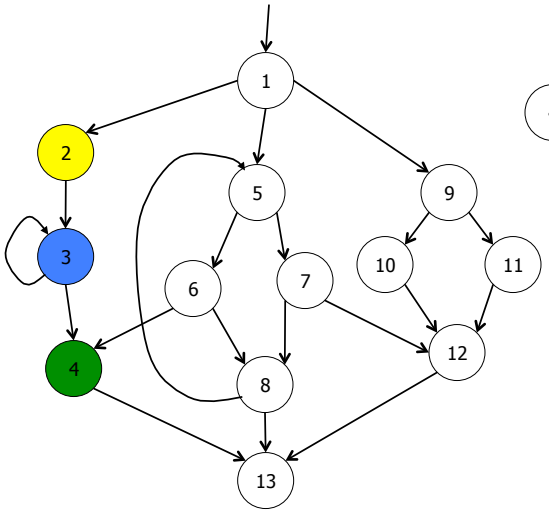


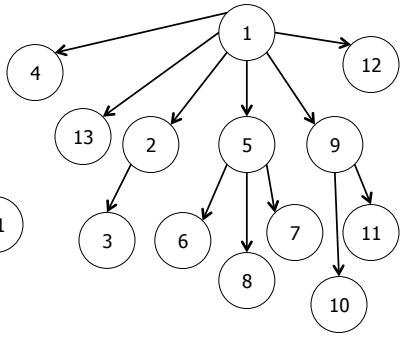
= x

= DominanceFrontier(x)

= StrictDom(x)

Example



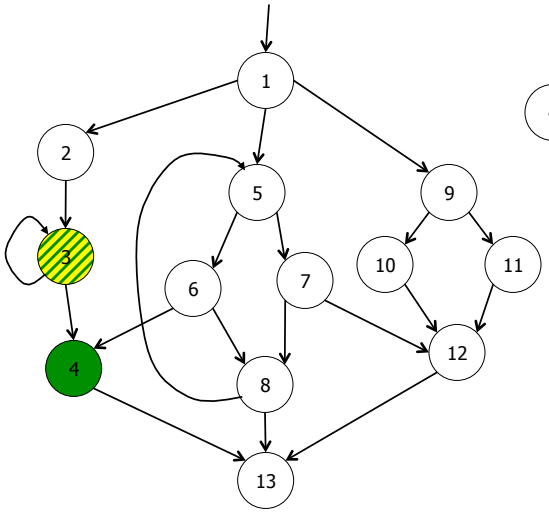


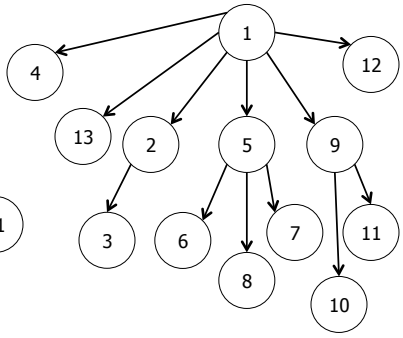
= x

= DominanceFrontier(x)

= StrictDom(x)

Example



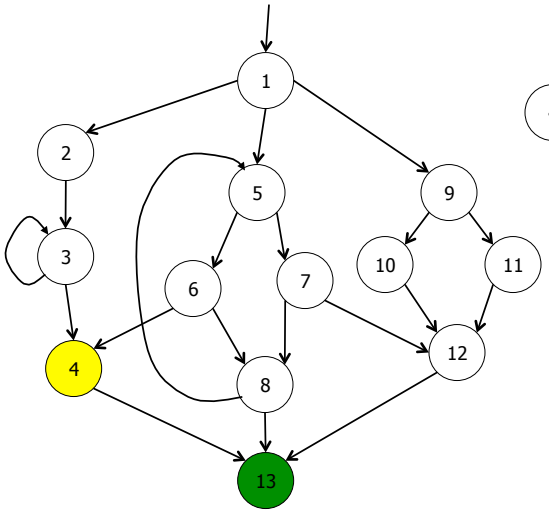


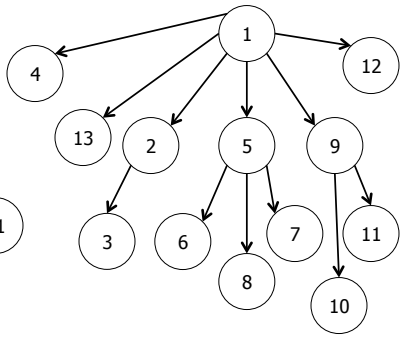
= x

= DominanceFrontier(x)

= StrictDom(x)

Example



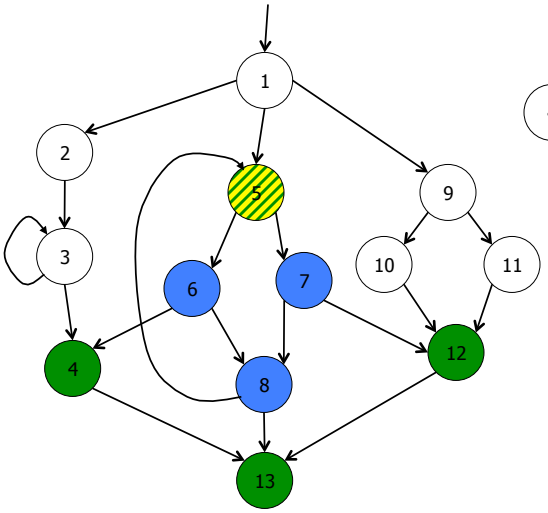


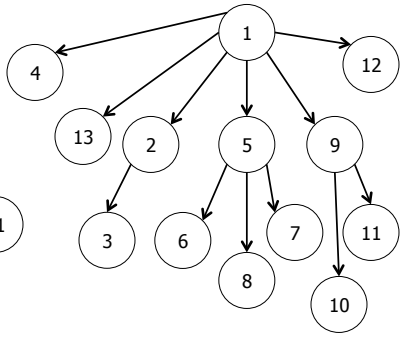
= x

= DominanceFrontier(x)

= StrictDom(x)

Example



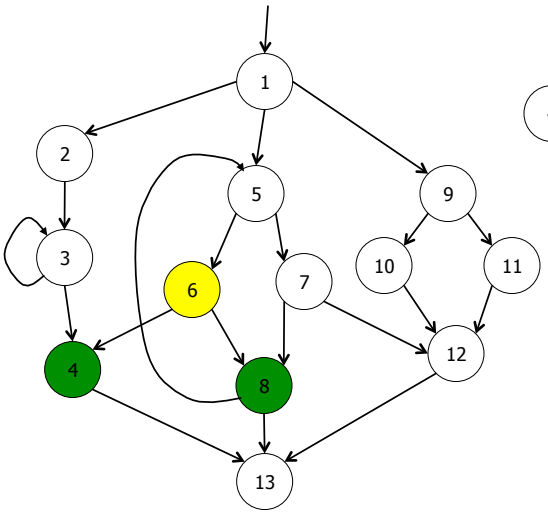


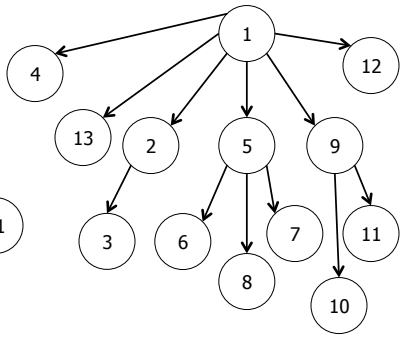
= x

= DominanceFrontier(x)

= StrictDom(x)

Example



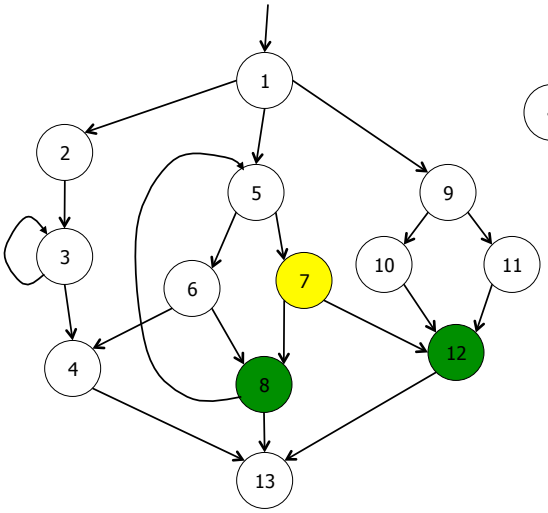


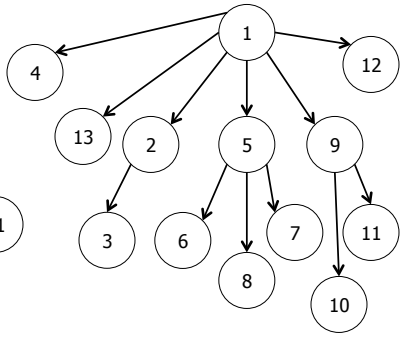
= x

= DominanceFrontier(x)

= StrictDom(x)

Example



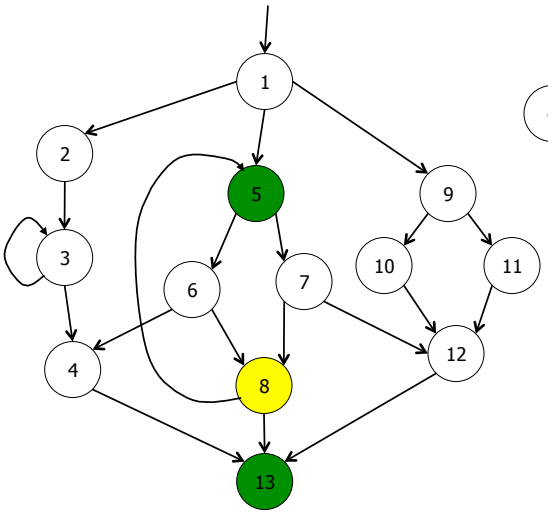


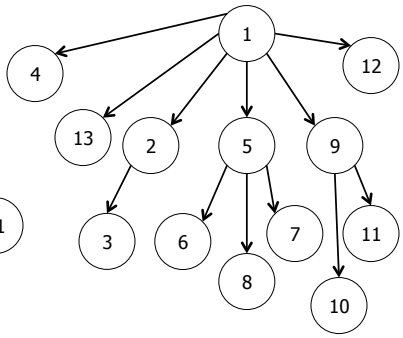
= x

= DominanceFrontier(x)

= StrictDom(x)

Example



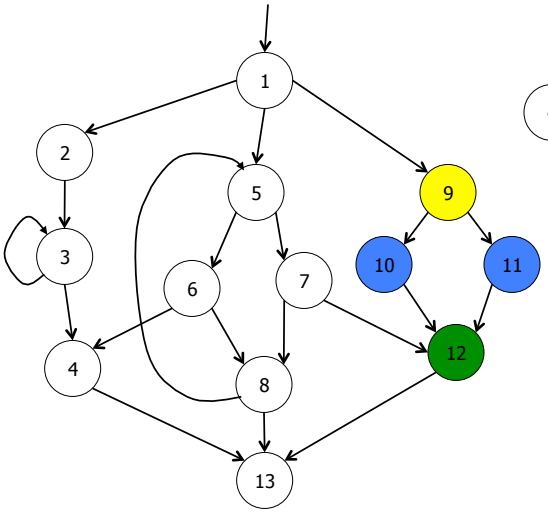


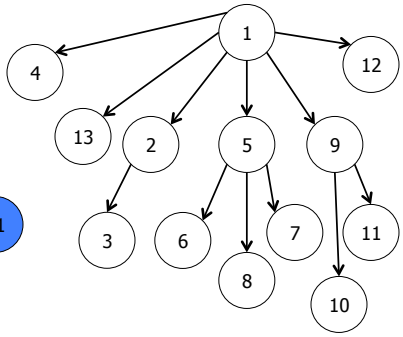
= x

= DominanceFrontier(x)

= StrictDom(x)

Example



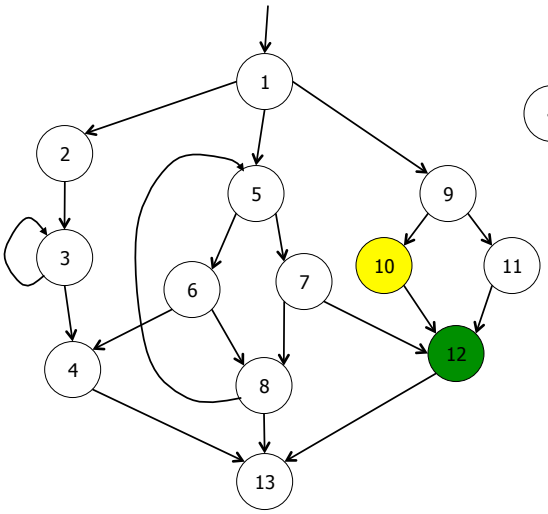


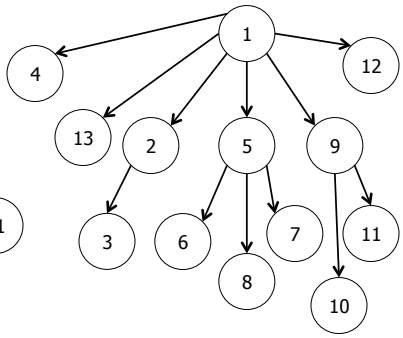
= x

= DominanceFrontier(x)

= StrictDom(x)

Example



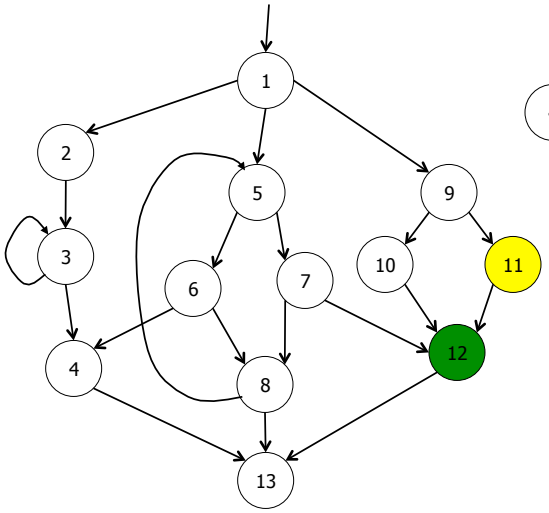


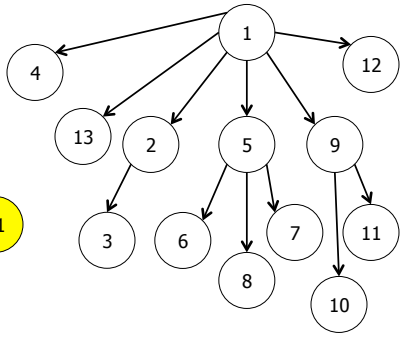
= x

= DominanceFrontier(x)

= StrictDom(x)

Example



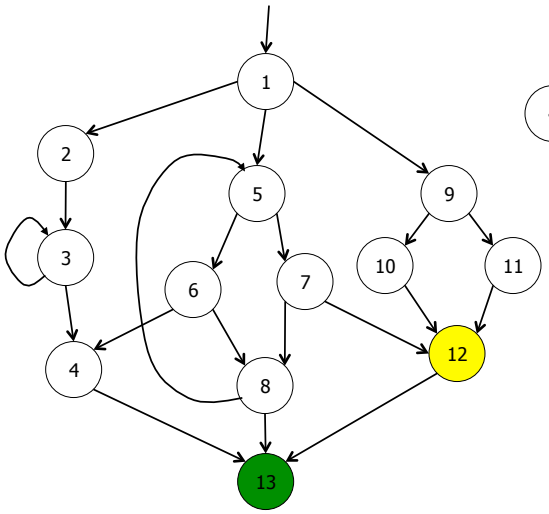


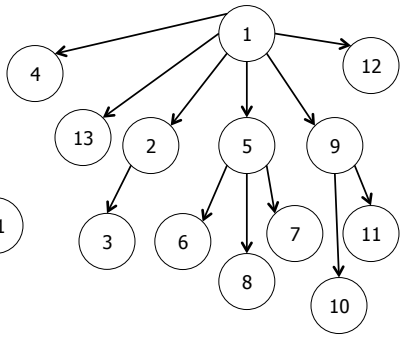
= x

= DominanceFrontier(x)

= StrictDom(x)

Example



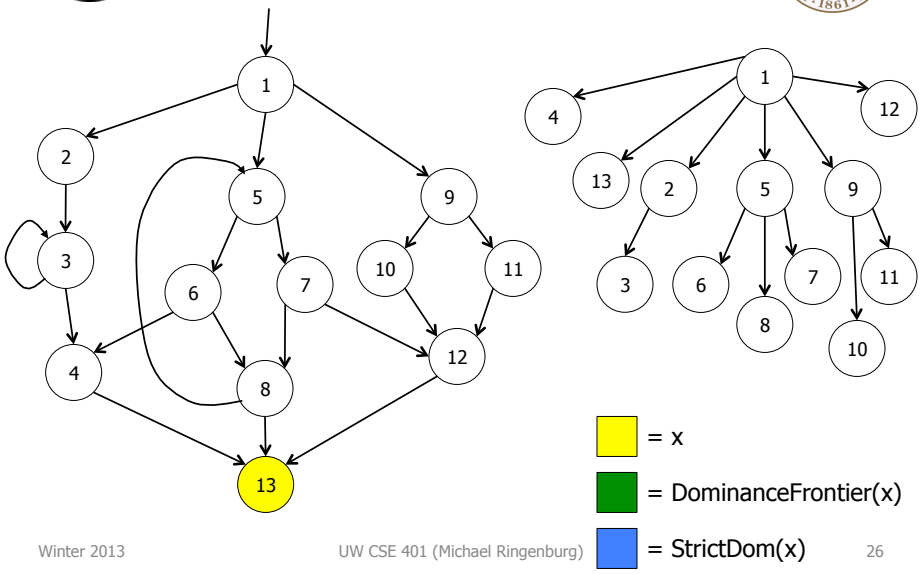


= x

= DominanceFrontier(x)

= StrictDom(x)

Example



Placing Φ-Functions

- If a node *x* contains the definition of variable *a*, then every node in the dominance frontier of *x* needs a Φ-function for *a*
  - Idea: Everything dominated by x will see x's definition. Dominance frontier represents first nodes we could have reached via an alternate path, which *will* have an alternate reaching definition (recall that the entry defines everything).
    - Why does this work for loops? Hint: Strict dominance …
  - Since the Φ-function itself is a definition, this needs to be iterated until it reaches a fixed-point
- Theorem: this algorithm places exactly the same set of Φ-functions as the path criterion given previously.

Placing Φ-Functions: Details

- We won't give the full constructions here (see your text). The basic steps are:
  1. Compute the dominance frontiers for each node in the control flow graph
  2. Insert just enough Φ-functions to satisfy the criterion. Use a worklist algorithm to avoid reexamining nodes unnecessarily
  3. Walk the dominator tree and rename the different definitions of variable a to be a_1, a_2, a_3, …

SSA Optimizations

- Advantage of SSA: Makes many optimizations and analyses simpler and more efficient.
  - We'll show a couple examples.
- But first, what do we know? (i.e., what information is kept in the SSA graph?)

SSA Data Structures

- Statement: links to containing block, next and previous statements, variables defined, variables used.
- Variable: link to its (single) definition statement and (possibly multiple) use sites
- Block: List of contained statements, ordered list of predecessors, successor(s)

Dead-Code Elimination

- A variable is live if and only if its list of uses is not empty(!)
  - Without SSA, possibly many stores to each variable. Have to disambiguate which might be used. With SSA each store defines a new variable, so this becomes trivial …
- Algorithm to delete dead code:

  while there is some variable v with no uses

  if the statement that defines v has no other side effects, then delete it
  - Need to remove this statement from the list of uses for its *operand variables* – which may cause those variables to become dead

Sparse Simple Constant Propagation (SSCP)

- If c is a constant in v := c, any use of v can be replaced by c
  - Then update every use of v to use constant c
- If the c_i's in v := $\Phi(c_1, c_2, \ldots, c_n)$ are all the same constant c (or "Undefined" via start node, if you like), we can replace this with v := c
- Can also incorporate copy propagation, constant folding, and others in the same worklist algorithm

Sparse Simple Constant Propagation

```
W := list of all statements in SSA program
while W is not empty
  remove some statement S from W
  if S is v:=Φ(c, c, …, c), replace S with v:=c
  if S is v:=c
     delete S from the program
     for each statement T that uses v
       substitute c for v in T
       add T to W
```

Converting Back from SSA

- Unfortunately, real machines do not include a Φ instruction
- So after analysis, optimization, and transformation, need to convert back to a "Φ-less" form for execution

Translating Φ-functions

- The meaning of $x := \Phi(x_1, x_2, \ldots, x_n)$ is "set $x := x_1$ if arriving on edge 1, set $x := x_2$ if arriving on edge 2, etc."
- So, for each i, insert $x := x_i$ at the end of predecessor block i
- Rely on copy propagation and coalescing in register allocation to eliminate redundant moves

SSA

- There are many details needed to fully and efficiently implement SSA, but these are the main ideas
  - Your text has some more details
- SSA is used in most modern optimizing compilers & has been retrofitted into many older ones (gcc is a well-known example)

Course Evaluations

- You all know the drill by now. Doing them today because rest of this week will be busy.
  - Two guest lectures (Wednesday and Friday)
- Any volunteers to collect and send them in?
  - Or I can always volunteer someone ☺