Advanced Compilers

CMPSCI 710
Spring 2003
Computing SSA

Emery Berger
University of Massachusetts, Amherst

More SSA

- Last time
  - dominance
  - SSA form
- Today
  - Computing SSA form

Criteria for Inserting f Functions

- Brute-force: Insert one ϕ function for each variable at every *join point* (point in CFG with more than one predecessor)
  - Wasteful
- When should we insert ϕ function for a variable **a** at node **z** of the CFG?
  - Intuitively: add ϕ if there are two definitions of **a** that can reach the point **z** through distinct paths

Path Convergence Criterion [Cytron-Ferrante '89]

Insert ϕ function for variable **a** at node **z** if **all** the following conditions are true:

1. There is a block **x** that defines **a**
2. There is a block **y** ≠ **x** that defines **a**
3. There are non-empty paths **x** ® **z** and **y** ® **z**
4. Paths **x** ® **z** and **y** ® **z** don't have nodes in common other than **z**
5. The node **z** does not appear within both **x** ® **z** and **y** ® **z** prior to the end, but it may appear in one or the other.

Note: The start node contains an implicit definition of every variable.

Iterated Path-Convergence Criterion

The ϕ function itself is a definition of **a**.
Therefore the path-convergence criterion is a set of equations that must be satisfied.

```
while there are nodes x, y, z satisfying conditions 1-5
      and z does not contain a ϕ function for a
do insert a ← ϕ(a_0, a_1, ..., a_n) at node z
```

This algorithm is extremely costly, because it requires the examination of every triple of nodes **x**, **y**, **z** and every path from **x** to **z** and from **y** to **z**.

Can we do better?

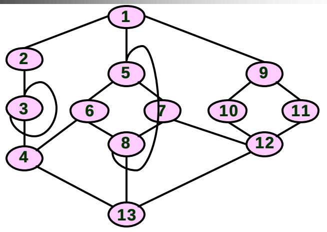
Dominance Reviewed

- Node **x** dominates node **w** if every path from the start node to **w** must go through **x**
- Node **x** strictly dominates node **w** if **x** dominates **w** and **x** ≠ **w**

Dominance Property of SSA Form

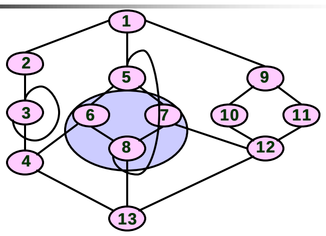
- In SSA form, definitions dominate uses:
  - If **x** is used in ϕ function in block **n**
    - def of **x** dominates every predecessor of **n**
  - If **x** is used in non-ϕ statement in block **n**
    - def of **x** dominates **n**
- The dominance frontier of node **x**:
  - nodes **w** such that **x** dominates predecessor of **w**, but **x** does not strictly dominate **w**

Example



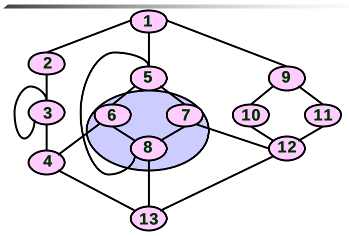
What is the dominance frontier of node 5?

Example



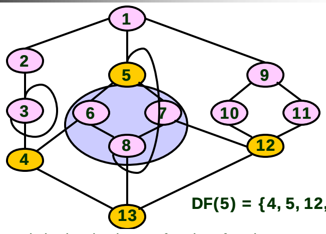
First we must find all nodes that
node 5 strictly dominates.

Example



A node **w** is in the dominance frontier of node 5
if 5 dominates a predecessor of **w**, but 5 does not strictly
dominate **w** itself. What is the dominance frontier of 5?

Example



DF(5) = {4, 5, 12, 13}

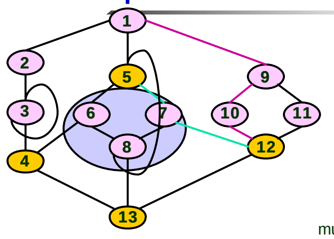
A node **w** is in the dominance frontier of node 5
if 5 dominates a predecessor of **w**, but 5 does not strictly
dominates **w** itself. What is the dominance frontier of 5?

Dominance Frontier Criterion

Dominance Frontier Criterion:
If a node **x** contains a definition of variable **a**, then any node **z** in the dominance frontier of **x** needs a ϕ function for **a**.

Can you think of an intuitive explanation
for why a node in the dominance frontier
of another node must be a join node?

Example



If a node (12) is in the dominance frontier of another node (5), than there must be at least two paths converging to (12).

These paths must be non-intersecting, and one of them (5,7,12) must contain a node strictly dominated by (5).

Dominator Tree

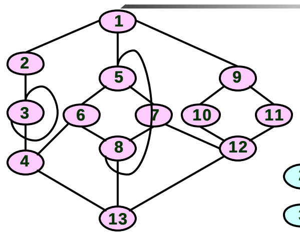
To compute the dominance frontiers, we first compute the dominator tree of the CFG.

There is an edge from node **x** to node **y** in the dominator tree if node **x** immediately dominates node **y**.

I.e., **x** dominates **y**¹**x**, and **x** does not dominate any other dominator of **y**.

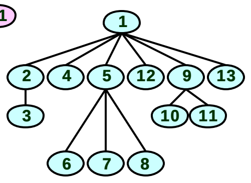
Dominator trees can be computed using the Lengauer-Tarjan algorithm in O(E α(E,N)) time

Example: Dominator Tree



Control Flow Graph

Dominator Tree

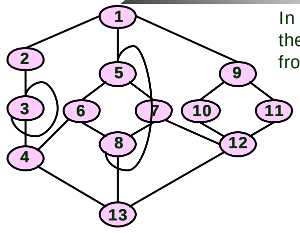


Local Dominance Frontier

Cytron-Ferrante define the local dominance frontier of a node **n** as:

$DF_{local}[\mathbf{n}]$ = successors of **n** in the CFG that are not strictly dominated by **n**

Example: Local Dominance Frontier



Control Flow Graph

In the example, what are the local dominance frontiers of nodes 5, 6 and 7?

$DF_{local}[5]$ =
$DF_{local}[6]$ =
$DF_{local}[7]$ =

$DF_{local}[\mathbf{n}]$ = successors of **n** in the CFG not strictly dominated by **n**

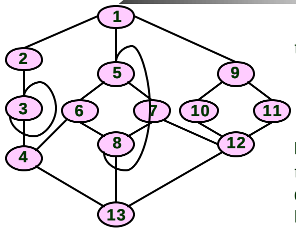
Dominance Frontier Inherited From Its Children

The dominance frontier of a node **n** is formed by its local dominance frontier plus nodes that are passed up by the children of **n** in the dominator tree.

The contribution of a node **c** to its parents' dominance frontier is defined as [Cytron-Ferrante, 1991]:

$DF_{up}[\mathbf{c}]$ = nodes in the dominance frontier of **c** that are not strictly dominated by the immediate dominator of **c**

Example:
Up Dominance Frontier



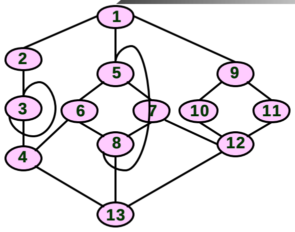
Control Flow Graph

In the example, what are the contributions of nodes 6, 7, and 8 to its parent dominance frontier?

First we compute the DF and immediate dominator of each node:

DF[6] = idom(6)= 5
DF[7] = idom(7)= 5
DF[8] = idom(8)= 5

Example:
Up Dominance Frontier



Control Flow Graph

First, we compute the DF and the immediate dominator of each node:

DF[6] = {4,8}, idom(6)= 5
DF[7] = {8,12}, idom(7)= 5
DF[8] = {5,13}, idom(8)= 5

Now, we check for the DF_{up} condition:

DF_{up}[6] =
DF_{up}[7] =
DF_{up}[8] =

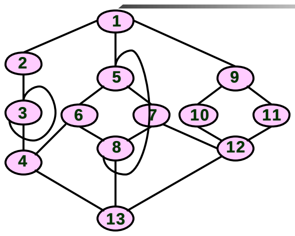
DF_{up}[**c**] = nodes in DF[**c**] not strictly dominated by idom(c)

Dominance Frontier
Inherited From Its Children

The dominance frontier of a node **n** is formed by its local dominance frontier plus nodes that are passed up by the children of **n** in the dominator tree.
Thus the dominance frontier of a node **n** is defined as [Cytron-Ferrante, 1991]:

$$DF[n] = DF_{local}[n] \cup \bigcup_{c \in DTchildren[n]} DF_{up}[c]$$

Example: Local Dominance
Frontier



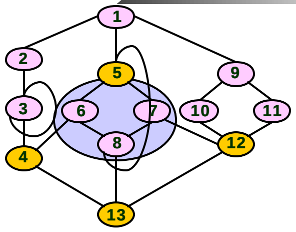
Control Flow Graph

What is DF[5]?

Remember that:

DF_{local}[5] = ∅
DF_{up}[6] = {4}
DF_{up}[7] = {12}
DF_{up}[8] = {5,13}
DTchildren[5] = {6,7,8}

Example: Local Dominance
Frontier



Control Flow Graph

What is DF[5]?

Remember that:

DF_{local}[5] = ∅
DF_{up}[6] = {4}
DF_{up}[7] = {12}
DF_{up}[8] = {5,13}
DTchildren[5] = {6,7,8}

Thus, DF[5] = {4, 5, 12, 13}

Computing Dominance Frontiers

- Use DF_{local} and DF_{up}

```
for X 2 bottom-up traversal of dominance tree
  DF(X) Ã ∅
  for Y 2 Succ(X)
    // local
    if idom(Y) ≠ X then DF(X) Ã DF(X) [ {Y}
  for Z 2 Children(X)
    for Y 2 DF(Z)
      // up
      if idom(Y) ≠ X then DF(X) Ã DF(X) [ {Y}
```

Join Sets

In order to insert ϕ-nodes for a variable **x** that is defined in a set of nodes S={$\mathbf{n_1}$, $\mathbf{n_2}$, …, $\mathbf{n_k}$} we need to compute the iterated set of join nodes of S.

Given a set of nodes S of a control flow graph G, the set of join nodes of S, J(S), is defined as follows:

J(S) ={**z** ∈ G| ∃ two paths $\mathbf{P_{xz}}$ and $\mathbf{P_{yz}}$ in G that
have **z** as its first common node,
x ∈ S and **y** ∈ S}

Iterated Join Sets

Because a ϕ-node is itself a definition of a variable, once we insert ϕ-nodes in the join set of S, we need to find out the join set of S ∪ J(S).
Thus, Cytron-Ferrante define the iterated join set of a set of nodes S, $J^+(S)$, as the limit of the sequence:

$$J_1 = J(S)$$
$$J_{i+1} = J(S \cup J_i)$$

Iterated Dominance Frontier

We can extend the concept of dominance frontier to define the dominance frontier of a set of nodes as:

$$DF(S) = \underset{X \in S}{\mathrm{Y}} DF(X)$$

Now we can define the iterated dominance frontier, $DF^+(S)$, of a set of nodes S as the limit of the sequence:

$$DF_1 = DF(S)$$
$$DF_{i+1} = DF(S \cup DF_i)$$

Location of f-Nodes

Given a variable **x** that is defined in a set of nodes S={$\mathbf{n_1}$, $\mathbf{n_2}$, …, $\mathbf{n_k}$} the set of nodes that must receive ϕ-nodes for **x** is $J^+(S)$.

An important result proved by Cytron-Ferrante is that:

$$J^+(S) = DF^+(S)$$

Thus we are really interested in computing the iterated dominance frontier of a set of nodes.

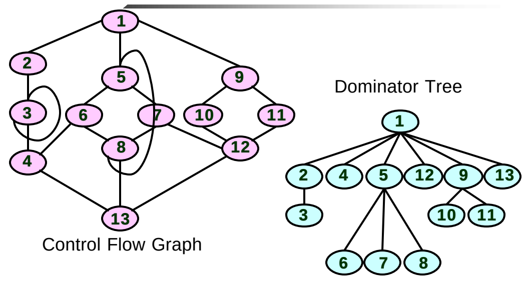
Algorithms to Compute F Node Placement

The algorithm to insert ϕ-nodes, due to Cytron and Ferrante (1991), computes the dominance frontier of each node in the set S before computing the iterated dominance frontier of the set.

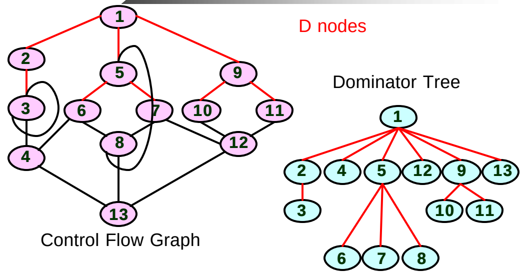
In the worst case, the combination of the dominance frontier of the sets can be quadratic in the number of nodes in the CFG. Thus, Cytron-Ferrante's algorithm has a complexity $O(N^2)$.

In 1994, Shreedar and Gao proposed a simple, linear algorithm for the insertion of ϕ-nodes.

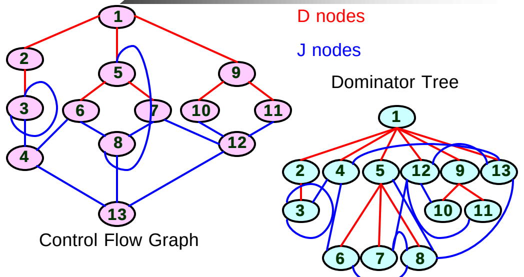
Sreedhar and Gao's DJ Graph



Sreedhar and Gao's DJ Graph



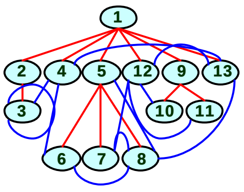
Sreedhar and Gao's DJ Graph



Shreedar-Gao's Dominance Frontier Algorithm

```
DominanceFrontier(x)
0:  DF[x] = ∅
1:  foreach y ∈ SubTree(x) do
2:      if((y → z == J-edge) and
3:              (z.level ≤ x.level))
4:          then DF[x] = DF[x] ∪ z
```

What is the DF[5]?

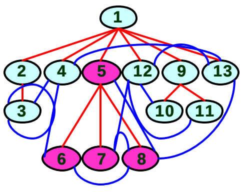


Shreedar-Gao's Dominance Frontier Algorithm

Initialization: DF[5] = ∅

```
DominanceFrontier(x)
0:  DF[x] = ∅
1:  foreach y ∈ SubTree(x) do
2:      if((y → z == J-edge) and
3:              (z.level ≤ x.level))
4:          then DF[x] = DF[x] ∪ z
```

SubTree(5) = {5, 6, 7, 8}



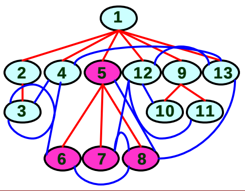
Shreedar-Gao's Dominance Frontier Algorithm

Initialization: DF[5] = ∅

```
DominanceFrontier(x)
0:  DF[x] = ∅
1:  foreach y ∈ SubTree(x) do
2:      if((y → z == J-edge) and
3:              (z.level ≤ x.level))
4:          then DF[x] = DF[x] ∪ z
```

SubTree(5) = {5, 6, 7, 8}

There are three edges originating in 5: {5→6, 5→7, 5→8} but they are all D-edges



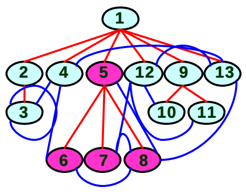
Shreedar-Gao's Dominance Frontier Algorithm

Initialization: DF[5] = ∅
After visiting 6: DF = {4}

```
DominanceFrontier(x)
0:  DF[x] = ∅
1:  foreach y ∈ SubTree(x) do
2:      if((y → z == J-edge) and
3:              (z.level ≤ x.level))
4:          then DF[x] = DF[x] ∪ z
```

SubTree(5) = {5, 6, 7, 8}

There are two edges originating in 6: {6→4, 6→8} but 8.level > 5.level



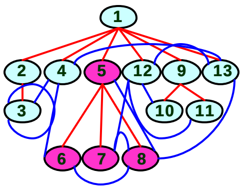
Shreedar-Gao's Dominance Frontier Algorithm

```
DominanceFrontier(x)
0:   DF[x] = ∅
1:   foreach  y ∈ SubTree(x) do
2:       if((y → z == J-edge) and
3:                   (z.level ≤ x.level))
4:         then DF[x] = DF[x] ∪ z
```

Initialization: DF[5] = ∅
After visiting 6: DF = {4}
After visiting 7: DF = {4,12}

SubTree(5) = {5, 6, 7, 8}

There are two edges originating in 7:
{7→8, 7→12}
again 8.level > 5.level



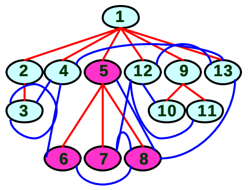
Shreedar-Gao's Dominance Frontier Algorithm

```
DominanceFrontier(x)
0:   DF[x] = ∅
1:   foreach  y ∈ SubTree(x) do
2:       if((y → z == J-edge) and
3:                   (z.level ≤ x.level))
4:         then DF[x] = DF[x] ∪ z
```

Initialization: DF[5] = ∅
After visiting 6: DF = {4}
After visiting 7: DF = {4,12}
After visiting 8: DF = {4, 12, 5, 13}

SubTree(5) = {5, 6, 7, 8}

There are two edges originating in 8:
{8→5, 8→13}
both satisfy cond. in steps 2-3



Shreedhar-Gao's f-Node Insertion Algorithm

Using the D-J graph, Shreedhar and Gao propose a linear time algorithm to compute the iterated dominance frontier of a set of nodes.

A important intuition in Shreedhar-Gao's algorithm is:

If two nodes **x** and **y** are in S, and **y** is an ancestor of **x** in the dominator tree, than if we compute DF[**x**] first, we do not need to recompute it when computing DF[**y**].

Shreedhar-Gao's f-Node Insertion Algorithm

Shreedhar-Gao's algorithm also use a work list of nodes hashed by their level in the dominator tree and a visited flag to avoid visiting the same node more than once.

The basic operation of the algorithm is similar to their dominance frontier algorithm, but it requires a careful implementation to deliver the linear time complexity.

Dead-Code Elimination in SSA Form

Only one definition for each variable
) if the list of uses of the variable is empty, definition is dead

When a statement **v** ¬ **x** Å **y** is eliminated because **v** is dead, this statement must be removed from the list of uses of **x** and **y**, which might cause those definitions to become dead.

Thus we need to iterate the dead code elimination algorithm.

Simple Constant Propagation in SSA

If there is a statement **v** ¬ **c**, where **c** is a constant, then all uses of **v** can be replaced for **c**.

A φ function of the form **v** ¬ f(**c**$_1$, **c**$_2$, ..., **c**$_n$) where all **c**$_i$ are identical can be replaced for **v** ¬ **c**.

Using a work list algorithm in a program in SSA form, we can perform constant propagation in linear time

In the next slide we assume that **x**, **y**, **z** are variables and **a**, **b**, **c** are constants.

Linear Time Optimizations in SSA form

Copy propagation: The statement **x ¬ f (y)** or the statement **x ¬ y** can be deleted and **y** can substitute every use of **x**.

Constant folding: If we have the statement **x ¬ a** Å **b**, we can evaluate **c ¬ a** Å **b** at compile time and replace the statement for **x ¬ c**

Constant conditions: The conditional if **a < b** goto L1 else L2 can be replaced for goto L1 or goto L2, according to the compile time evaluation of **a < b**, and the CFG, use lists, adjust accordingly

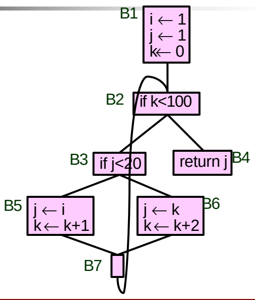
Unreachable Code: eliminate unreachable blocks.

Next Time

- Implementing other optimizations with SSA

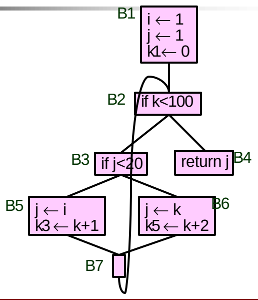
Single Assignment Form

```
i=1;
j=1;
k=0;
 while(k<100)
     {
       if(j<20)
         {
            j=i;
            k=k+1;
         }
       else
         {
            j=k;
            k=k+2;
         }
     }
 return j;
}
```



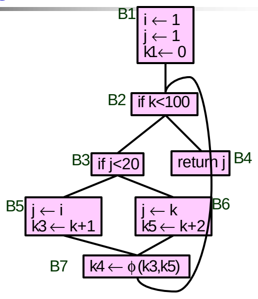
Single Assignment Form

```
i=1;
j=1;
k=0;
 while(k<100)
     {
       if(j<20)
         {
            j=i;
            k=k+1;
         }
       else
         {
            j=k;
            k=k+2;
         }
     }
 return j;
}
```



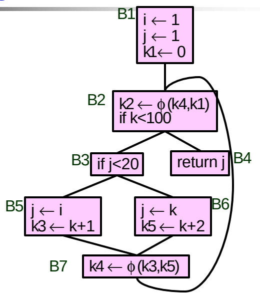
Single Assignment Form

```
i=1;
j=1;
k=0;
 while(k<100)
     {
       if(j<20)
         {
            j=i;
            k=k+1;
         }
       else
         {
            j=k;
            k=k+2;
         }
     }
 return j;
}
```



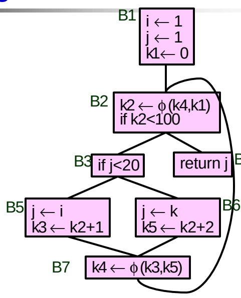
Single Assignment Form

```
i=1;
j=1;
k=0;
 while(k<100)
     {
       if(j<20)
         {
            j=i;
            k=k+1;
         }
       else
         {
            j=k;
            k=k+2;
         }
     }
 return j;
}
```



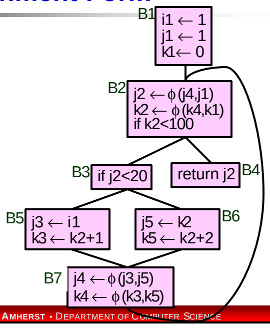
Single Assignment Form

```
i=1;
j=1;
k=0;
 while(k<100)
      {
       if(j<20)
          {
            j=i;
            k=k+1;
          }
       else
          {
            j=k;
            k=k+2;
          }
      }
  return j;
}
```

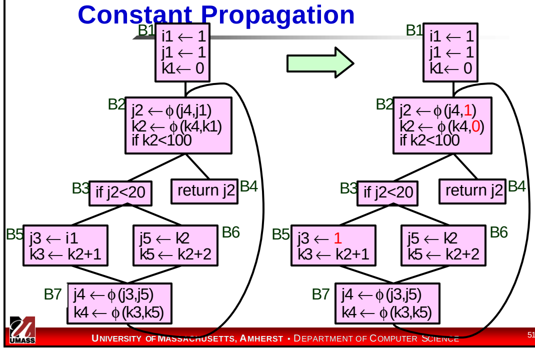


Single Assignment Form

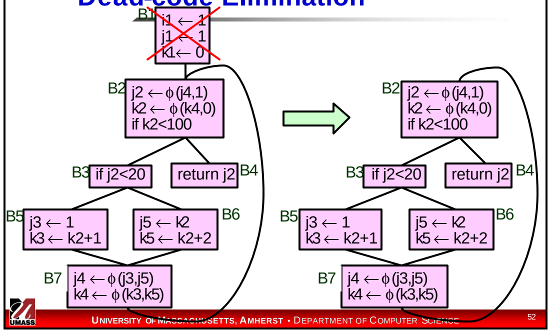
```
i=1;
j=1;
k=0;
 while(k<100)
      {
       if(j<20)
          {
            j=i;
            k=k+1;
          }
       else
          {
            j=k;
            k=k+2;
          }
      }
  return j;
}
```



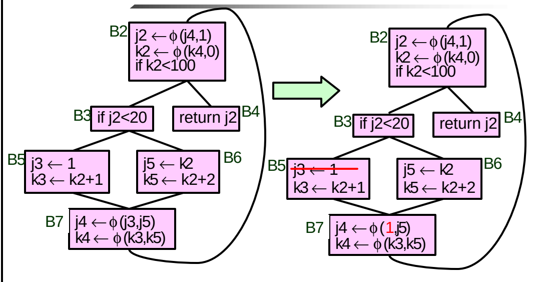
Example:
Constant Propagation



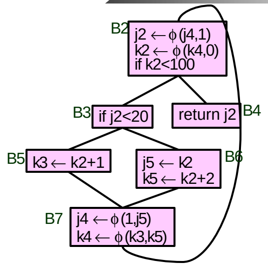
Example:
Dead-code Elimination



Constant Propagation and Dead Code Elimination



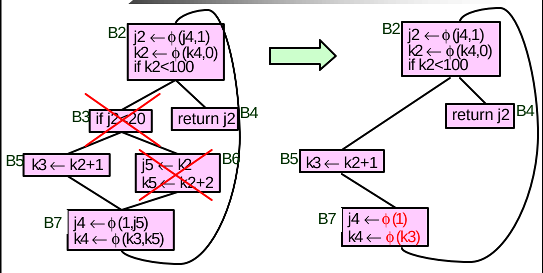
Example:
Is this the end?



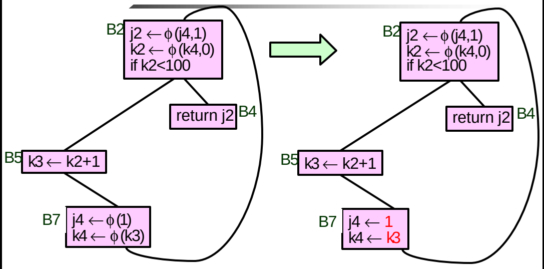
But block 6 is never executed! How can we find this out, and simplify the program?

SSA conditional constant propagation finds the *least fixed point* for the program and allows further elimination of dead code.

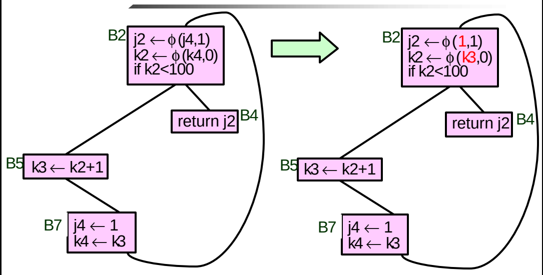
Example: Dead code elimination



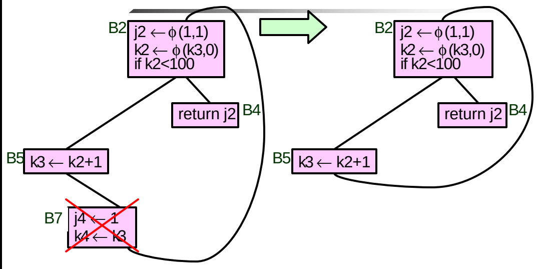
Example: Single Argument f-Function Elimination



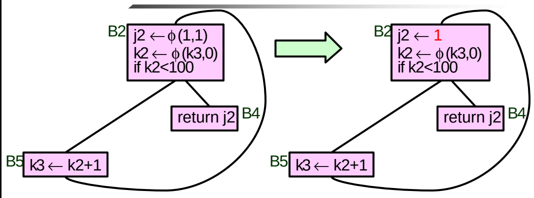
Example: Constant and Copy Propagation



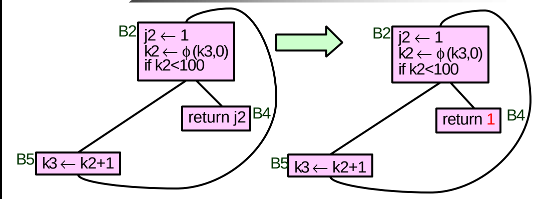
Example: Dead Code Elimination



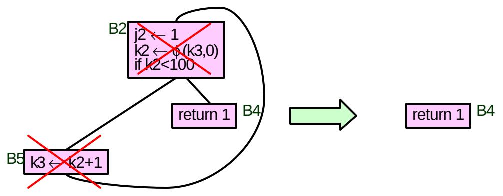
Example: f-Function Simplification



Example: Constant Propagation



Example: Dead Code Elimination



Next Time