Translating Control Flow Graphs

Dominator Trees

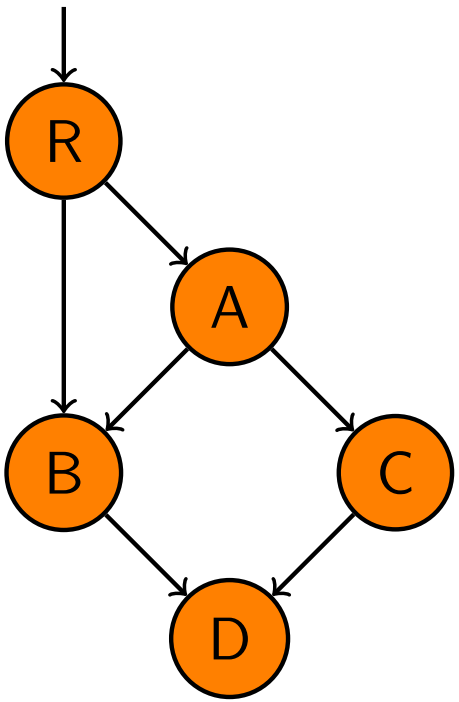
Christoph Hoffmeyer

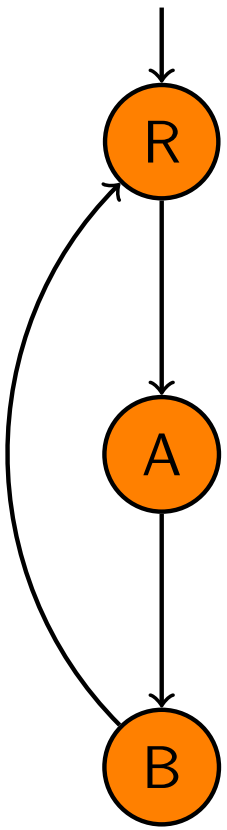
Radboud University

23th February 2023

Overview

Control Flow Graphs



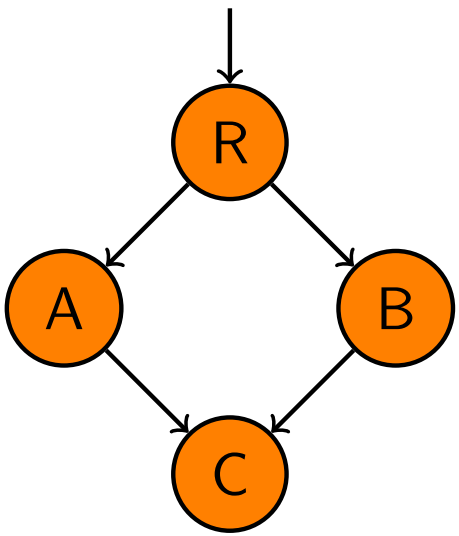


Web Assembly

- WebAssembly is a low-level intermediate code for the web like JavaScript
- Instructions are **if**, **loop**, **block** and **br** (branch)
- We don't have a **goto** and have to work around this

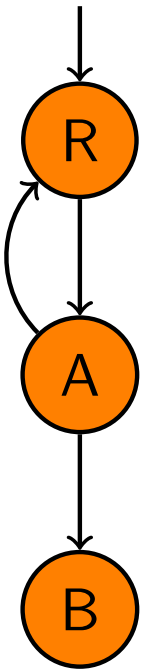
Examples

WebAssembly: Simple conditional



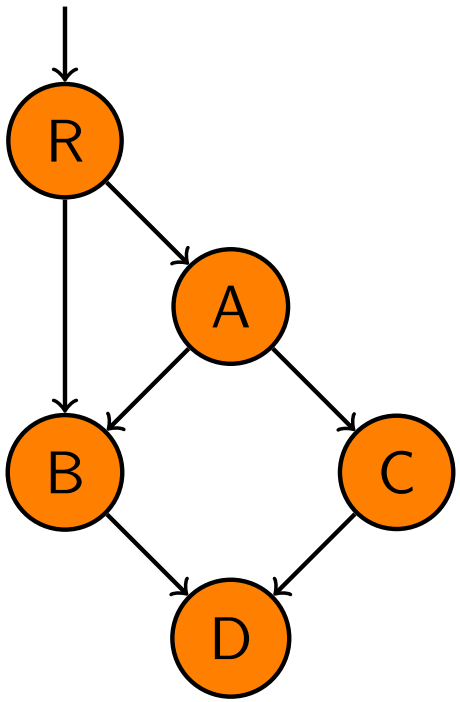

```
do R
if R
    do A
else
    do B
end
do C
```

WebAssembly: Simple loop



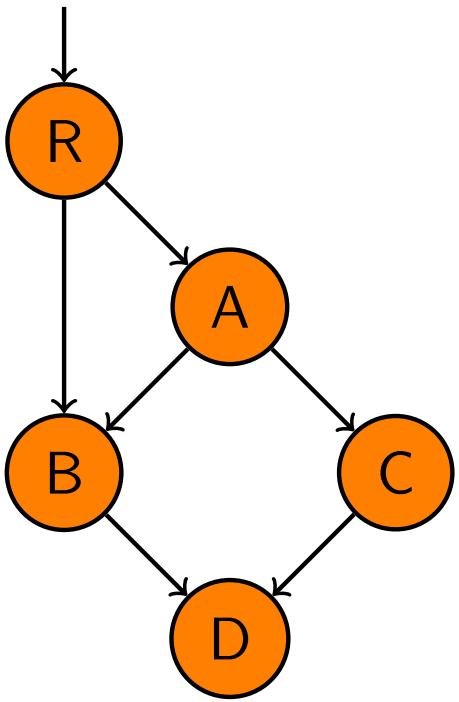

```
loop
    do R
    do A
    if A
        br R
    else
        do B
        exit
    end
end
```

Control Flow Graph without loops



How do we translate this into WebAssembly?

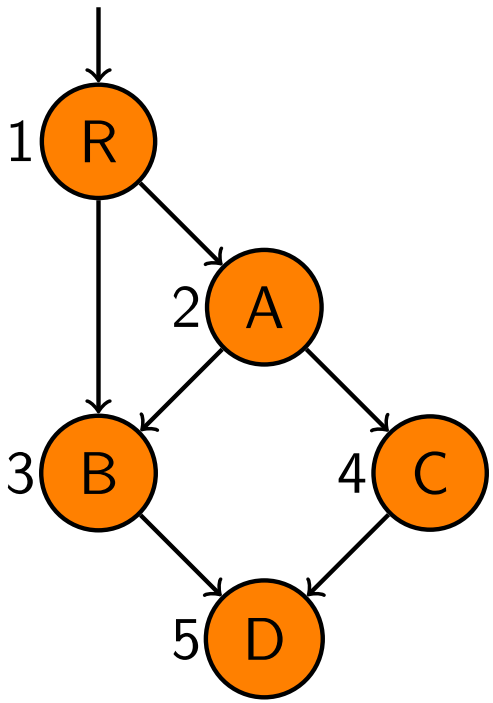
- Do we have to copy B?




```
do R
if R
    do A
    if A
        do B?
    else
        do C
else
    do B?
...
```

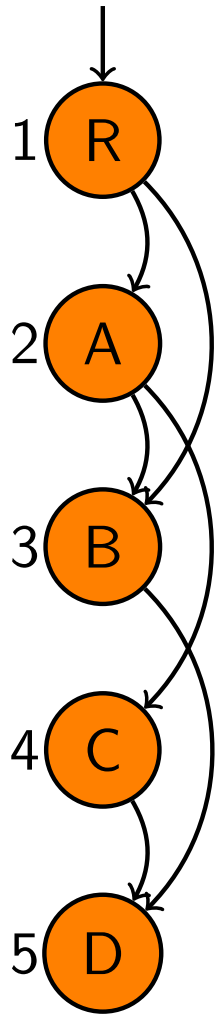
Reverse Post Order Numbering

1. Recursively traverse the current vertex's right subtree
2. Recursively traverse the current vertex's left subtree
3. Visit the current vertex.



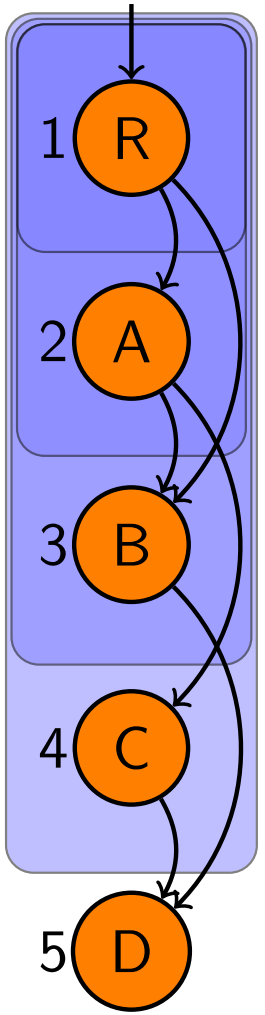
Control Flow Graph without loops

- We can rearrange the vertices
- Create blocks around each vertex
- We can always jump to the end of each block
- We can always apply this ordering if we have no loops

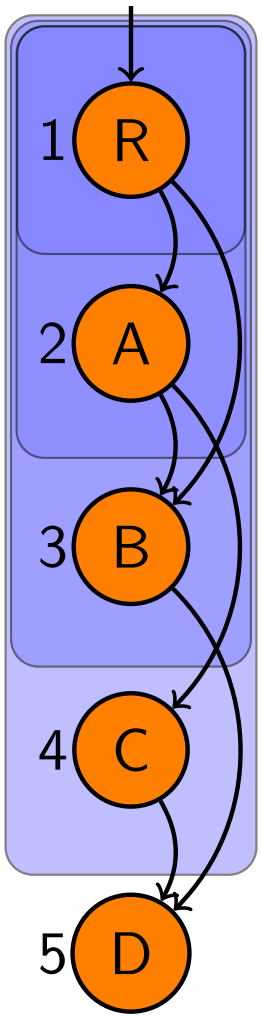


Control Flow Graph without loops

- We can rearrange the vertices
- Create blocks around each vertex
- We can always jump to the end of each block
- We can always apply this ordering if we have no loops



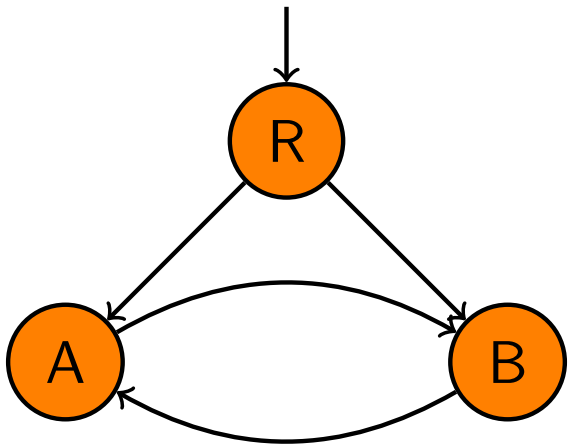
Control Flow Graph without loops



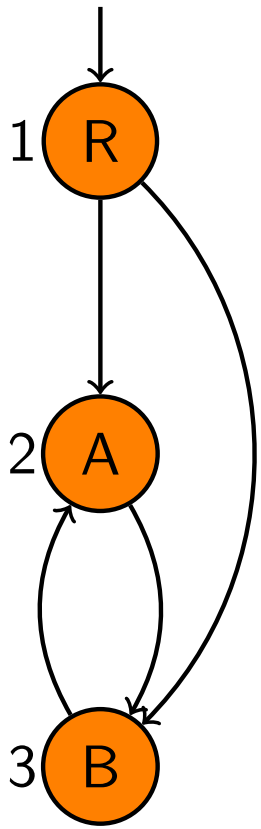

```
block D
block C
block B
block A
    do R
    if R
        br A
    else
        br B
    end
end
do A
...
```

Irreducible Flow Graphs

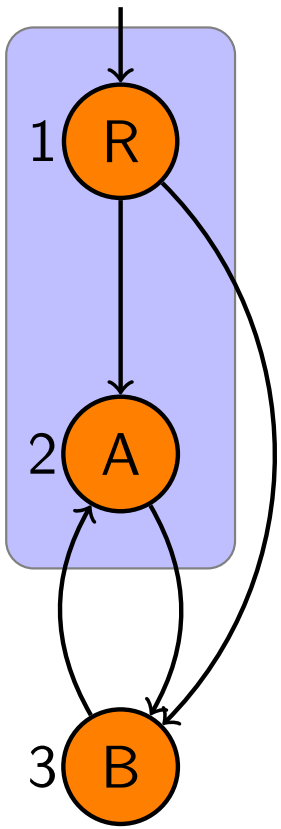
- We cannot place blocks correctly for every graph

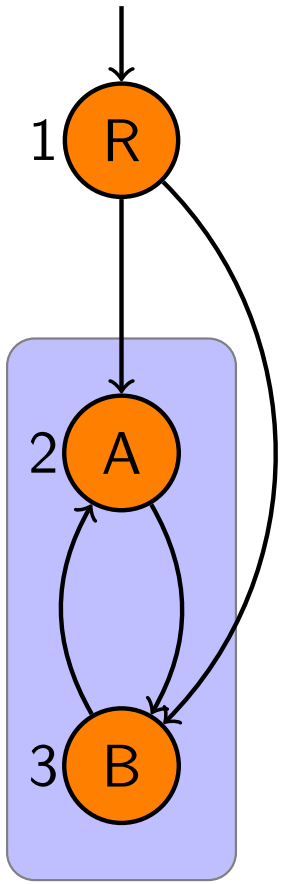


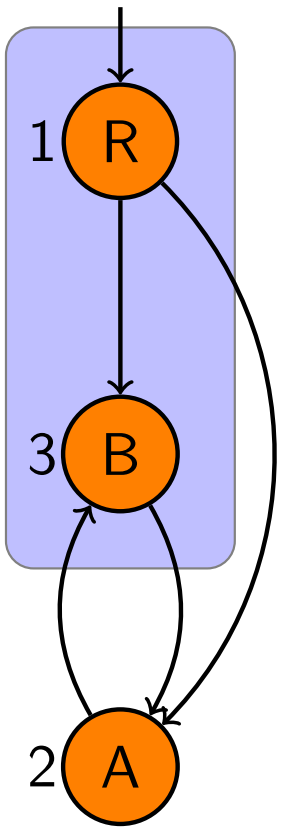
Irreducible Flow Graphs



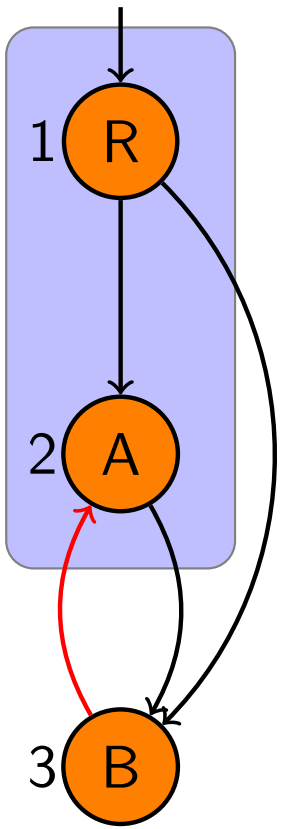
Irreducible Flow Graphs

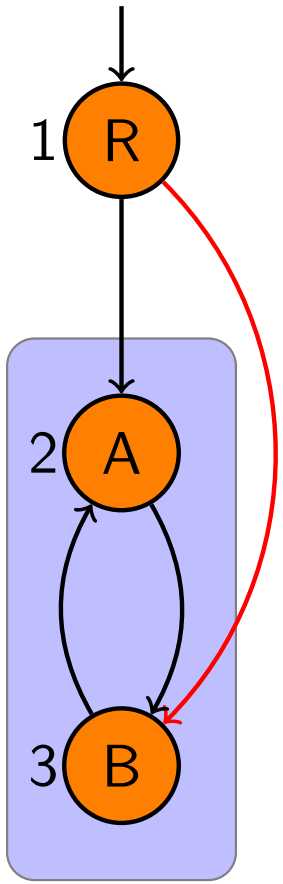


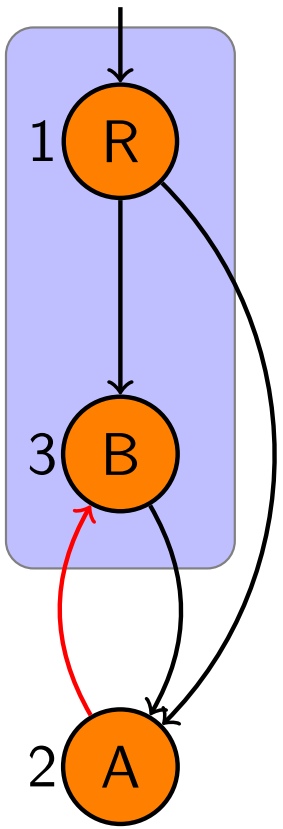




Irreducible Flow Graphs

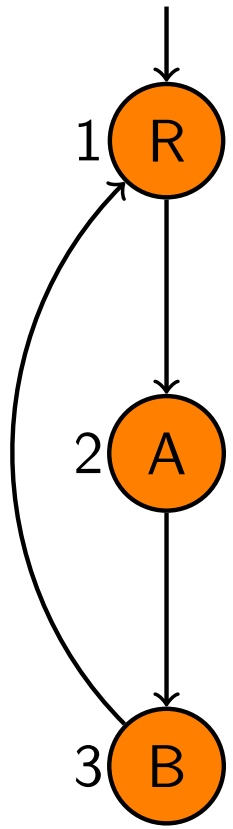






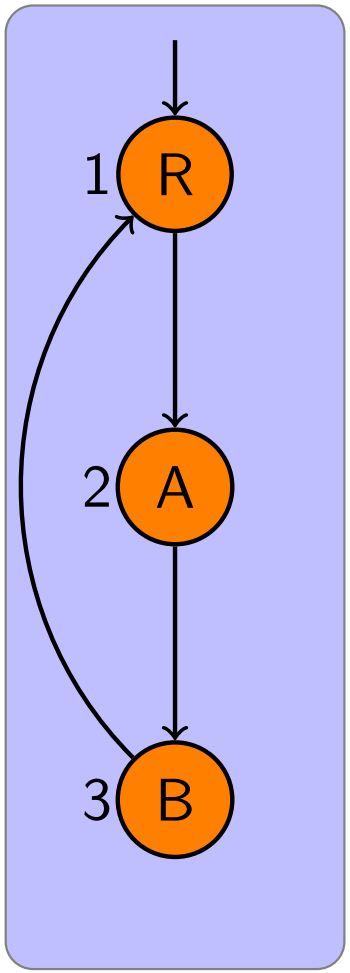
Reducibility

- We can go to the beginning of a block
- Every loop needs a unique entry point
- All vertices in a loop can only be reached by traversing the entry point
- The entry point dominates all paths to the vertices within the loop



Reducibility

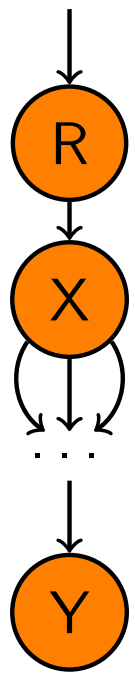
- We can go to the beginning of a block
- Every loop needs a unique entry point
- All vertices in a loop can only be reach by traversing the entry point
- The entry point dominates all paths to the vertices within the loop



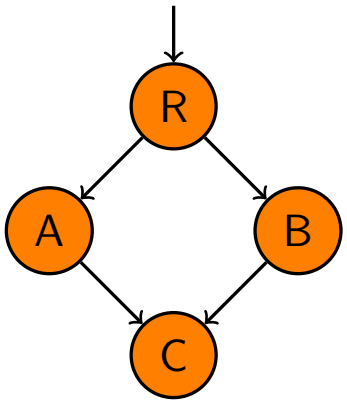
Dominator Tree

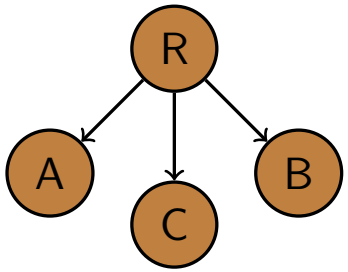
Dominator Tree

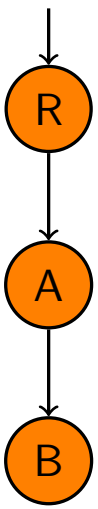
- Expresses dominance relation between vertices of a control-flow graph
- Vertex X dominates vertex Y if X appears on every path from the root R to Y

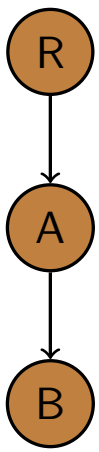


Example Trees



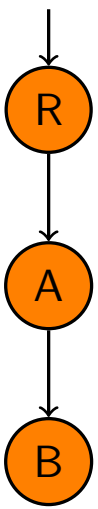


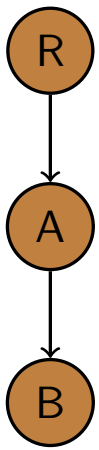




Dominator Tree

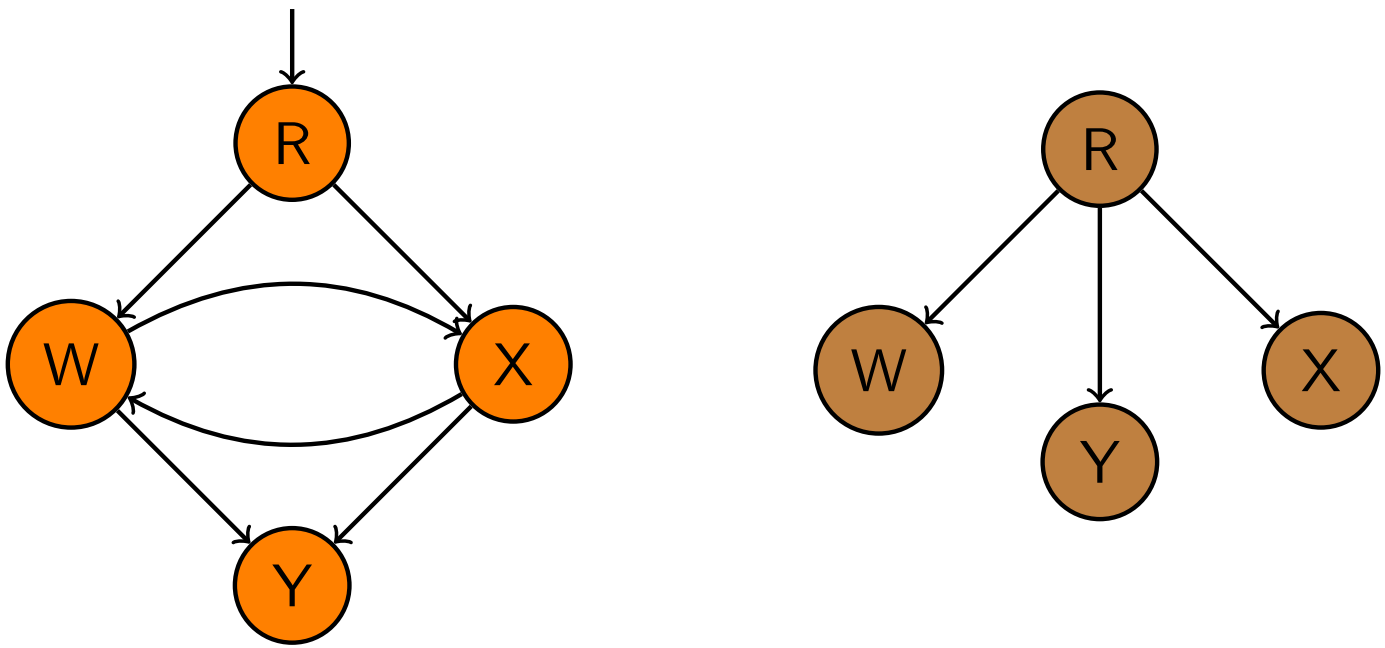
- Every vertex Y dominates itself
- A vertex may dominate more than one other vertex, and it may be dominated by more than one other vertex





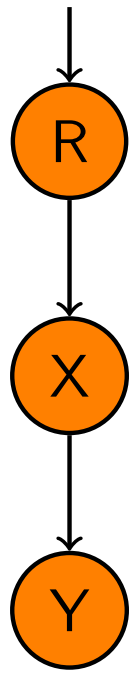
Dominator Tree

- If both W and X dominate Y, then either W dominates X or X dominates W
- Counterexample:



Immediate Dominator

- Vertex X is immediate dominator of Y if X dominates Y and every other dominator of Y dominates X
- All dominator of Y must be on all paths towards Y and they can't dominate each other
- That means there is always an immediate dominator

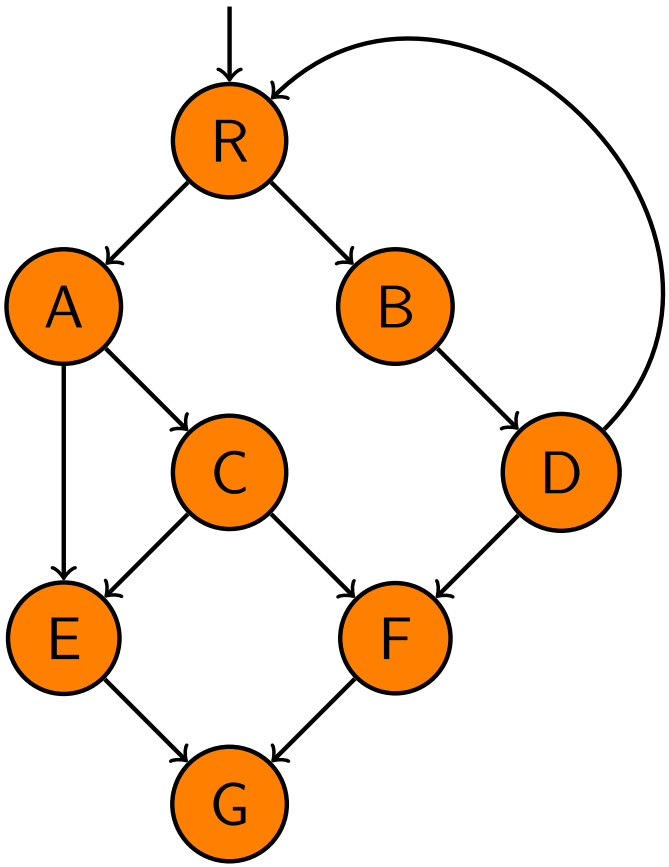


Algorithm

- Compute set $S(v)$ of vertices that can be reached without traversing v
- Compute set of vertices dominated by v: $dom(v) = V - \{v\} - S(v)$
- Construct dominator tree
- Runtime $\mathcal{O}(nm)$ with $|V| = n$ and $|E| = m$

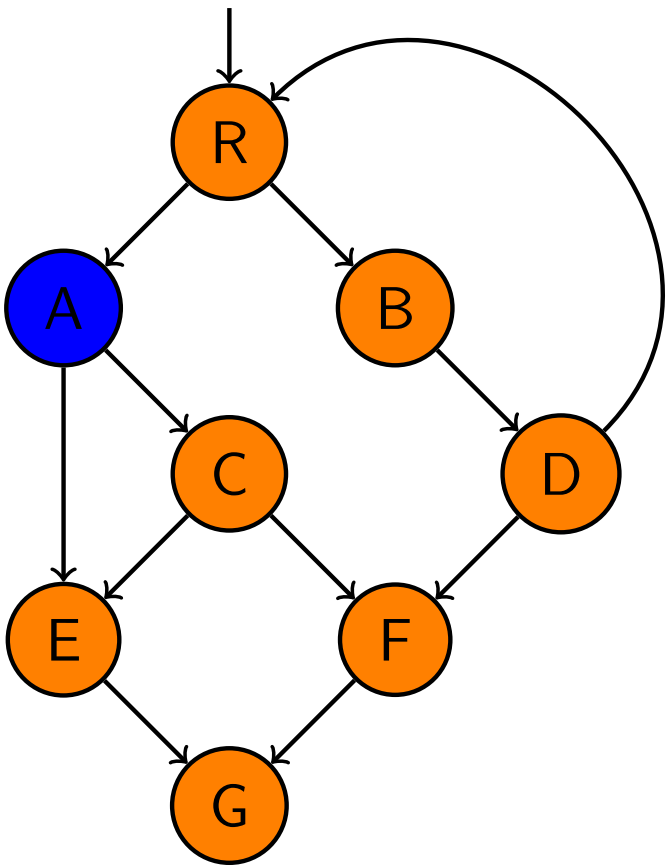
Example

- Compute set $S(v)$ of vertices that can be reached without traversing $v \neq r$



Example

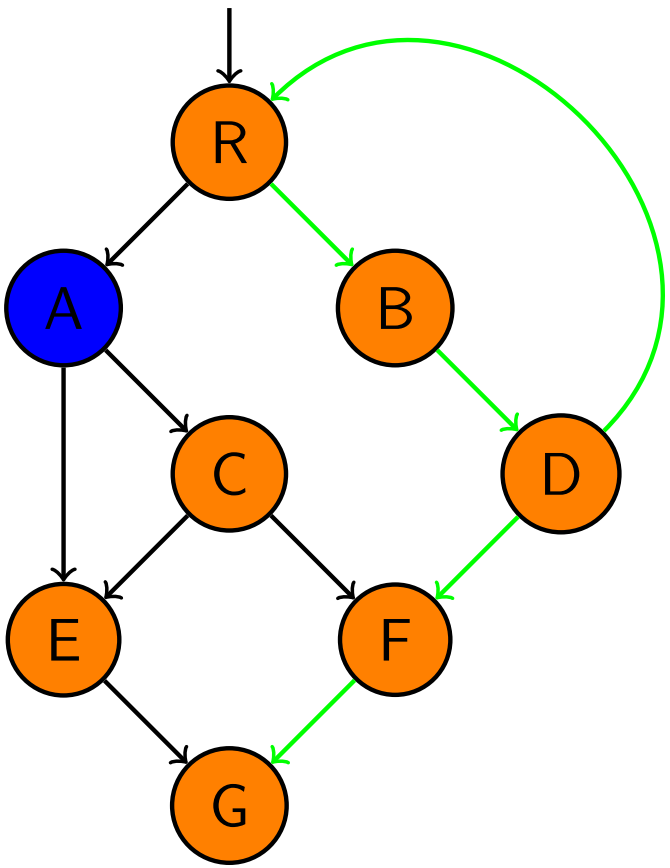
- Compute set $S(v)$ of vertices that can be reached without traversing $v \neq r$



- $S(A)$

Example

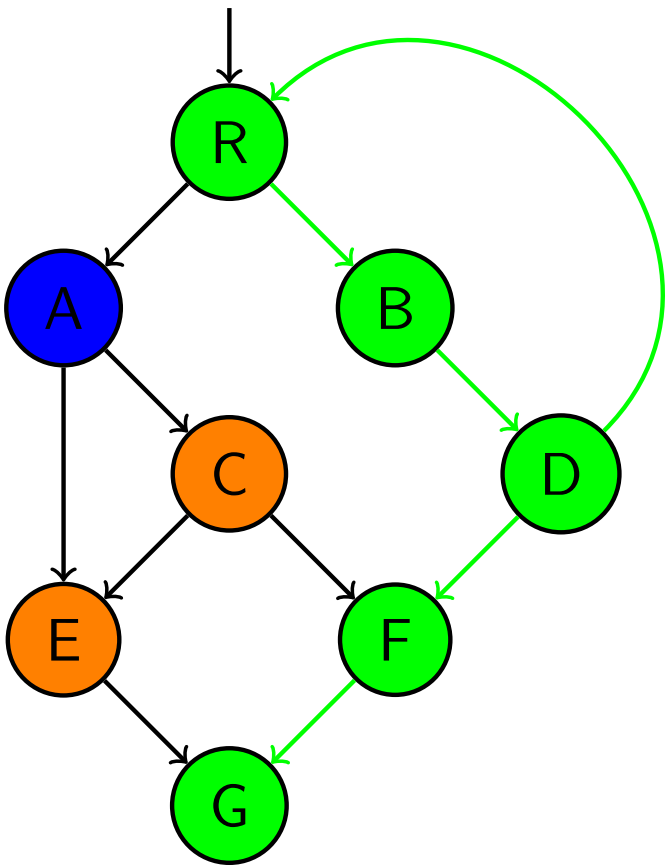
- Compute set $S(v)$ of vertices that can be reached without traversing $v \neq r$



- $S(A)$

Example

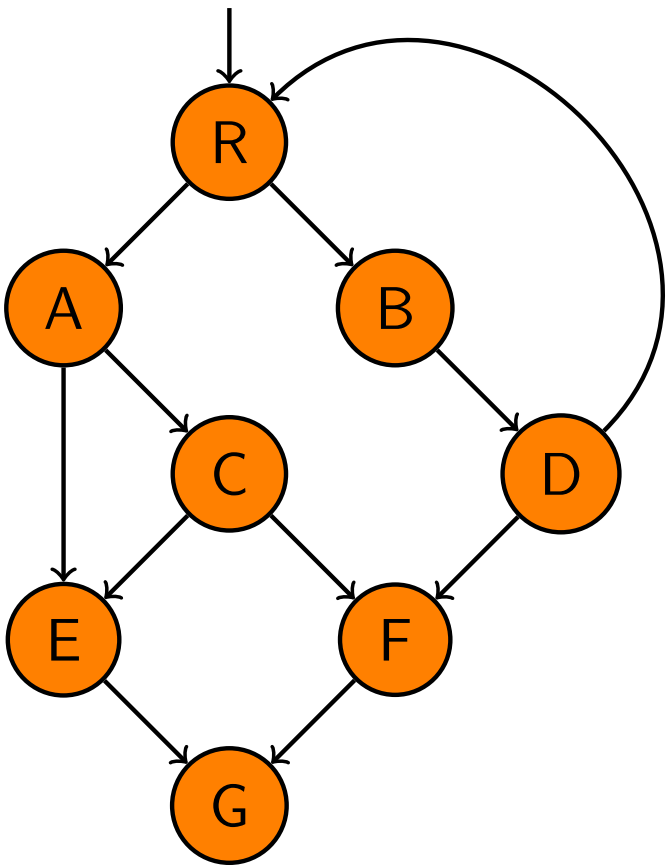
- Compute set $S(v)$ of vertices that can be reached without traversing $v \neq r$



- $S(A) = \{R, B, D, F, G\}$

Example

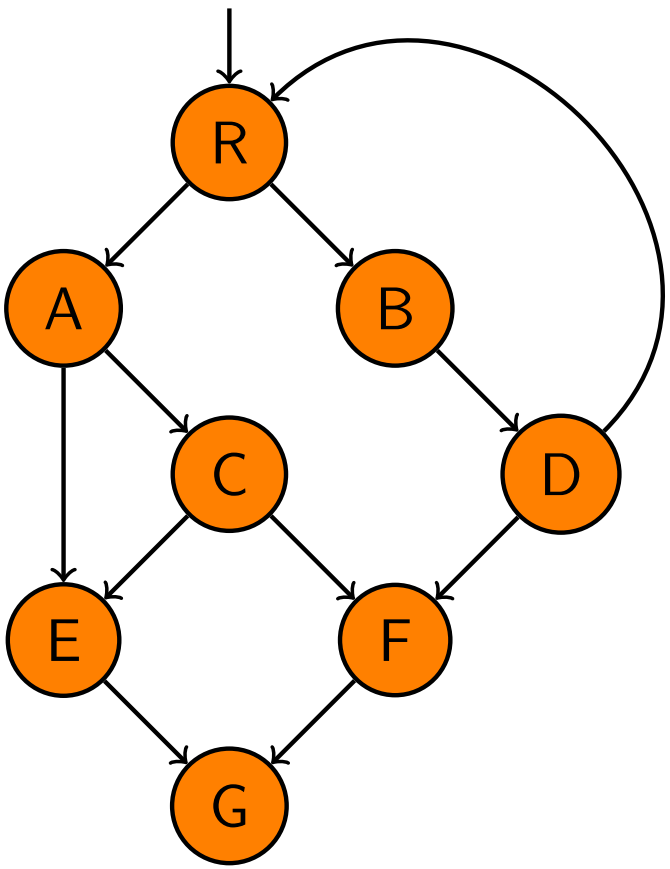
- Compute set $S(v)$ of vertices that can be reached without traversing $v \neq r$



- $S(A) = \{R, B, D, F, G\}$
- $S(B) = \{R, A, C, E, F, G\}$
- $S(C) = V \setminus \{C\}$
- $S(D) = V \setminus \{D\}$
- $S(E) = V \setminus \{E\}$
- $S(F) = V \setminus \{F\}$
- $S(G) = V \setminus \{G\}$

Example

- Compute set of vertices dominated by v:
  $dom(v) = V - \{v\} - S(v)$



- $S(A) = \{R, B, D, F, G\}$
- $S(B) = \{R, A, C, E, F, G\}$
- $S(C) = V \setminus \{C\}$
- $S(D) = V \setminus \{D\}$
- $S(E) = V \setminus \{E\}$
- $S(F) = V \setminus \{F\}$
- $S(G) = V \setminus \{G\}$

Example

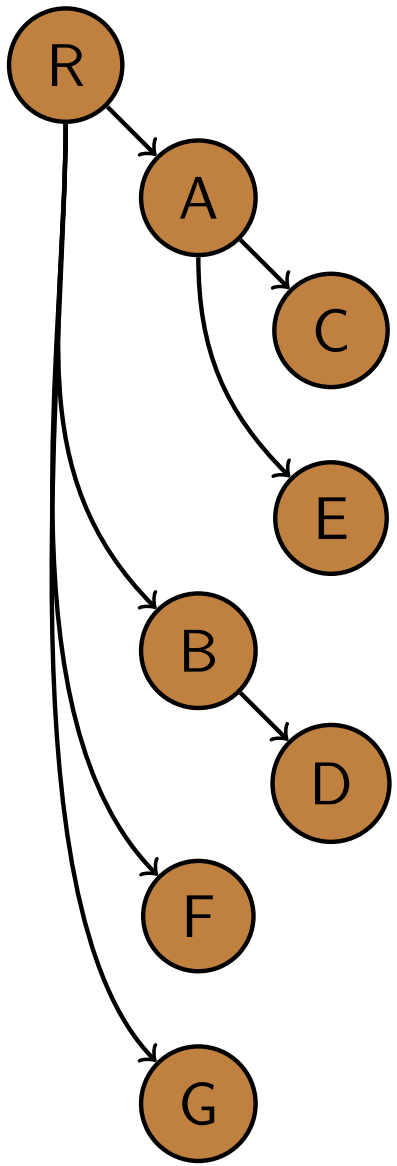
- Compute set of vertices dominated by v: $dom(v) = V - \{v\} - S(v)$

- $S(A) = \{R, B, D, F, G\}$
- $S(B) = \{R, A, C, E, F, G\}$
- $S(C) = V \setminus \{C\}$
- $S(D) = V \setminus \{D\}$
- $S(E) = V \setminus \{E\}$
- $S(F) = V \setminus \{F\}$
- $S(G) = V \setminus \{G\}$

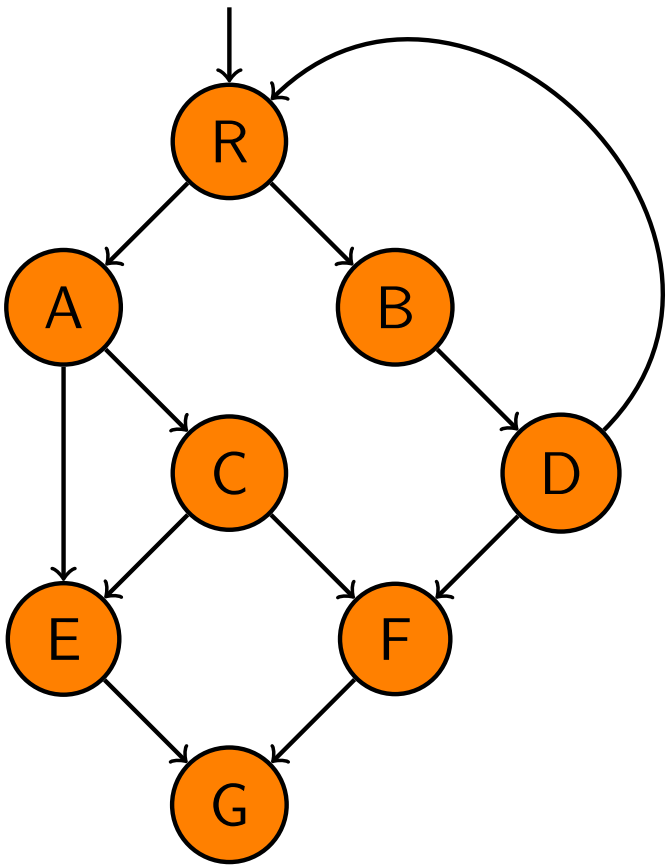
- $dom(A) = \{C, E\}$
- $dom(B) = \{D\}$
- $dom(C) = \emptyset$
- $dom(D) = \emptyset$
- $dom(E) = \emptyset$
- $dom(F) = \emptyset$
- $dom(G) = \emptyset$

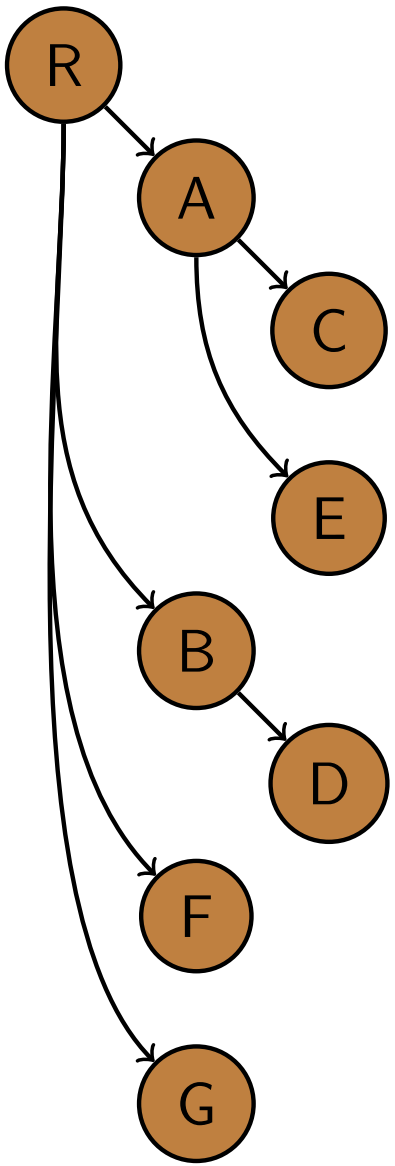
Construct dominator tree

- $dom(R) = V$
- $dom(A) = \{C, E\}$
- $dom(B) = \{D\}$
- $dom(C) = \emptyset$
- $dom(D) = \emptyset$
- $dom(E) = \emptyset$
- $dom(F) = \emptyset$
- $dom(G) = \emptyset$



Construct dominator tree





State of the art

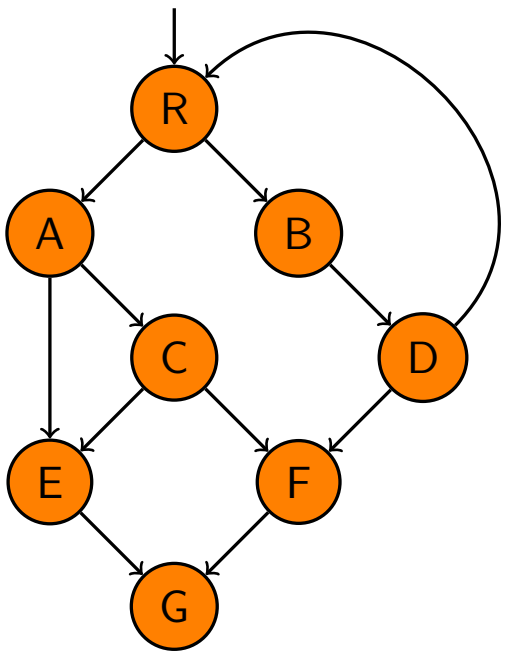
- An algorithm by T. Lengauer and R. E. Tarjan
- It computes an intermediate step called semidominator to compute immediate dominators
- Runtime is $\mathcal{O}(m \log n)$, where m is the number of edges and n is the number of vertices

Translating Control Flow

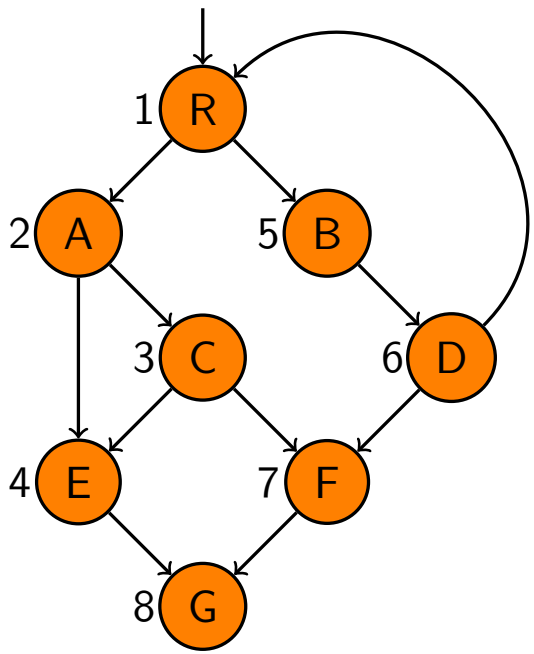
Algorithm

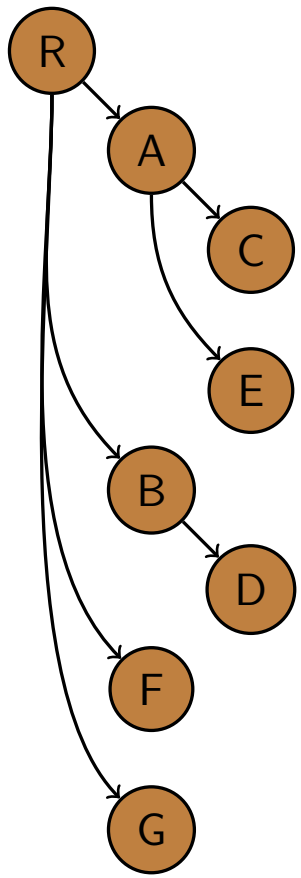
Compute reverse post order numbering and dominator tree
Start with the root and traverse the dominator tree

1. Check for loop
   - Create loop structure
2. Check for merge vertices
   - Create block structure
   - Repeat for merge vertices
3. Translate vertex
4. Check out-going edges
   - Create if-then-else structure
5. Place children, branch or exit. Repeat for direct children

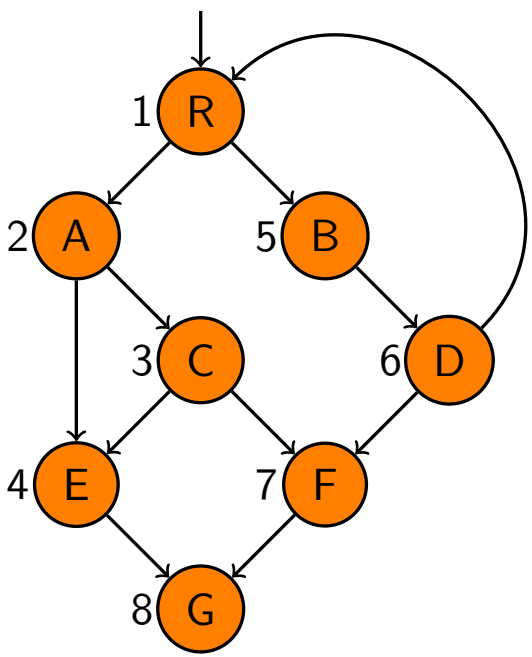


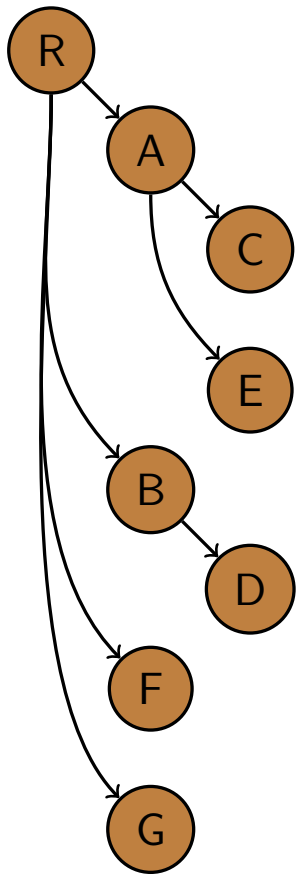
Algorithm

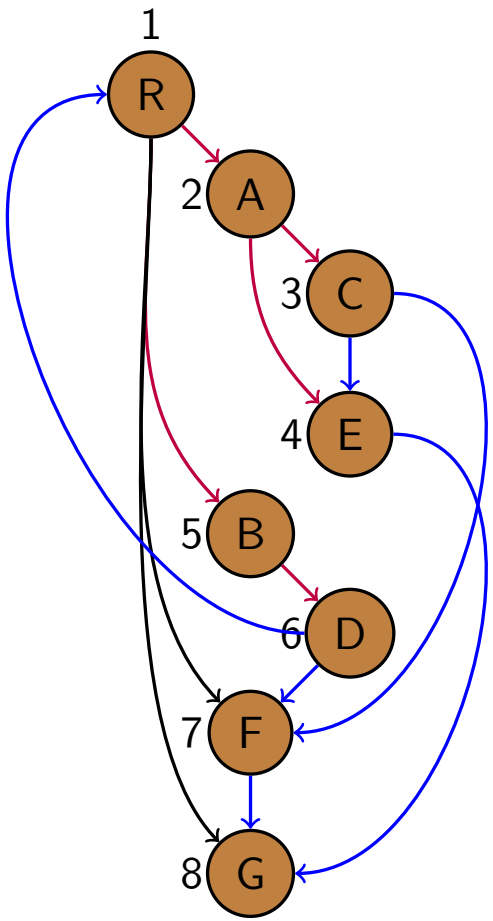




Algorithm

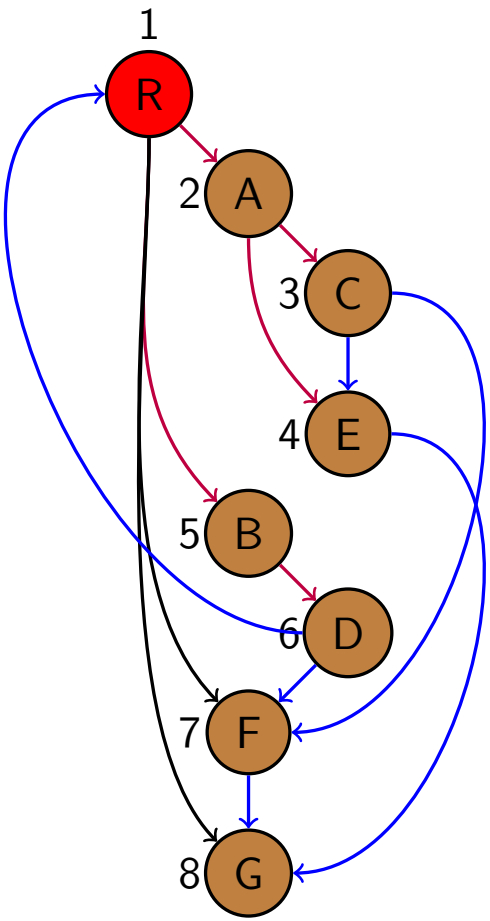






Algorithm

- Check for loop

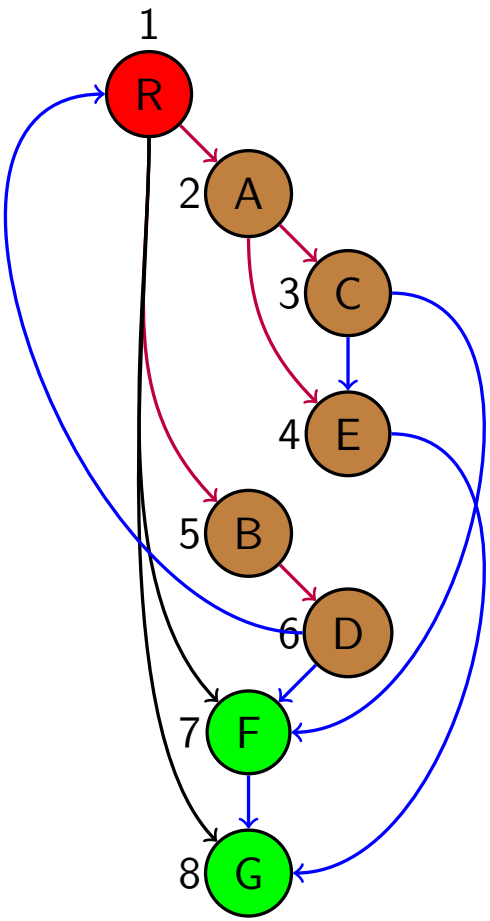


loop

end

Algorithm

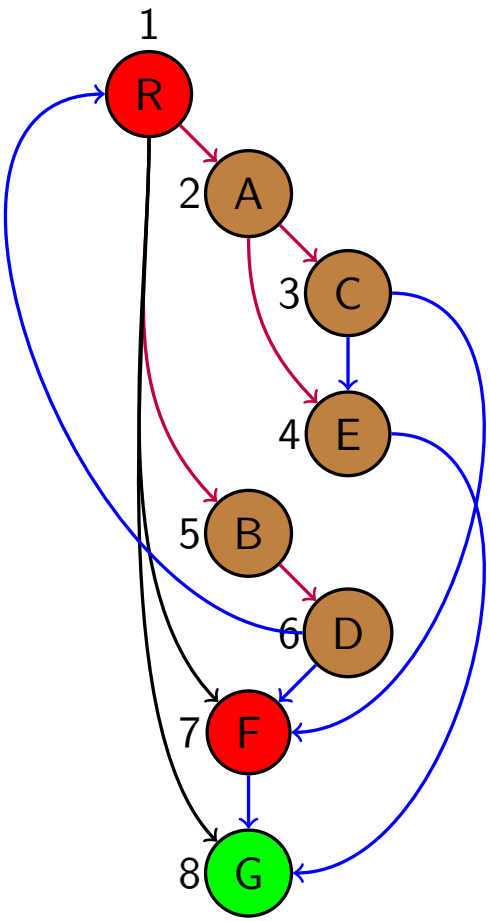
- Check for merge vertices




```
loop
 block G
  block F
   Translation of R
  end
  Translation of F
 end
 Translation of G
end
```

Algorithm

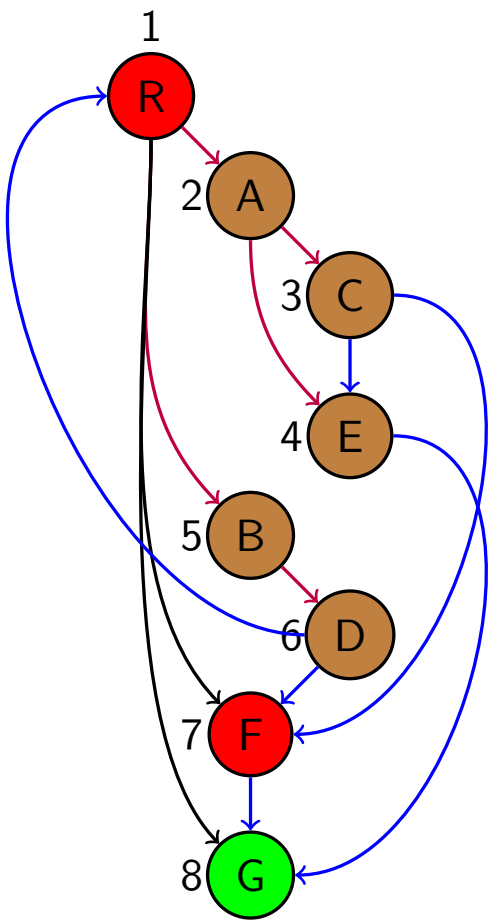
- Recursively translate F




```
loop
 block G
  block F
   Translation of R
  end
  Translation of F
 end
 Translation of G
end
```

Algorithm

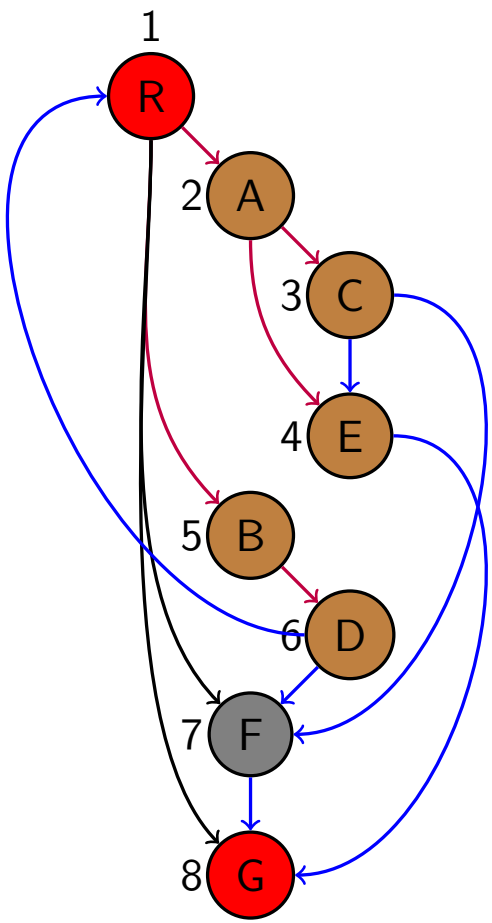
- Translate F and place br




```
loop
 block G
  block F
   Translation of R
  end
  do F
  br G
 end
 Translation of G
end
```

Algorithm

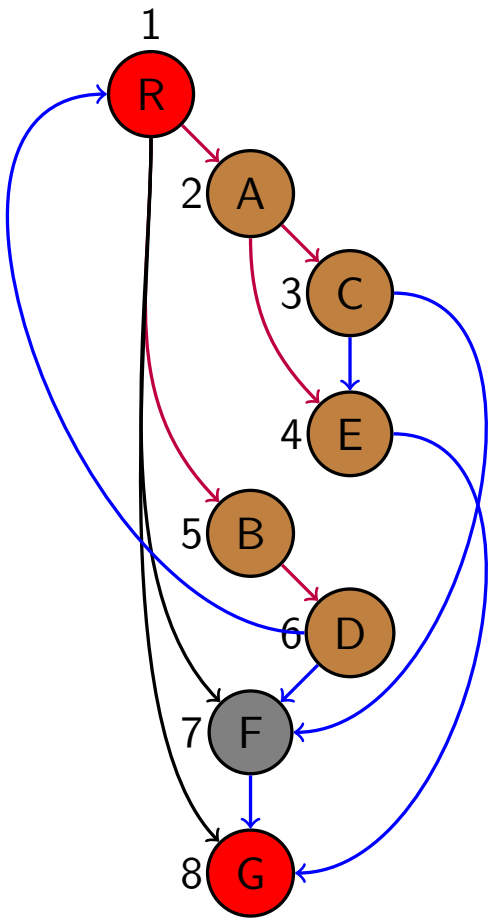
- Recursively translate G




```
loop
 block G
  block F
   Translation of R
  end
  do F
  br G
 end
 Translation of G
end
```

Algorithm

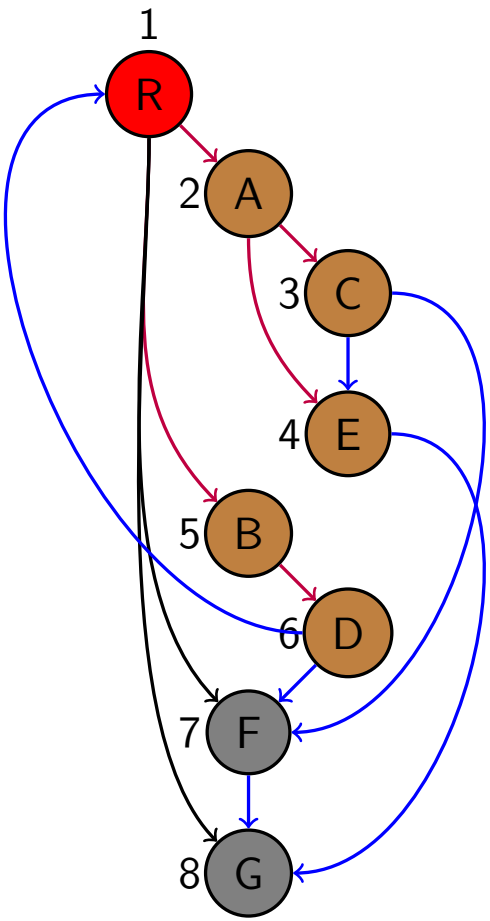
- Translate G and place exit




```
loop
 block G
  block F
   Translation of R
  end
  do F
  br G
 end
 do G
 exit
end
```

Algorithm

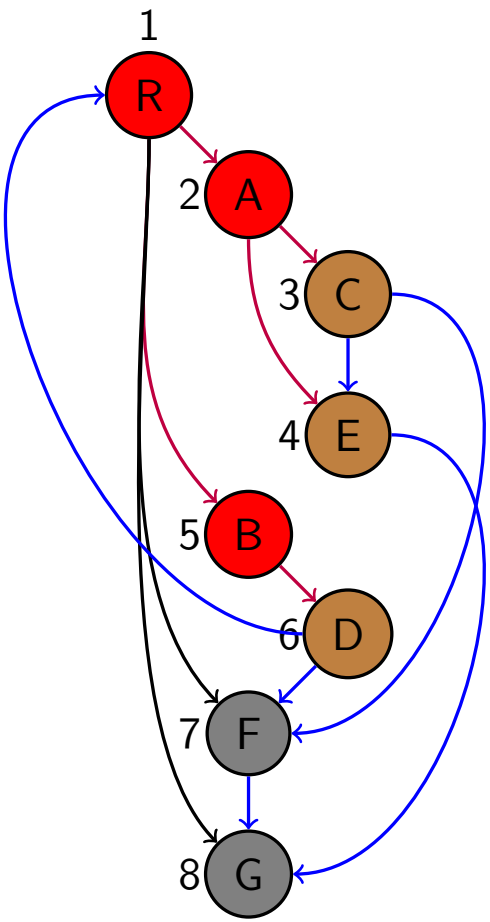
- Going back to translating R




```
loop
  block G
    block F
      do R
      ...
    end
    ...
```

Algorithm

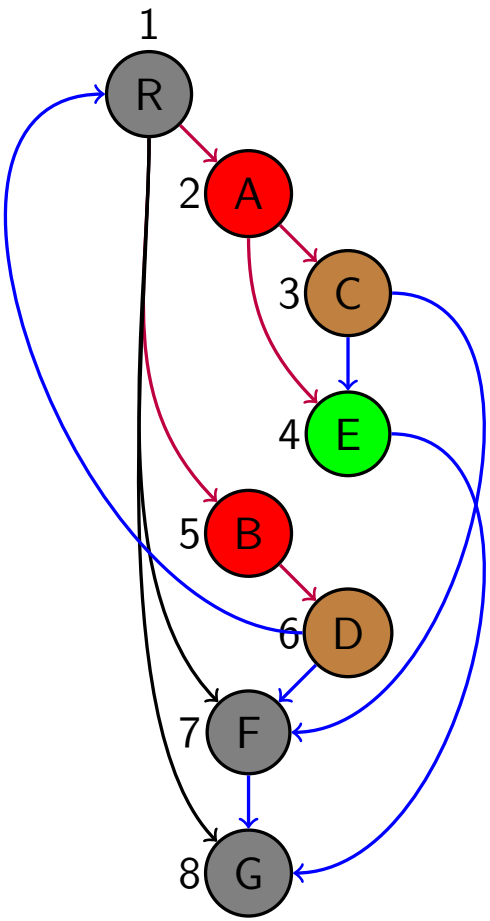
- Check out-going edges (A and B)




```
loop
 block G
  block F
   do R
   if R
    Translation of A
   else
    Translation of B
   end
  end
  ...
```

Algorithm

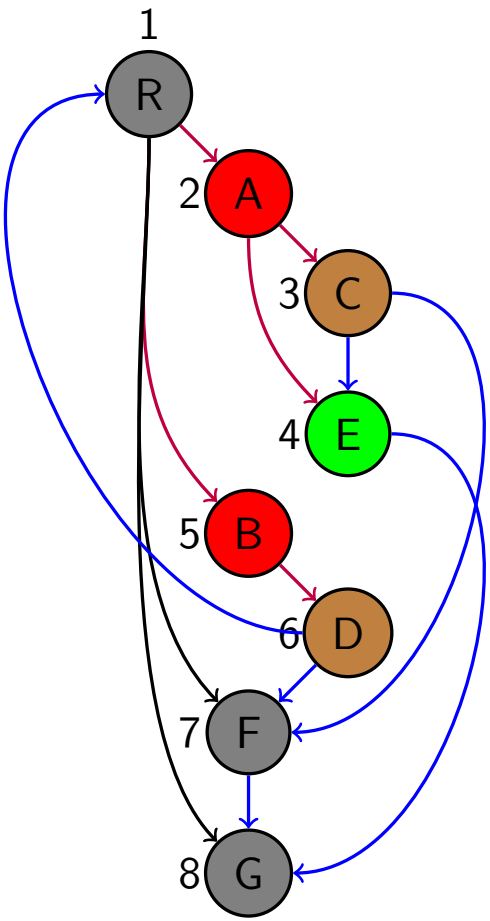
- Translating A, checking for merge vertices




```
...
if R
  Translation of A
else
  Translation of B
end
...
```

Algorithm

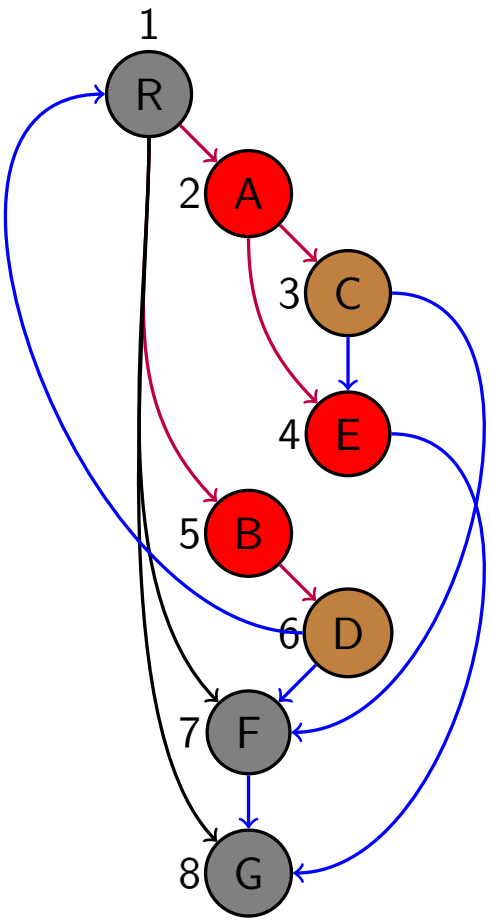
- Translating A, checking for merge vertices




```
...
if R
 Block E
  Translation of A
 end
 Translation of E
else
 Translation of B
end
...
```

Algorithm

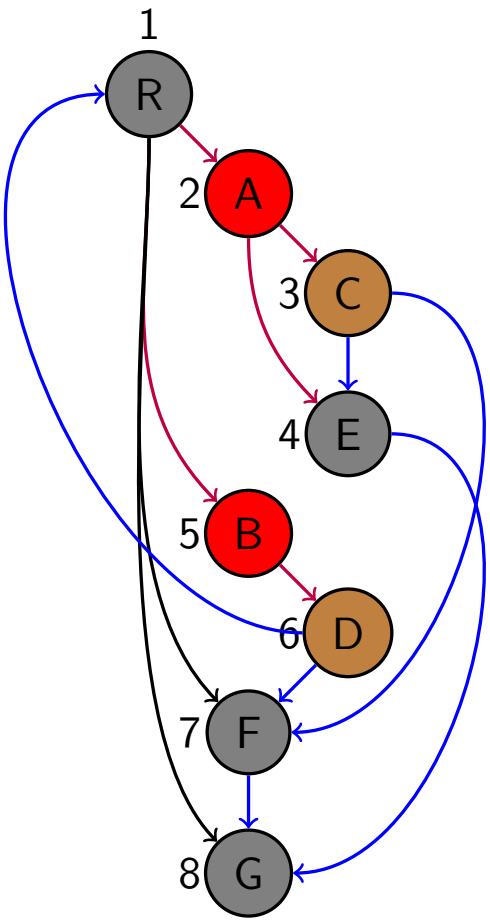
- Recursively translating E, place br G




```
...
if R
 Block E
  Translation of A
 end
 do E
 br G
else
 Translation of B
end
...
```

Algorithm

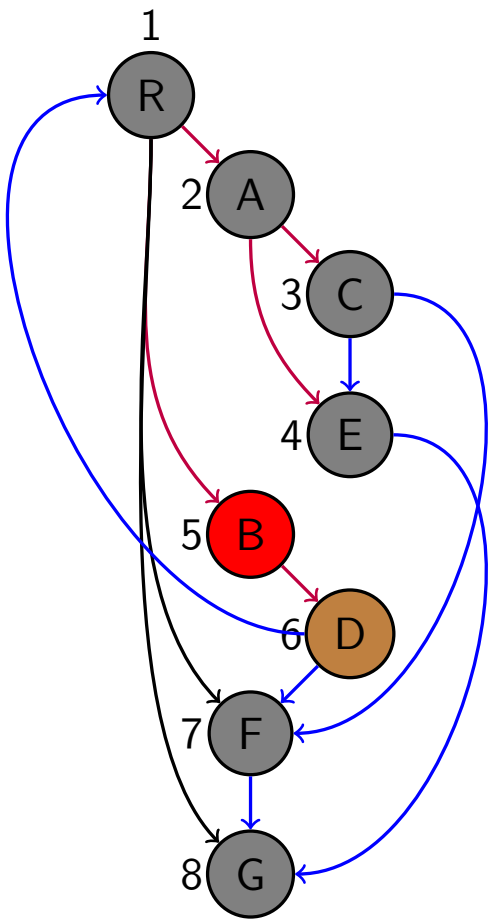
- Finish A, create if-else and place C




```
...
Block E
 do A
 if A
   Translation of C
 else
   br E
 end
end
...
```

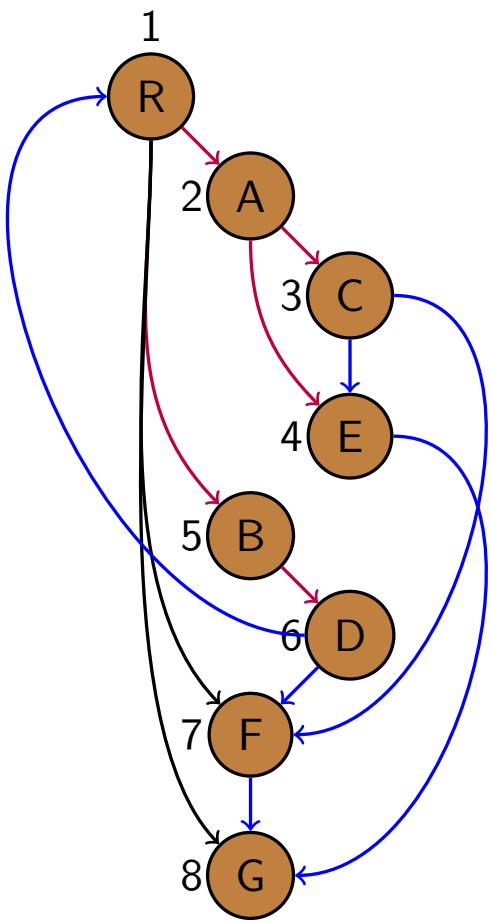
Algorithm

- Finish B and D




```
...
do B
do D
if D
  br R
else
  br F
end
```

Algorithm

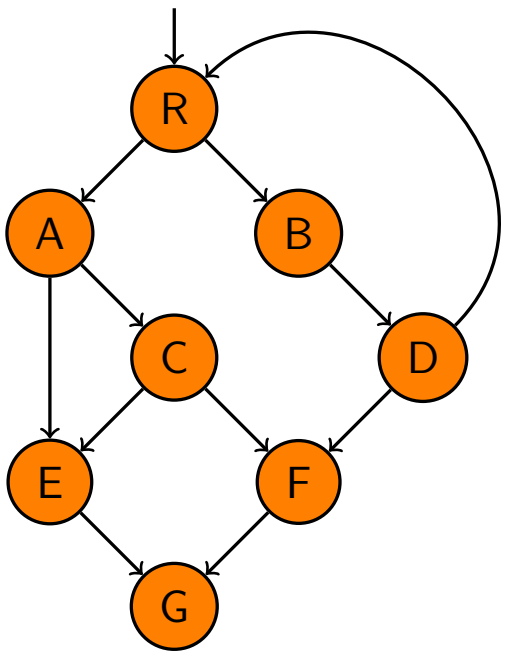



```
loop R
    Block G
        Block F
            Translation of R
            if
                Block E
                    Translation of A
                    if
                        Translation of C
                    else
                    end
                end
                Translation of E
            else
                Translation of B
                Translation of D
            end
        end
        Translation of F
    end
    Translation of G
end
```

Algorithm

Compute reverse post order numbering and dominator tree
Start with the root and traverse the dominator tree

1. Check for loop
   - Create loop structure
2. Check for merge vertices
   - Create block structure
   - Repeat for merge vertices
3. Translate vertex
4. Check out-going edges
   - Create if-then-else structure
5. Place children, branch or exit. Repeat for direct children



Sources


Thomas Lengauer and Robert Endre Tarjan.
A fast algorithm for finding dominators in a flowgraph.
ACM Transactions on Programming Languages and Systems (TOPLAS), 1(1):121–141, 1979.


Norman Ramsey.
Beyond relooper: recursive translation of unstructured control flow to structured control flow (functional pearl).
Proceedings of the ACM on Programming Languages, 6(ICFP):1–22, 2022.