Control Flow Analysis

Adapted from Lectures by
Prof. Saman Amarasinghe (MIT)

Outline

- Control-Flow Analysis
- Dominators
- Graph Traversal
- Reducible Graphs
- Interval Analysis

Constant Propagation

```
int sumcalc(int a, int b, int N)
{
    int i;
    int x, y;
    x = 0;
    y = 0;
    for(i = 0; i <= N; i++) {
       x = x + (4*a/b)*i + (i+1)*(i+1);
       x = x + b*y;
    }
    return x;
}
```

Constant Propagation

```
int sumcalc(int a, int b, int N)
{
    int i;
    int x, y;
    x = 0;
    y = 0;
    for(i = 0; i <= N; i++) {
       x⊗= x + (4*a/b)*i + (i+1)*(i+1);
       x = x + b*y;
    }
    return x;
}
```

Constant Propagation

```
int sumcalc(int a, int b, int N)
{
    int i;
    int x, y;
    x = 0;
    y = 0;
    for(i = 0; i <= N; i++) {
       x = x + (4*a/b)*i + (i+1)*(i+1);
       x = x + b*y;
    }
    return x;
}
```

Constant Propagation

```
int sumcalc(int a, int b, int N)
{
    int i;
    int x, y;
    x = 0;
    y = 0;
    for(i = 0; i <= N; i++) {
       x = x + (4*a/b)*i + (i+1)*(i+1);
       x = x + b*0;
    }
    return x;
}
```

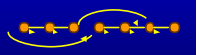
Implementing constant propagation

- Find an RHS expression which is a constant
- Replace the use of the LHS variable with the RHS constant given that:
  - All paths to the use of LHS variable passes the assignment of the LHS variable with the constant
  - There are no intervening definition of the LHS variable
- Need to know the "control-flow" of the program

Representing the control flow of the program

- Most instructions
  - execute the next instruction
    - straight line control-flow

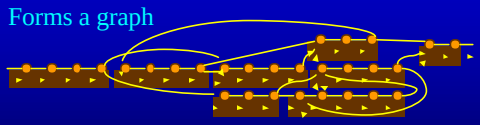
- Jump instructions
  - execute from different location
    - jump in control-flow



- Branch instructions
  - execute either the next instruction or from a different location
    - fork in the control-flow

Representing the control flow of the program

- Forms a graph



- Observation
  - a very large graph
  - lot of straight-line connections
  - simplify the graph by grouping some instructions

Basic Blocks

- A basic block is a maximal sequence of instructions such that
  - Only the first instruction can be reached from outside the basic block
  - All the instructions are executed consequetively if the first instruction is executed
    - No branch or jump instructions in the basic block, except the last instruction
    - No labels within the basic block, except before the first instruction

Control Flow Graph (CFG)

- Control-Flow Graph G = <N, E>
- Nodes(N): Basic Blocks
- Edges(E): (x,y) ∈ E iff first instruction in the basic block y follows the last instruction in the basic block x
  - First instruction in y is the target of branch or jump instruction (last instruction) in the basic block x
  - first instruction of y is next after the last instruction of x in memory and the last instruction of x is not a jump instruction

Control Flow Graph (CFG)

- Block with the first instruction of the procedure is the entry node (block with the procedure label)
- The blocks with the return instruction (jsr) are the exit nodes.
  - Can make a single exit node by adding a special node

Why do we perform Control-flow Analysis

- Loops are important to optimize
  - Programs spend a lot of times in loops and recursive cycles
  - Many special optimizations can be done on loops
- Programmers organize code using structured control-flow (if-then-else, for-loops etc.)
  - Optimizer can exploit this, once they are discovered

Challenges in Control-Flow Analysis

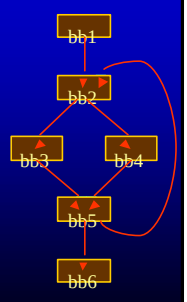
- Unstructured Control Flow
  - Use of goto's by the programmer
  - Only way to build certain control structures

```
L1: x = 0
    if (y > 0) goto L3
L2: if (y < 0) goto L1
L3: y = y + z
    goto L2
```

- Obscured control flow
  - method invocations
  - procedure variables
  - higher-order functions
  - jump tables

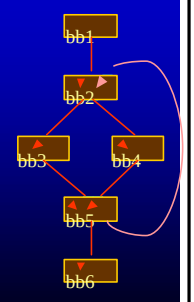
```
Myobject->run()
```


Identifying Recursive Structures Loops



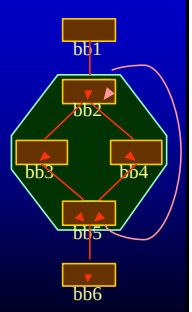
Identifying Recursive Structures Loops

- Identify back edges



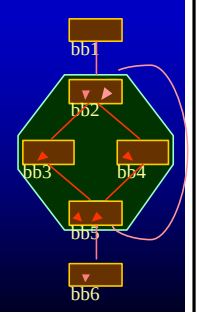
Identifying Recursive Structures Loops

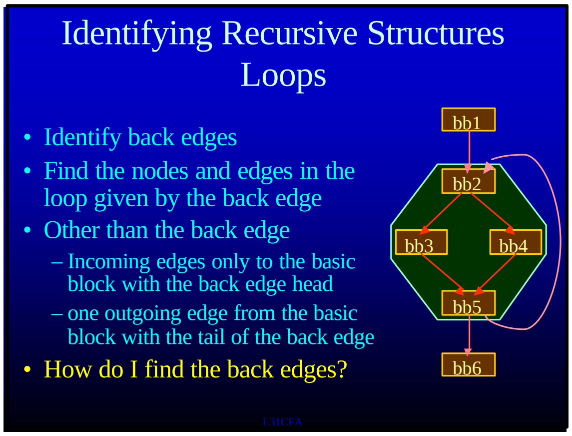
- Identify back edges
- Find the nodes and edges in the loop given by the back edge



Identifying Recursive Structures Loops

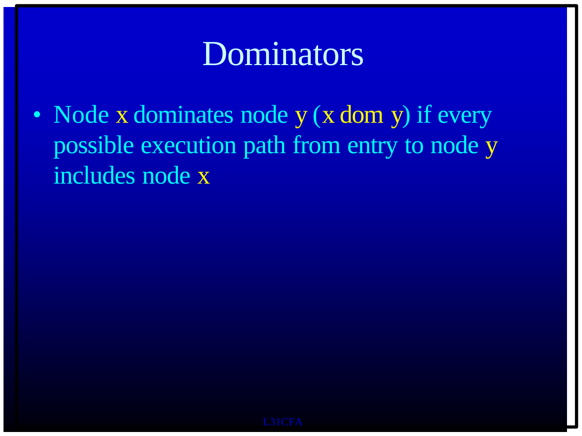
- Identify back edges
- Find the nodes and edges in the loop given by the back edge
- Other than the back edge
  - Incoming edges only to the basic block with the back edge head
  - one outgoing edge from the basic block with the tail of the back edge





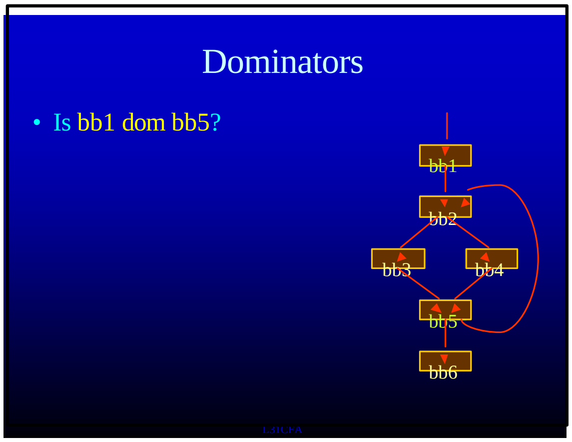
Identifying Recursive Structures Loops

- Identify back edges
- Find the nodes and edges in the loop given by the back edge
- Other than the back edge
  - Incoming edges only to the basic block with the back edge head
  - one outgoing edge from the basic block with the tail of the back edge
- How do I find the back edges?



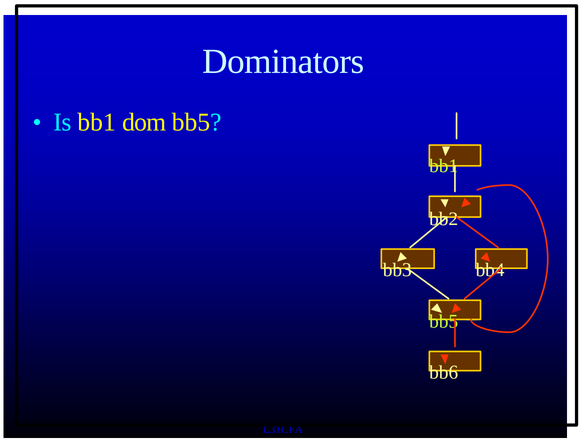
Dominators

- Node x dominates node y (x dom y) if every possible execution path from entry to node y includes node x



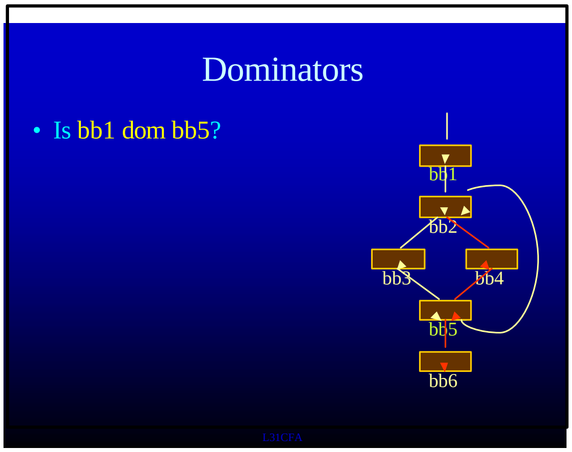
Dominators

- Is bb1 dom bb5?



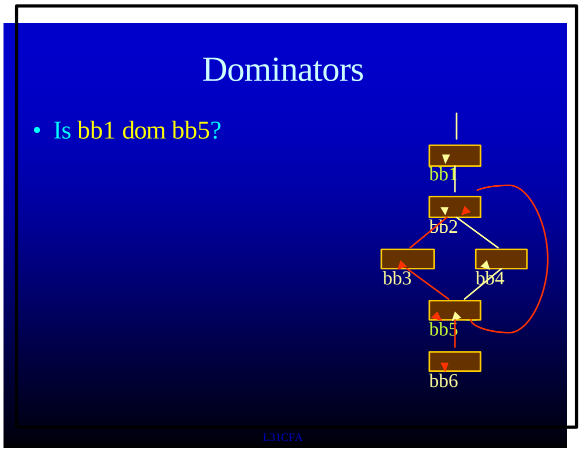
Dominators

- Is bb1 dom bb5?



Dominators

- Is bb1 dom bb5?

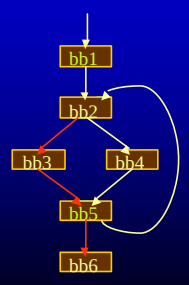


Dominators

- Is bb1 dom bb5?

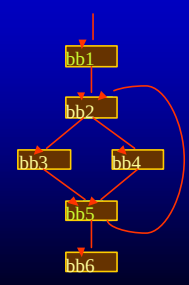
Dominators

- Is bb1 dom bb5?



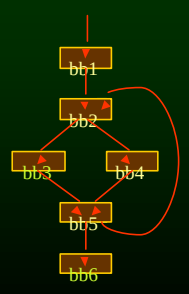
Dominators

- Is bb1 dom bb5? Yes!



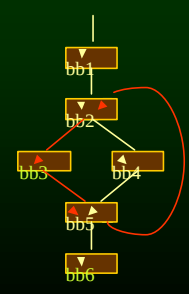
QUESTION

- Is bb1 dom bb5? Yes!
- Is bb3 dom bb6?



QUESTION

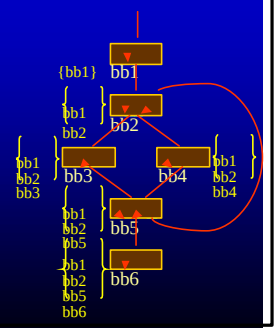
- Is bb1 dom bb5? Yes!
- Is bb3 dom bb6? No!

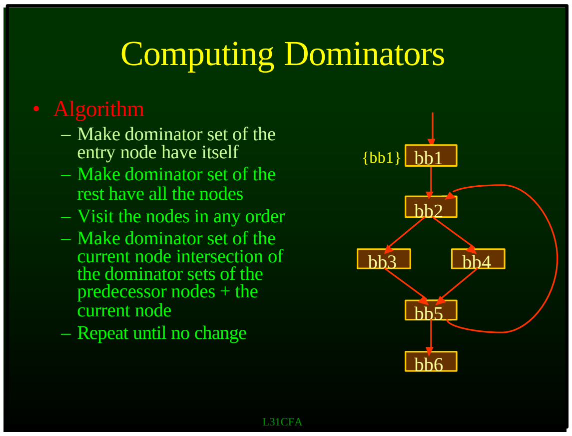


Computing Dominators

- a dom b iff
  - a = b or
  - a is the unique immediate predecessor of b or
  - a is a dominator of all immediate predecessor of b
- Algorithm
  - Make dominator set of the entry node have itself
  - Make dominator set of the rest have all the nodes
  - Visit the nodes in any order
  - Make dominator set of the current node intersection of the dominator sets of the predecessor nodes + the current node
  - Repeat until no change

Dominators (Desired Solution)

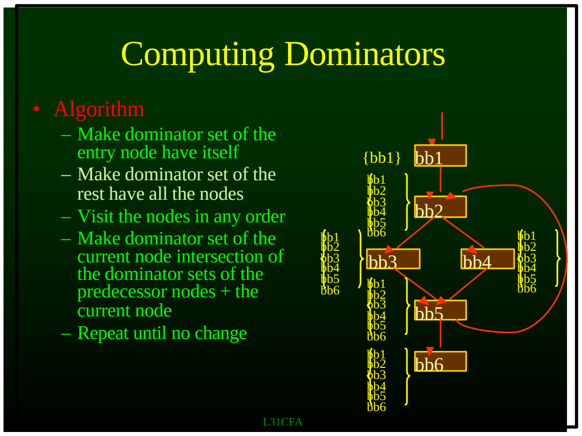




Computing Dominators

- Algorithm
  - Make dominator set of the entry node have itself
  - Make dominator set of the rest have all the nodes
  - Visit the nodes in any order
  - Make dominator set of the current node intersection of the dominator sets of the predecessor nodes + the current node
  - Repeat until no change

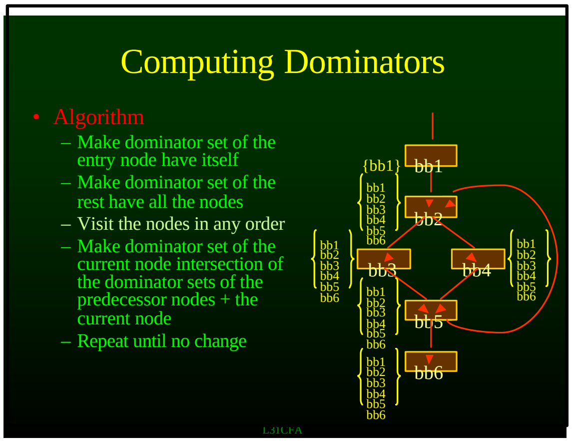
L31CFA



Computing Dominators

- Algorithm
  - Make dominator set of the entry node have itself
  - Make dominator set of the rest have all the nodes
  - Visit the nodes in any order
  - Make dominator set of the current node intersection of the dominator sets of the predecessor nodes + the current node
  - Repeat until no change

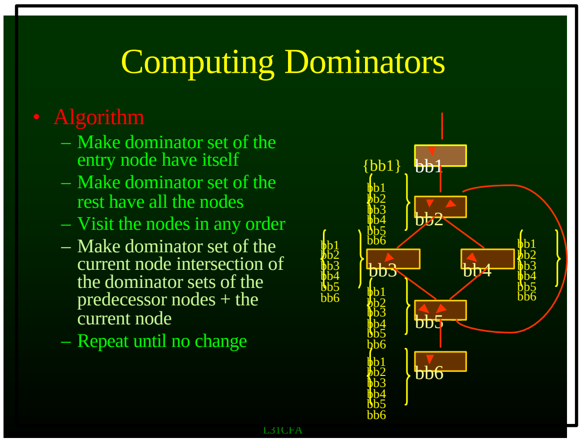
L31CFA



Computing Dominators

- Algorithm
  - Make dominator set of the entry node have itself
  - Make dominator set of the rest have all the nodes
  - Visit the nodes in any order
  - Make dominator set of the current node intersection of the dominator sets of the predecessor nodes + the current node
  - Repeat until no change

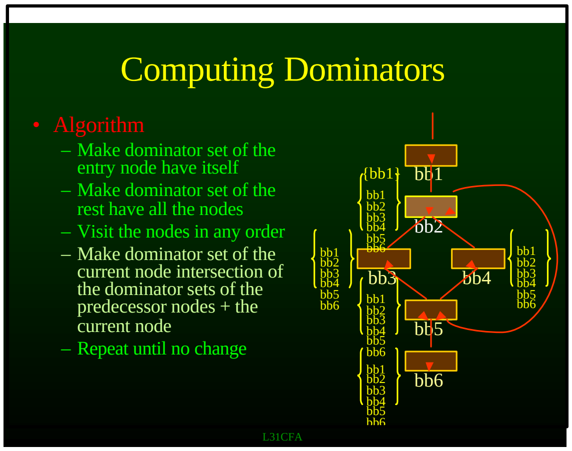
L31CFA



Computing Dominators

- Algorithm
  - Make dominator set of the entry node have itself
  - Make dominator set of the rest have all the nodes
  - Visit the nodes in any order
  - Make dominator set of the current node intersection of the dominator sets of the predecessor nodes + the current node
  - Repeat until no change

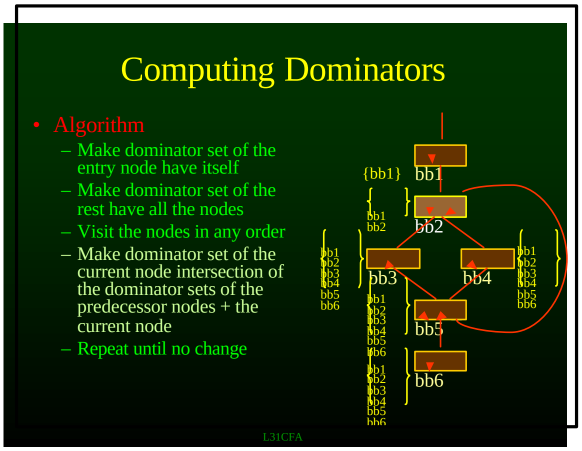
L31CFA



Computing Dominators

- Algorithm
  - Make dominator set of the entry node have itself
  - Make dominator set of the rest have all the nodes
  - Visit the nodes in any order
  - Make dominator set of the current node intersection of the dominator sets of the predecessor nodes + the current node
  - Repeat until no change

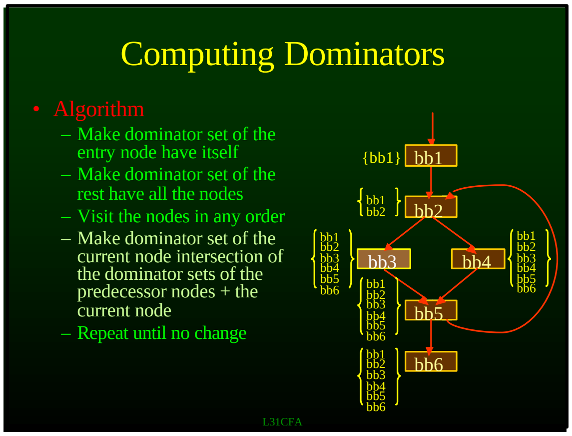
L31CFA



Computing Dominators

- Algorithm
  - Make dominator set of the entry node have itself
  - Make dominator set of the rest have all the nodes
  - Visit the nodes in any order
  - Make dominator set of the current node intersection of the dominator sets of the predecessor nodes + the current node
  - Repeat until no change

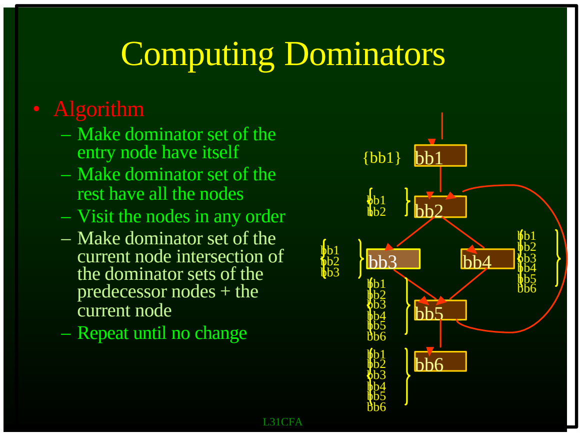
L31CFA



Computing Dominators

- Algorithm
  - Make dominator set of the entry node have itself
  - Make dominator set of the rest have all the nodes
  - Visit the nodes in any order
  - Make dominator set of the current node intersection of the dominator sets of the predecessor nodes + the current node
  - Repeat until no change

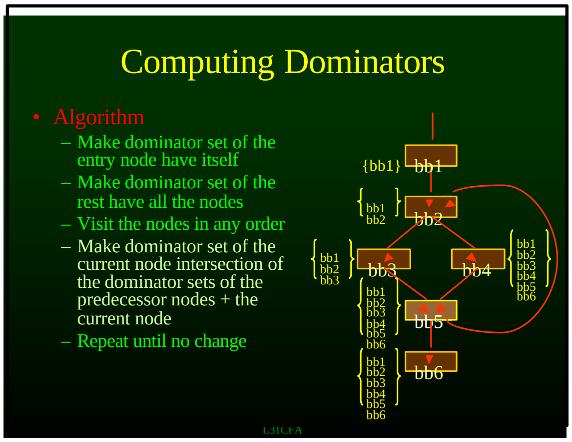
L31CFA



Computing Dominators

- Algorithm
  - Make dominator set of the entry node have itself
  - Make dominator set of the rest have all the nodes
  - Visit the nodes in any order
  - Make dominator set of the current node intersection of the dominator sets of the predecessor nodes + the current node
  - Repeat until no change

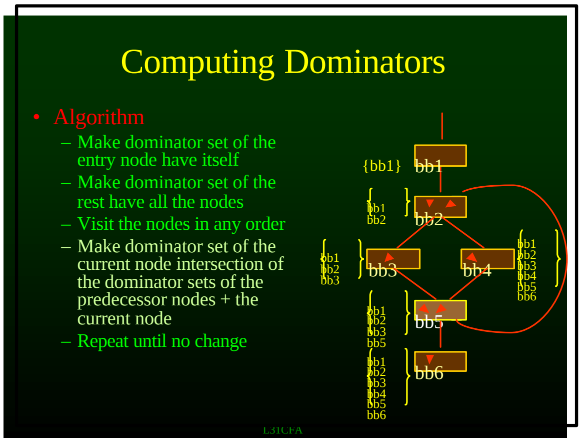
L31CFA



Computing Dominators

- Algorithm
  - Make dominator set of the entry node have itself
  - Make dominator set of the rest have all the nodes
  - Visit the nodes in any order
  - Make dominator set of the current node intersection of the dominator sets of the predecessor nodes + the current node
  - Repeat until no change

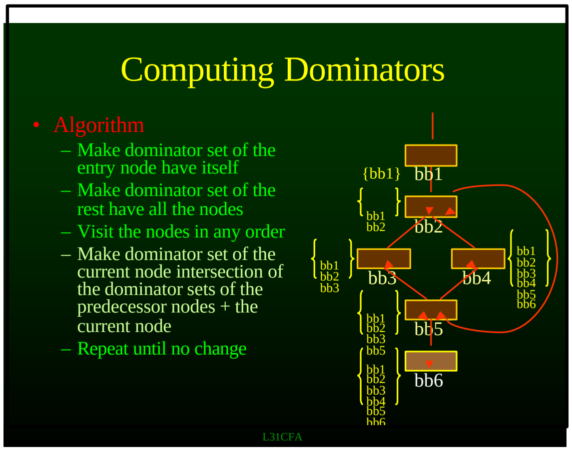
L31CFA



Computing Dominators

- Algorithm
  - Make dominator set of the entry node have itself
  - Make dominator set of the rest have all the nodes
  - Visit the nodes in any order
  - Make dominator set of the current node intersection of the dominator sets of the predecessor nodes + the current node
  - Repeat until no change

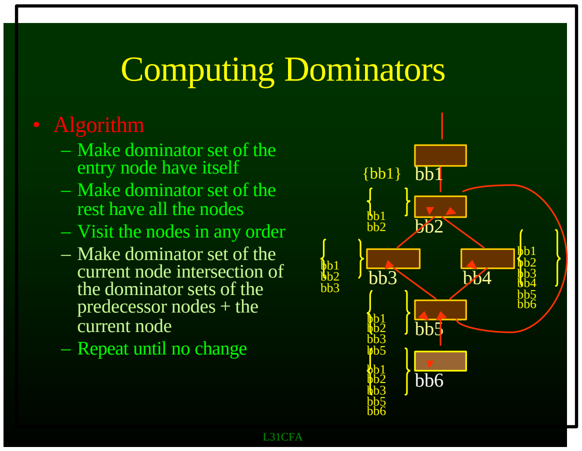
L31CFA



Computing Dominators

- Algorithm
  - Make dominator set of the entry node have itself
  - Make dominator set of the rest have all the nodes
  - Visit the nodes in any order
  - Make dominator set of the current node intersection of the dominator sets of the predecessor nodes + the current node
  - Repeat until no change

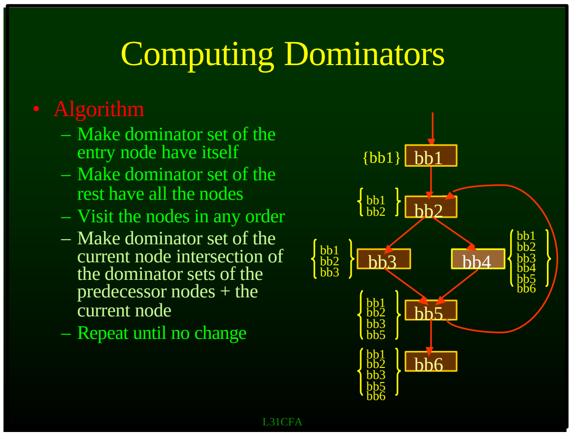
L31CFA



Computing Dominators

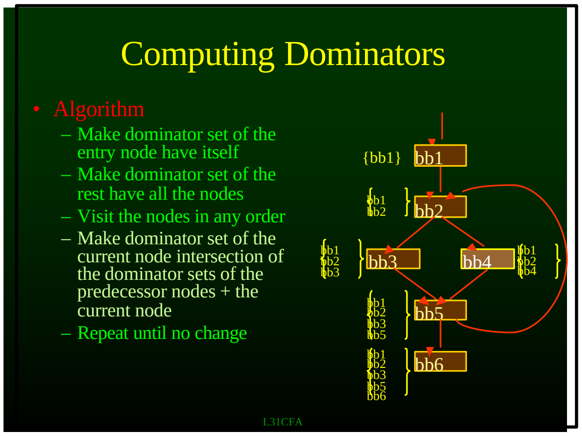
- Algorithm
  - Make dominator set of the entry node have itself
  - Make dominator set of the rest have all the nodes
  - Visit the nodes in any order
  - Make dominator set of the current node intersection of the dominator sets of the predecessor nodes + the current node
  - Repeat until no change

L31CFA



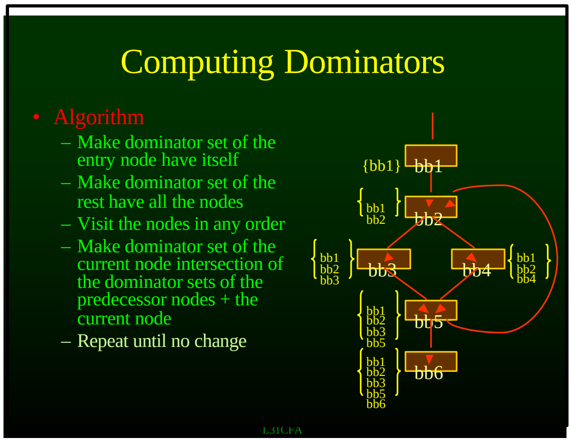
Computing Dominators

- Algorithm
  - Make dominator set of the entry node have itself
  - Make dominator set of the rest have all the nodes
  - Visit the nodes in any order
  - Make dominator set of the current node intersection of the dominator sets of the predecessor nodes + the current node
  - Repeat until no change



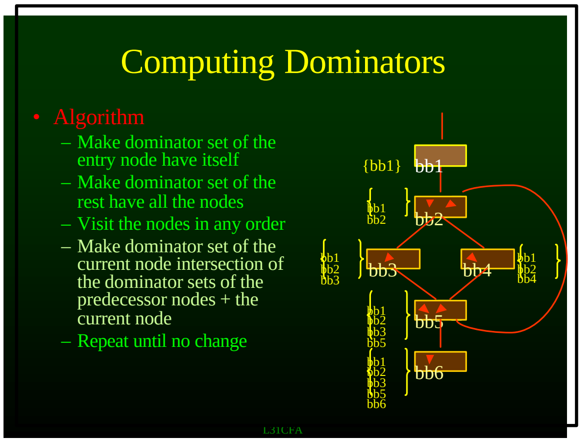
Computing Dominators

- Algorithm
  - Make dominator set of the entry node have itself
  - Make dominator set of the rest have all the nodes
  - Visit the nodes in any order
  - Make dominator set of the current node intersection of the dominator sets of the predecessor nodes + the current node
  - Repeat until no change



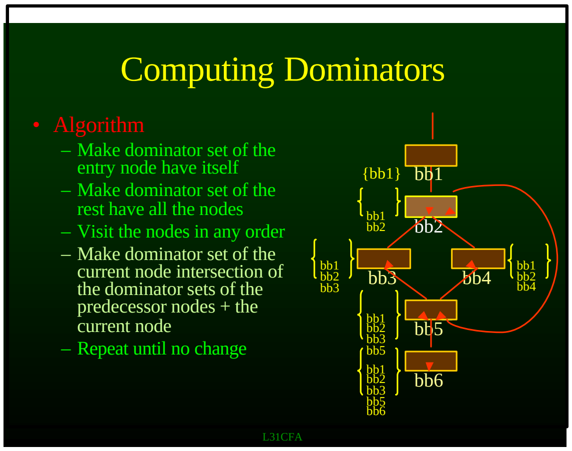
Computing Dominators

- Algorithm
  - Make dominator set of the entry node have itself
  - Make dominator set of the rest have all the nodes
  - Visit the nodes in any order
  - Make dominator set of the current node intersection of the dominator sets of the predecessor nodes + the current node
  - Repeat until no change



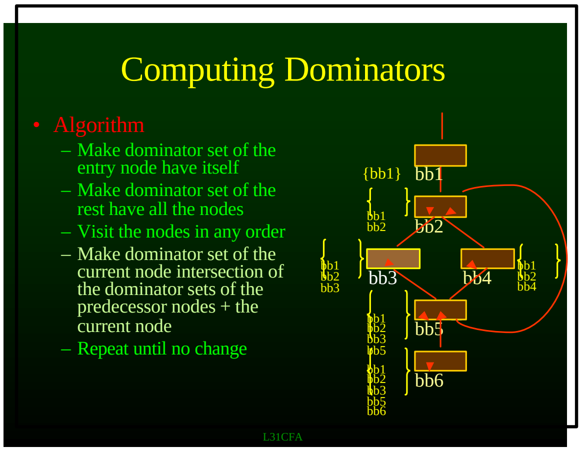
Computing Dominators

- Algorithm
  - Make dominator set of the entry node have itself
  - Make dominator set of the rest have all the nodes
  - Visit the nodes in any order
  - Make dominator set of the current node intersection of the dominator sets of the predecessor nodes + the current node
  - Repeat until no change



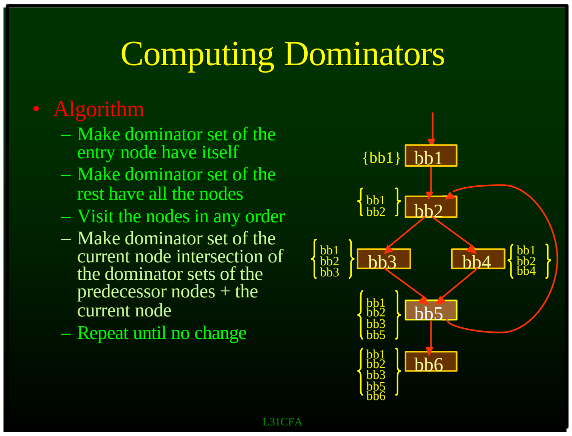
Computing Dominators

- Algorithm
  - Make dominator set of the entry node have itself
  - Make dominator set of the rest have all the nodes
  - Visit the nodes in any order
  - Make dominator set of the current node intersection of the dominator sets of the predecessor nodes + the current node
  - Repeat until no change



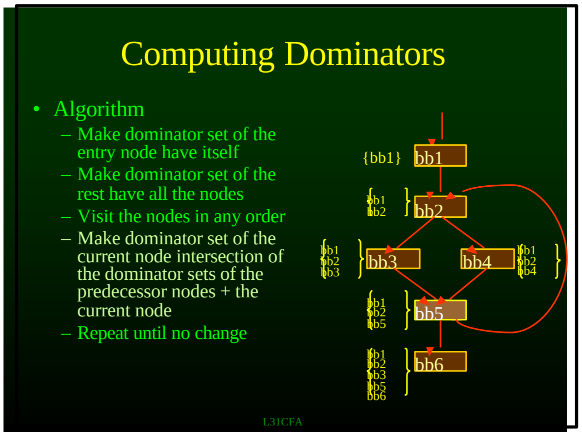
Computing Dominators

- Algorithm
  - Make dominator set of the entry node have itself
  - Make dominator set of the rest have all the nodes
  - Visit the nodes in any order
  - Make dominator set of the current node intersection of the dominator sets of the predecessor nodes + the current node
  - Repeat until no change



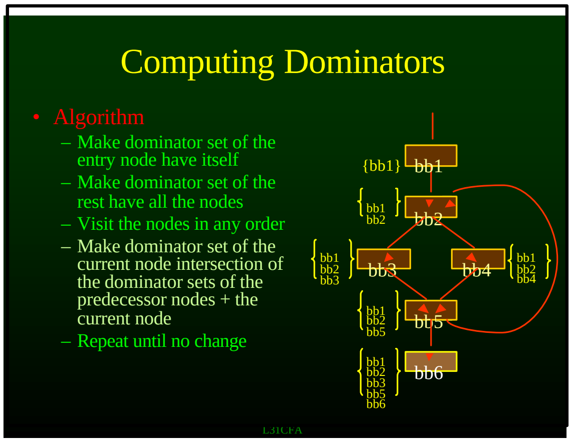
Computing Dominators

- Algorithm
  - Make dominator set of the entry node have itself
  - Make dominator set of the rest have all the nodes
  - Visit the nodes in any order
  - Make dominator set of the current node intersection of the dominator sets of the predecessor nodes + the current node
  - Repeat until no change



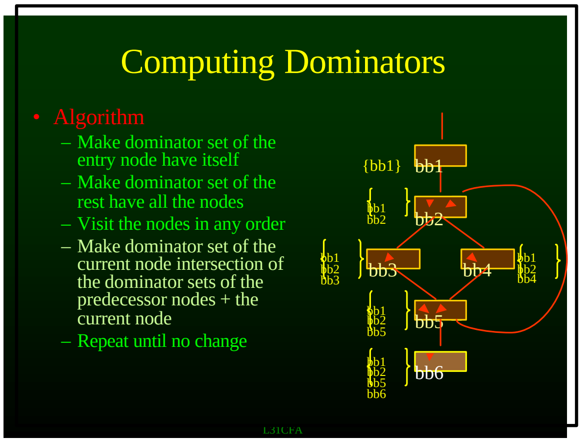
Computing Dominators

- Algorithm
  - Make dominator set of the entry node have itself
  - Make dominator set of the rest have all the nodes
  - Visit the nodes in any order
  - Make dominator set of the current node intersection of the dominator sets of the predecessor nodes + the current node
  - Repeat until no change



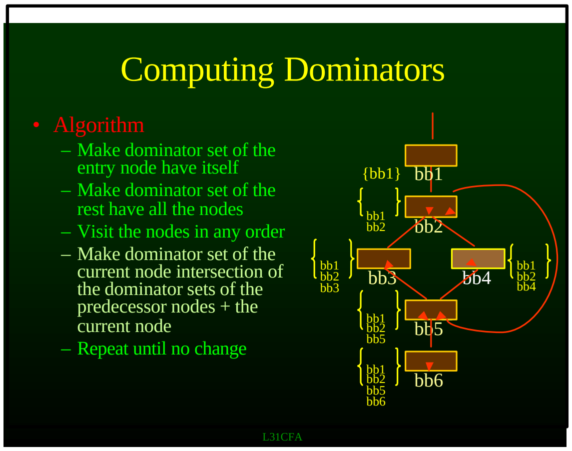
Computing Dominators

- Algorithm
  - Make dominator set of the entry node have itself
  - Make dominator set of the rest have all the nodes
  - Visit the nodes in any order
  - Make dominator set of the current node intersection of the dominator sets of the predecessor nodes + the current node
  - Repeat until no change



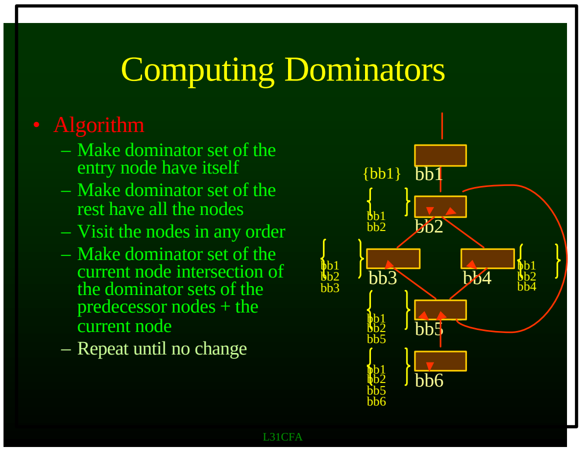
Computing Dominators

- Algorithm
  - Make dominator set of the entry node have itself
  - Make dominator set of the rest have all the nodes
  - Visit the nodes in any order
  - Make dominator set of the current node intersection of the dominator sets of the predecessor nodes + the current node
  - Repeat until no change



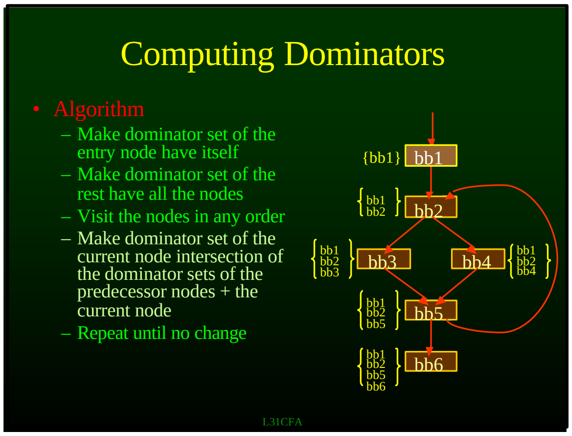
Computing Dominators

- Algorithm
  - Make dominator set of the entry node have itself
  - Make dominator set of the rest have all the nodes
  - Visit the nodes in any order
  - Make dominator set of the current node intersection of the dominator sets of the predecessor nodes + the current node
  - Repeat until no change



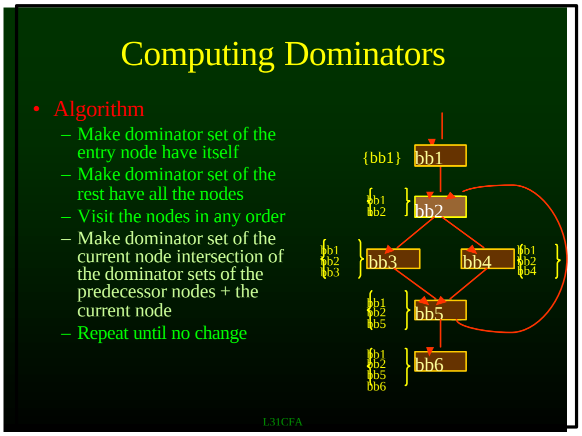
Computing Dominators

- Algorithm
  - Make dominator set of the entry node have itself
  - Make dominator set of the rest have all the nodes
  - Visit the nodes in any order
  - Make dominator set of the current node intersection of the dominator sets of the predecessor nodes + the current node
  - Repeat until no change



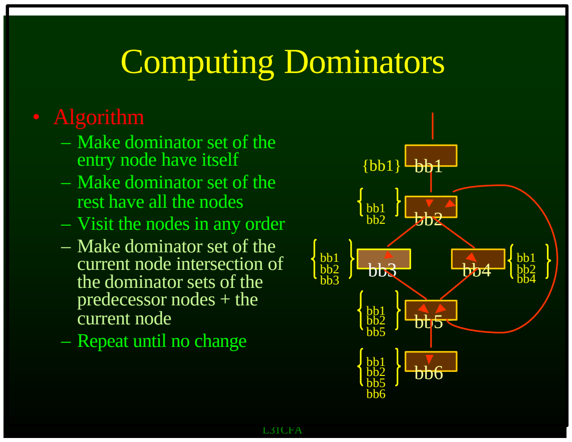
Computing Dominators

- Algorithm
  - Make dominator set of the entry node have itself
  - Make dominator set of the rest have all the nodes
  - Visit the nodes in any order
  - Make dominator set of the current node intersection of the dominator sets of the predecessor nodes + the current node
  - Repeat until no change



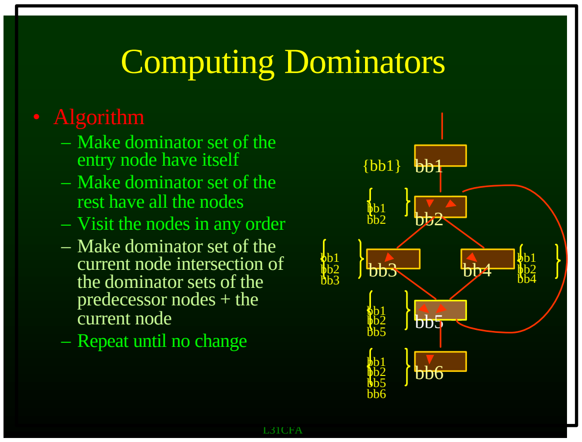
Computing Dominators

- Algorithm
  - Make dominator set of the entry node have itself
  - Make dominator set of the rest have all the nodes
  - Visit the nodes in any order
  - Make dominator set of the current node intersection of the dominator sets of the predecessor nodes + the current node
  - Repeat until no change



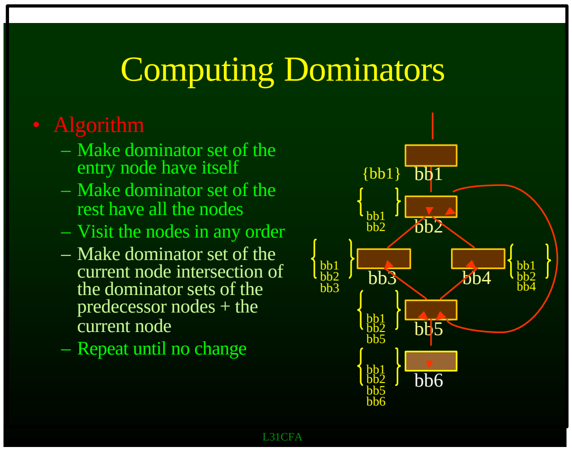
Computing Dominators

- Algorithm
  - Make dominator set of the entry node have itself
  - Make dominator set of the rest have all the nodes
  - Visit the nodes in any order
  - Make dominator set of the current node intersection of the dominator sets of the predecessor nodes + the current node
  - Repeat until no change



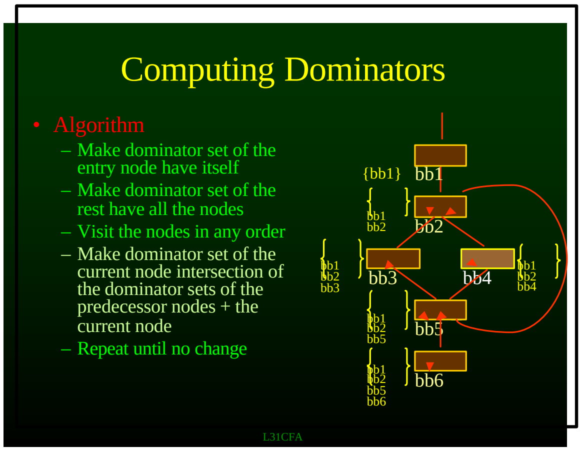
Computing Dominators

- Algorithm
  - Make dominator set of the entry node have itself
  - Make dominator set of the rest have all the nodes
  - Visit the nodes in any order
  - Make dominator set of the current node intersection of the dominator sets of the predecessor nodes + the current node
  - Repeat until no change



Computing Dominators

- Algorithm
  - Make dominator set of the entry node have itself
  - Make dominator set of the rest have all the nodes
  - Visit the nodes in any order
  - Make dominator set of the current node intersection of the dominator sets of the predecessor nodes + the current node
  - Repeat until no change

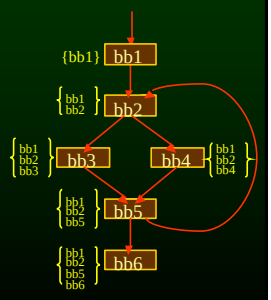


Computing Dominators

- Algorithm
  - Make dominator set of the entry node have itself
  - Make dominator set of the rest have all the nodes
  - Visit the nodes in any order
  - Make dominator set of the current node intersection of the dominator sets of the predecessor nodes + the current node
  - Repeat until no change

Computing Dominators

- Algorithm
  - Make dominator set of the entry node have itself
  - Make dominator set of the rest have all the nodes
  - Visit the nodes in any order
  - Make dominator set of the current node intersection of the dominator sets of the predecessor nodes + the current node
  - Repeat until no change

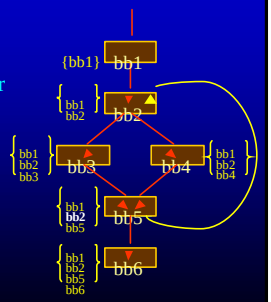


Computing Dominators

- What we just witnessed was an iterative data-flow analysis algorithm in action
  - Initialize all the nodes to a given value
  - Visit nodes in some order
  - Calculate the node's value
  - Repeat until no value changes

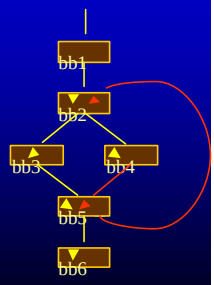
What is a back edge?

- An edge (x, y) $\in$ E is a back edge iff y dom x
  - is node y in the dominator set of node x



Traversing the CFG

- Depth-First Traversal
  - Visit all the descendants of a node before visiting any siblings
- Depth-first spanning tree
  - a set of edges corresponding to a depth-first visitation of CFG

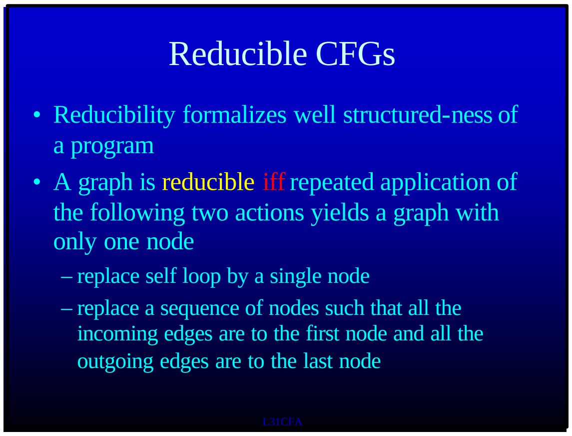


Preorder and Postorder

- In preorder traversal, each node is processed before its descendants in the depth-first tree
- In postorder traversal, each node is processed after its descendants in the depth-first tree

Outline

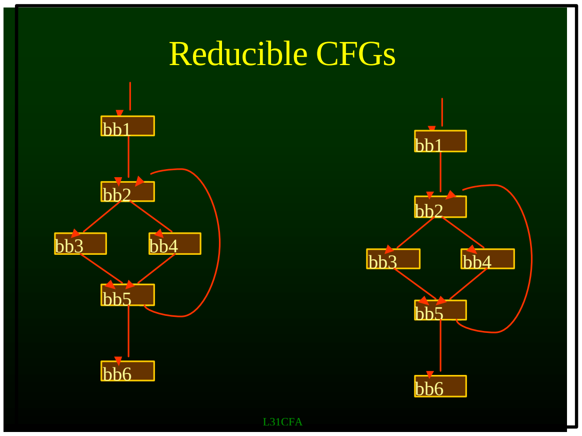
- Control-Flow Analysis
- Dominators
- Graph Traversal
- Reducible Graphs
- Interval Analysis



Reducible CFGs

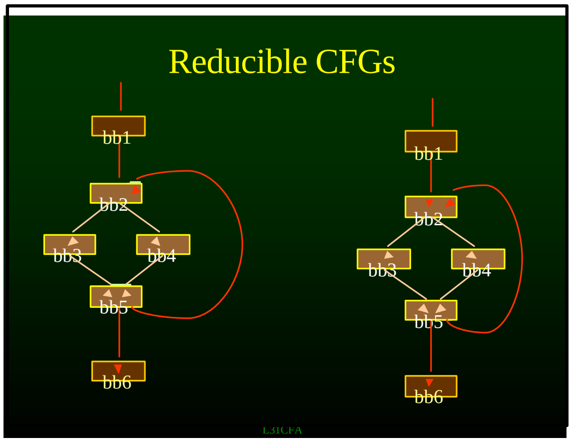
- Reducibility formalizes well structured-ness of a program
- A graph is reducible iff repeated application of the following two actions yields a graph with only one node
  - – replace self loop by a single node
  - – replace a sequence of nodes such that all the incoming edges are to the first node and all the outgoing edges are to the last node

L31CFA



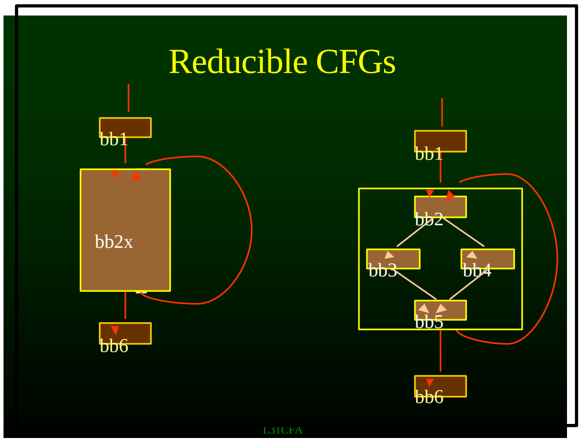
Reducible CFGs

L31CFA



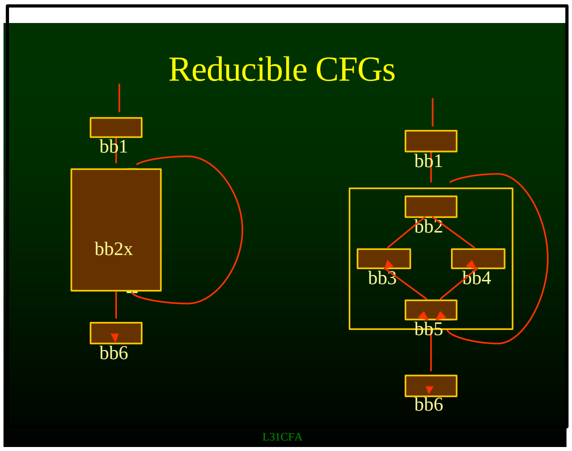
Reducible CFGs

L31CFA



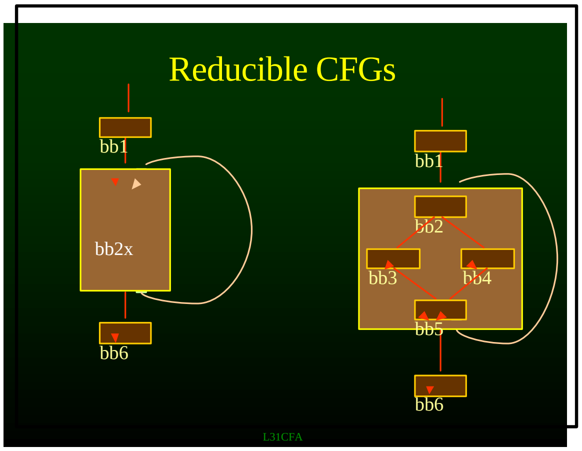
Reducible CFGs

L31CFA



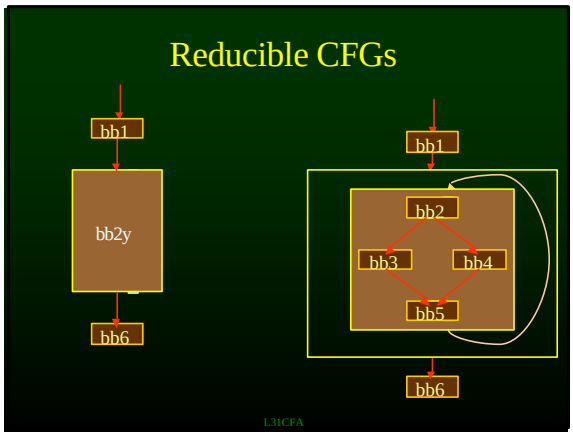
Reducible CFGs

L31CFA

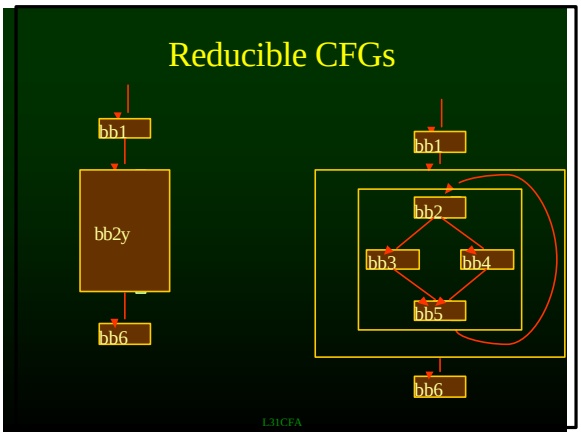


Reducible CFGs

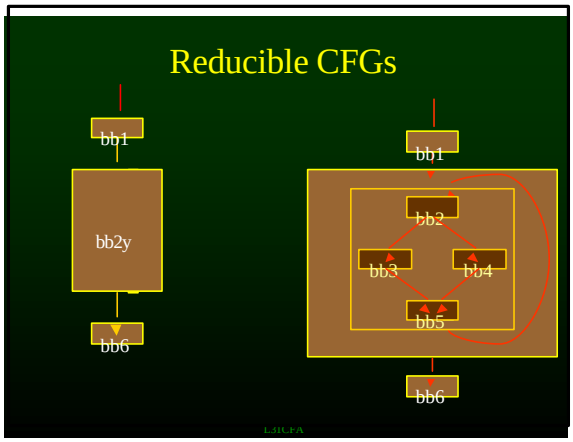
L31CFA



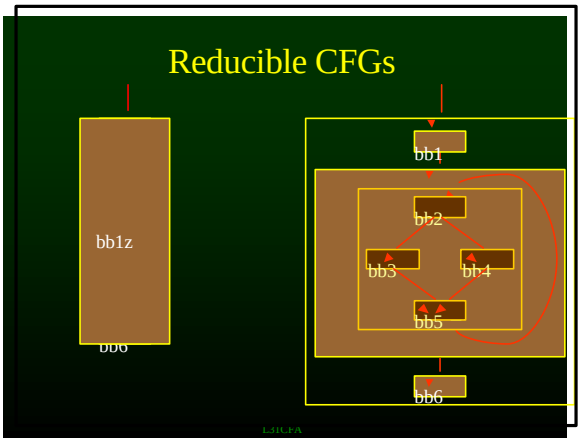
Reducible CFGs



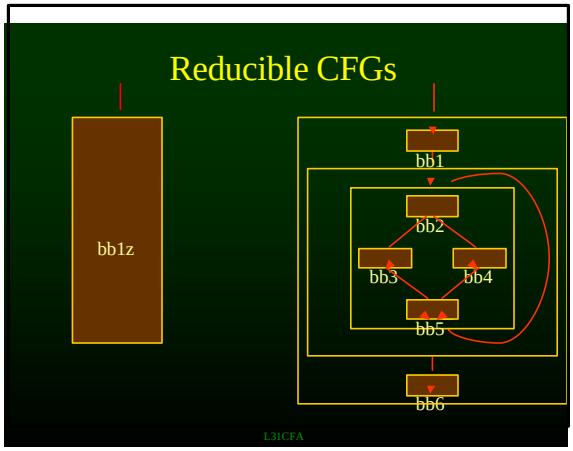
Reducible CFGs



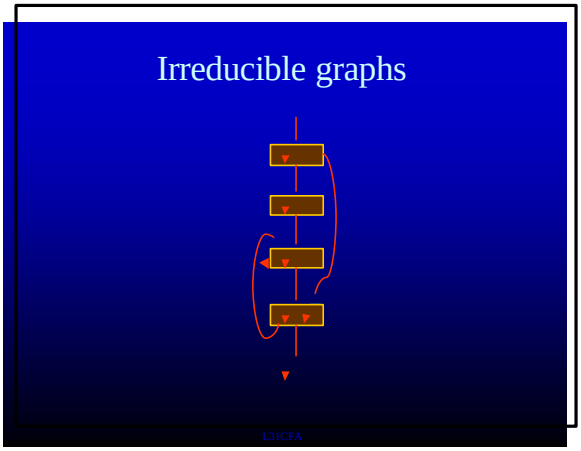
Reducible CFGs



Reducible CFGs

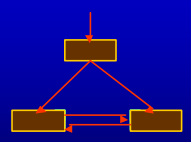


Reducible CFGs



Irreducible graphs

Irreducible graphs



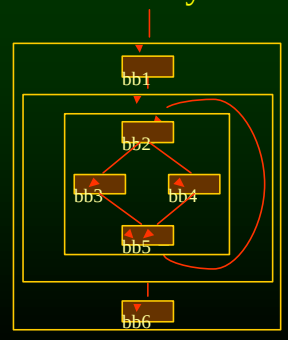
Approaches of Control-Flow Analysis

- Iterative Analysis
  - Use a CFG
  - Propagate values
  - Iterate until no change
- Interval Based Analysis
  - Use a reducible CFG
  - Calculate in hierarchical graphs
  - No iterations (faster)

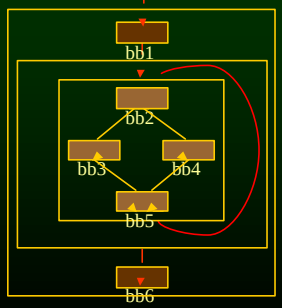
Interval Based Analysis

- If a node does not include a graph:
  - calculate value
- If a node includes a graph
  - Calculate values of the nodes in that graph
  - Propagate values (no back edges ⇒ no iteration)
  - Use entry (or exit) value as the value of the enclosing node

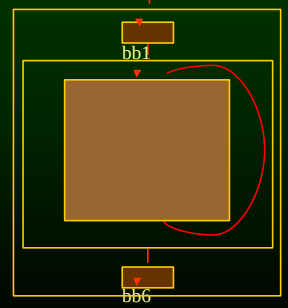
Interval Analysis



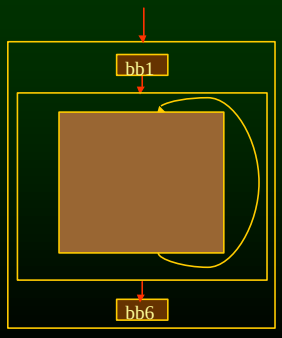
Interval Analysis



Interval Analysis

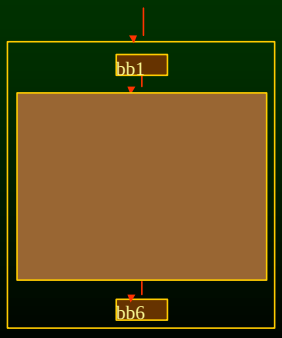


Interval Analysis



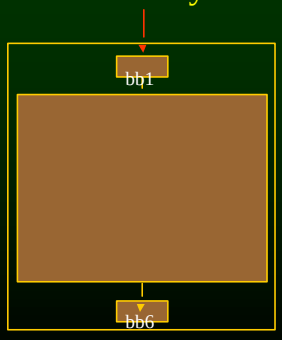
L31CFA

Interval Analysis



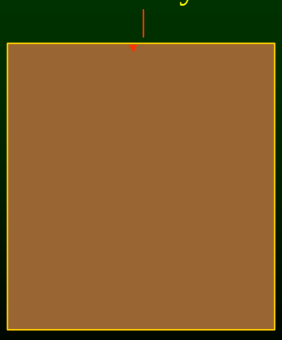
L31CFA

Interval Analysis



L31CFA

Interval Analysis



L31CFA

Outline

- Control-Flow Analysis
- Dominators
- Graph Traversal
- Reducible Graphs
- Interval Analysis

L31CFA