Outline

• CFA and a first example: dominators

• Another example of CFA: Post-dominators

• Example of CFA and CFT: basic block merging and splitting
Control Flow Analysis

• Storing order ≠ executing order

• Control Flow Analyses are designed to understand the possible execution paths (control flows) while ignoring data values and operations/operators

• We need to identify all possible control flows between instructions

• We need to identify all possible control flows between basic blocks

• Let’s look at an example of CFA
Control Flow Graph

... ... y = 0

After executing this basic block
This other basic block might be executed

y = 3

x = y ...
...
Sometimes “may” isn’t enough

How can I know that a given basic block will be executed no matter what?

This is what our first CFA computes.
Dominators

**Definition:** Node $d$ dominates node $n$ in a CFG ($d \ dom \ n$) iff every control flow from the start node to $n$ goes through $d$. Every node dominates itself.

What is the relation between instructions within a basic block?

What is the relation between instructions in different basic blocks?

It depends on the CFG

In other words, dominators depend on the control flows
Dominators

**Definition:** Node $d$ dominates node $n$ in a CFG ($d \ dom \ n$) iff every control flow from the start node to $n$ goes through $d$. Every node dominates itself.

What are the dominators of basic blocks 1 and 2?

What are the dominators of basic blocks 1, 2, and 3?
Dominators

**Definition:** Node $d$ dominates node $n$ in a CFG ($d \text{ dom } n$) iff every control flow from the start node to $n$ goes through $d$. Every node dominates itself.

What are now the dominators of basic blocks 1, 2, and 3?
Now that we know what we want to obtain (the dominance binary relation between basic blocks),

let us define an algorithm (a CFA) that computes it.
A CFA to find dominators

Consider a block n with k predecessors p₁, ..., pk

**Observation 1:** if d dominates each pᵢ (1≤i≤k), then d dominates n

**Observation 2:** if d dominates n, then it must dominate all pᵢ

\[ D[n] = \{n\} \cup (\cap_{p \in \text{predecessors}(n)} D[p]) \]

To compute it:
- By iteration
- Initialize each D[n] to include every one

This is our first CFA

*Notice: this CFA does not depend on values and/or operations/operators*
Dominance

CFG

1
2
3

Dominators

1
2
3
We can now introduce new concepts based on the dominator relation
Strict dominance

**Definition:**
a node $d$ strictly dominates $n$ iff

- $d$ dominates $n$ and
- $d$ is not $n$

![Diagram](image_url)
Immediate dominators

**Definition:** the immediate dominator of a node $n$ is the unique node that strictly dominates $n$ but does not strictly dominate another node that strictly dominates $n$.
Immediate dominators

**Definition:** the immediate dominator of a node $n$ is the unique node that strictly dominates $n$ but does not strictly dominate another node that strictly dominates $n$.

**CFG**

**Dominator tree**

- **1**
  - **2**
  - **3**

**Strict dominators**

- **1**
  - **2**
  - **3**

**Immediate dominators**

- **1**
  - **2**
  - **3**
Dominators in LLVM

```cpp
#include "llvm/IR/Dominators.h"

bool runOnFunction (Function &F) override {
  errs() << "=== Dominators\n";
  DominatorTree &DT = getAnalysis<DomTreeWrapperPass>().getDomTree();
  for (auto & bb : F){
    auto inst = bb.begin();
    errs() << *inst << "\n";
    auto instNode = DT.getNode(&bb);
    for (auto child : instNode->getChildren()){        
      auto dominatedBB = child->getBlock();
      auto dominatedInst = dominatedBB->begin();
      errs() << " -> " << *dominatedInst << "\n";
    }
  }
  return false;
}

void getAnalysisUsage(AnalysisUsage &AU) const override {
  AU.addRequired<DomTreeWrapperPass>();
  AU.setPreservesAll();
}
```

```cpp
3 int main(int argc, char *argv[]) {
4  printf("START\n");
5  if (argc > 0){
6    printf("THEN\n");
7  } else {
8    printf("ELSE\n");
9  }
10  printf("END\n");
11  if (argc == 0){
12    return 0;
13  }
14 }
15 return 1;
16 }
```
Dominators in LLVM

```c
#include "llvm/IR/Dominators.h"

bool runOnFunction (Function &F) override {
  errs() << "=== Dominators\n".
  DominatorTree &DT = getAnalysis<DominatorTreeWrapperPass>().getDomTree();
  for (auto &bb : F) {
    auto inst = bb.begin();
    errs() << "*" << inst << "\n";
    auto instNode = DT.getNode(&bb);
    for (auto child : instNode->getChildren()) {
      auto dominatedBB = child->getBlock();
      auto dominatedInst = dominatedBB->begin();
      errs() << "-> " << "dominatedInst << "\n";
    }
  }
  return false;
}

void getAnalysisUsage(AnalysisUsage &AU) const override {
  AU.addRequired<DominatorTreeWrapperPass>();
  AU.setPreservesAll();
}
```

Notice the order
Dominators in LLVM: example 2

```c
#include "llvm/IR/Dominators.h"

bool runOnFunction (Function &F) override {
    errs() << "--- Dominators\n";

    DominatorTree &DT = getAnalysis<DominatorTreeWrapperPass>().getDomTree();

    for (auto& bb : F){
        auto inst = bb.begin();
        errs() << *inst << "\n";
        auto instNode = DT.getNode(&bb);

        for (auto child : instNode->getChildren()){  
            auto dominatedBB = child->getBlock();
            auto dominatedInst = dominatedBB->begin();
            errs() << " - -> " << *dominatedInst << "\n";
        }
    }
    return false;
}

void getAnalysisUsage(AnalysisUsage &AU) const override {
    AU.addRequired<DominatorTreeWrapperPass>();
    AU.setPreservesAll();
}

3 int main(int argc, char *argv[]) {
    printf("START\n");

    if (argc > 0){
        printf("THEN\n");

        if (argc > 20){
            printf("Inside THEN\n");

            if (argc > 40){
                printf("Inside the inside of THEN\n");
            }
        }
    }
    printf("END\n");
    return 1;
}
```
Dominators in LLVM: example 2

```c
#include "llvm/IR/Dominators.h"

bool runOnFunction (Function &F) override {
    errs() << "=== Dominators\n";
    DominatorTree &DT = getAnalysis<DominatorTreeWrapperPass>().getDomTree();

    for (auto &bb : F){
        auto inst = bb.begin();
        errs() << "inst << ""\n";  
        auto instNode = DT.getNode(&bb);
        for (auto child : instNode->getChildren()){  
            auto dominatedBB = child->getBlock();
            auto dominatedInst = dominatedBB->begin();
            errs() << " -> " " << "dominatedInst << ""\n";  
        }
    }
    return false;
}

doAnalysisUsage(AnalysisUsage &AU) const override {
    AU.addRequired<DominatorTreeWrapperPass>();
    AU.setPreservesAll();
}
```

What is going to be the output?

Is it correct?
LLVM-specific notes for dominators

• bool DominatorTree::dominates (...)
  • bool dominates (Instruction *i, Instruction *j)
    Return true if the basic block that includes i is an immediate dominator
    of the basic block that includes j
  • bool dominates (Instruction *i, BasicBlock *b)
    Return true if the basic block that includes i is an immediate dominator of b

• If the first argument (either instruction or basic block)
  is not reachable from the entry point of the function, return false

• If the second argument (either instruction or basic block)
  is not reachable from the entry point of the function, return true
Outline

• CFA and a first example: dominators

• Another example of CFA: Post-dominators

• Example of CFA and CFT: basic block merging and splitting
Post-dominators

Assumption: Single exit node in CFG

Definition: Node $d$ post-dominates node $n$ in a graph iff every path from $n$ to the exit node goes through $d$
Post-dominators

Assumption: Single exit node in CFG

Definition: Node $d$ post-dominates node $n$ in a graph iff every path from $n$ to the exit node goes through $d$

```
B: if (par1 > 5)
C: varX = par1 + 1
C2: ...
D: print(varX)
```
Post dominators in LLVM

```cpp
#include "llvm/Analysis/PostDominator.h"

bool runOnFunction (Function &F) override {
  errs() << "--- Post dominators\n";

  PostDominatorTree & PDT = getAnalysis<PostDominatorTreeWrapperPass>().getPostDomTree();

  for (auto & bb : F){
    auto inst = bb.begin();
    errs() << inst << "\n";
    auto instNode = PDT.getNode(&bb);
    for (auto child : instNode->getChildren()){
      auto dominatedBB = child->getBlock();
      auto dominatedInst = dominatedBB->begin();
      errs() << " -> " << *dominatedInst << "\n";
    }
  }

  return false;
}

void getAnalysisUsage(AnalysisUsage &AU) const override {
  AU.addRequired<PostDominatorTreeWrapperPass>();
  AU.setPreservesAll();
}
```
Post dominators in LLVM

```c
#include "llvm/Analysis/PostDominators.h"

bool runOnFunction (Function *F) override {
  errs() << "=== Post dominators\n";
  PostDominatorTree & PDT = getAnalysis<PostDominatorTreeWrapperPass>().getPostDomTree();
  for (auto & bb : F){
    auto inst = bb.begin();
    errs() << inst << "\n";
    auto instNode = PDT.getNode(&bb);
    for (auto child : instNode->getChildren()){*
      auto dominatedBB = child->getBlock();
      auto dominatedInst = dominatedBB->begin();
      errs() << "- " << *dominatedInst << "\n";
    }
  }
  return false;
}

void getAnalysisUsage(AnalysisUsage &AU) const override {
  AU.addRequired<PostDominatorTreeWrapperPass>();
  AU.setPreservesAll();
}
```

What is going to be the output?
LLVM-specific notes for post dominators

• bool PostDominatorTree::dominates (...)
  • bool dominates (Instruction *i, Instruction *j)
    Return true if the basic block that includes i is an immediate post-dominator of the basic block that includes j
  • bool dominates (Instruction *i, BasicBlock *b)
    Return true if the basic block that includes i is an immediate post-dominator of b

• If the first argument (either instruction or basic block) is not reachable from the entry point of the function, return false

• If the second argument (either instruction or basic block) is not reachable from the entry point of the function, return true
LLVM-specific notes for *dominators

DominatorTreeBase
::bool dominates(...)
Outline

• CFA and a first example: dominators

• Another example of CFA: Post-dominators

• Example of CFA and CFT: basic block merging and splitting
Another example of CFA (and CFT)

A homework of this class could be the following one: design and implement an algorithm to implement this CFA
- CFA: it says whether it is safe to merge two basic blocks
- CFT: it merges only the basic block pairs identified by the CFA

Existing LLVM pass: simplifycfg

goto L1
L1: call printf()
return
call printf()
return

CFA
The two basic blocks can be merged

CFT
return

call printf()

This is a simple CFA and CFG, but useful after applying several other code transformations
Another example of CFA

- What are the possible equivalent CFGs the compiler can choose from?
- The compiler needs to be able to transform CFGs
  - CFAs tell the compiler what are the equivalent CFGs

```c
... 
if (b == 2) {
    return;
}
#endif
```

```bash
clang myfile.c -DCRAZY -o myprog
```

```
... 
if (b == 2) {
    return;
}
```

```
clang myfile.c -DCRAZY -o myprog
```

```
... 
return
```

```
... 
b == 2
```

```
... 
return
```

```
# ifdef CRAZY
printf(“Yep”);
#endif
... 
return
```
Critical edges

Definition:
A critical edge is an edge in the CFG which is neither the only edge leaving its source block, nor the only edge entering its destination block.

These edges must be split: a new block must be created and inserted in the middle of the edge, to insert computations on the edge without affecting any other edges.