CFA

Simone Campanoni
simone.campanoni@northwestern.edu
Outline

• CFA and a first example: dominators

• Another example of CFA: dominance frontier

• 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

After executing this basic block
This other basic block might be executed
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 \((d \ dom \ n)\) 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 $p_1$, ..., $p_k$

**Observation 1:** if $d$ dominates each $p_i$ ($1 \leq i \leq k$), then $d$ dominates $n$

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

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

To compute it:
- By iteration
- Initialize each $D[n]$ to ?
A CFA to find dominators

Consider a block $n$ with $k$ predecessors $p_1, \ldots, p_k$

Observation 1: if $d$ dominates each $p_i$ ($1 \leq i \leq k$), then $d$ dominates $n$

Observation 2: if $d$ dominates $n$, then it must dominate all $p_i$

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

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 $n$
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 $n$.
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();
}
```
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();
}
```

What is going to be the output?

Notice the order

You cannot assume any 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();
}
```
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();
}
```

What is going to be the output?
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
  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
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
D: print(varX)

How to compute post-dominators?
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)
```

![CFG and Immediate post-dominator tree diagram]
Post dominators in LLVM

```c
#include "llvm/Analysis/PostD dominators\n"

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();
}
```

```c
int main(int argc, char *argv[]){
  printf("START\n");
  if (argc > 0){
    printf("THEN\n");
  } else {
    printf("ELSE\n");
  }
  printf("END\n");
  if (argc == 0){
    return 0;
  }
  return 1;
}
```
Post dominators in LLVM

#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() << "\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();
}
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 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: dominance frontier

• Example of CFA and CFT: basic block merging and splitting
A problem: deciding where the place PHIs

- Problem:
  we would like to map a stack location into a set of IR variables

CFG

%\texttt{p} = \texttt{alloca} ...

\begin{align*}
\text{store} \ %\texttt{v}1, \ %\texttt{p} \\
\text{...} \\
\text{store} \ %\texttt{v}2, \ %\texttt{p} \\
\text{store} \ %\texttt{v}3, \ %\texttt{p} \\
\text{B1} &\quad \text{B2} &\quad \text{B4} \\
\text{B3} &\quad \text{B5} \\
%\texttt{v}9 = \%\texttt{load} \ %\texttt{p} \\
\end{align*}

\begin{align*}
\text{store} \ %\texttt{v}1, \ %\texttt{p} \\
\text{...} \\
\text{store} \ %\texttt{v}2, \ %\texttt{p} \\
\text{store} \ %\texttt{v}3, \ %\texttt{p} \\
\text{B1} &\quad \text{B2} &\quad \text{B4} \\
\text{B3} &\quad \text{B5} \\
%\texttt{v}9 = \texttt{PHI} \ [\%\texttt{v}1, \ B3] \ [?, \ B5] \\
\end{align*}
A problem: deciding where the place PHIs

• Problem:
  we would like to map a stack location into a set of IR variables

%p = alloca ...

store %v1, %p
...%v9 = %load %p

store %v2, %p
B1
B2

store %v3, %p
B4
B5

B0

%v9 = PHI [%v2, ...
%v4 = PHI [%v2, ...
%v3, ...

B1
B2
B3
B5

%v9 = PHI [%v1, B3] [%v4, B5]

How can we identify where to insert PHI?
A problem: deciding where the place PHIs

• Problem:
  we would like to map a stack location into a set of IR variables

• Solutions:
  • Simple:
    insert PHI in all basic blocks for all variables (expensive)
  • Smarter:
    for each variable, identify the subset of basic blocks that need PHI
Dominance frontier

• Dominators of block \( N \) tell us which basic blocks **must** be executed prior to \( N \)

• We need to identify blocks “just after” those blocks that are dominated by \( N \)

• **Definition:**
The Dominance Frontier of a basic block \( N \), \( DF(N) \), is the set of all blocks that are immediate successors to blocks dominated by \( N \), but which aren’t strictly dominated by \( N \)

\[
DF(B0) = \{B2\}
\]

**Definition**  
**Where to insert PHI**
Dominance frontier

• **Definition:**
  The Dominance Frontier of a basic block $N$, $DF(N)$, is the set of all blocks that are immediate successors to blocks dominated by $N$, but which aren’t strictly dominated by $N$.

• $DF(N)$ includes a basic block $X$ if and only if
  1. $N$ dominates a predecessor of $X$
  2. $N$ does not strictly dominate $X$

• How can we compute $DF(N)$?
Dominance frontier computation

1. From the CFG

2. Compute the dominators tree

3. Compute the local dominance frontier for all nodes

\[ \text{DF}_{\text{local}}[N] = \text{successors of } N \text{ in the CFG that are not strictly dominated by } N \]

3. Compute the dominance frontier

\[ \text{DF}[N] = \text{DF}_{\text{local}}[N] \cup \bigcup_{c \in \text{children}(N)} \text{DF}_{\text{up}}[c] \]
Dominance frontier computation

Compute the dominance frontier

\[ \text{DF}[N] = \text{DF}_{\text{local}}[N] \cup_{c \in \text{children}(N)} \text{DF}_{\text{up}}[c] \]

\( \text{DF}_{\text{local}}[N] \) = successors of \( N \) in the CFG that are not strictly dominated by \( N \)

\( \text{DF}_{\text{up}}[c] \) = nodes in \( \text{DF}[c] \) that are not strictly dominated by parent(c) of the dominator tree

\[
\begin{align*}
\text{DF}_{\text{local}}[B0] &= \{\} & \text{DF}[B0] &= \{\} \\
\text{DF}_{\text{local}}[B1] &= \{\} & \text{DF}[B1] &= \{\} \\
\text{DF}_{\text{local}}[B2] &= \{\} & \text{DF}[B2] &= \{\} \\
\text{DF}_{\text{local}}[B3] &= \{\text{B4}\} & \text{DF}[B3] &= \{\text{B4}\} \\
\text{DF}_{\text{local}}[B4] &= \{\} & \text{DF}[B4] &= \{\}
\end{align*}
\]
Dominance frontier computation

Compute the dominance frontier

\[ \text{DF}[N] = \text{DF}_{\text{local}}[N] \cup_{c \in \text{children}(N)} \text{DF}_{\text{up}}[c] \]

\[ \text{DF}_{\text{local}}[N] = \text{successors of } N \text{ in the CFG that are not strictly dominated by } N \]

\[ \text{DF}_{\text{up}}[c] = \text{nodes in } \text{DF}[c] \text{ that are not strictly dominated by } \text{parent}(c) \text{ of the dominator tree} \]

\[
\begin{align*}
\text{DF}_{\text{local}}[B0] &= \{\} & \text{DF}[B0] &= \{\} \\
\text{DF}_{\text{local}}[B1] &= \{\} & \text{DF}[B1] &= \{\} \\
\text{DF}_{\text{local}}[B2] &= \{\} & \text{DF}[B2] &= \{\} \\
\text{DF}_{\text{local}}[B3] &= \{B4\} & \text{DF}[B3] &= \{B4\} \\
\text{DF}_{\text{local}}[B4] &= \{\} & \text{DF}[B4] &= \{\}
\end{align*}
\]
Dominance frontier computation

Compute the dominance frontier

$$DF[N] = DF_{local}[N] \bigcup_{c \in \text{children}(N)} DF_{up}[c]$$

$DF_{local}[N] = $ successors of $N$ in the CFG that are not strictly dominated by $N$

$DF_{up}[c] = $ nodes in $DF[c]$ that are not strictly dominated by parent$(c)$ of the dominator tree

```
<table>
<thead>
<tr>
<th>B0</th>
<th>B1</th>
<th>B2</th>
<th>B3</th>
<th>B4</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

```
<table>
<thead>
<tr>
<th>B0</th>
<th>B1</th>
<th>B2</th>
<th>B3</th>
<th>B4</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

- $DF_{local}[B0] = \{\}$
- $DF_{local}[B1] = \{\}$
- $DF_{local}[B2] = \{\}$
- $DF_{local}[B3] = \{B4\}$
- $DF_{local}[B4] = \{\}$

- $DF[B0] = \{\}$
- $DF[B1] = \{\}$
- $DF[B2] = \{\}$
- $DF[B3] = \{B4\}$
- $DF[B4] = \{\}$

37
Dominance frontier computation

Compute the dominance frontier

\[ \text{DF}[^N] = \text{DF}_\text{local}[^N] \cup \bigcup_{c \in \text{children}(N)} \text{DF}_\text{up}[c] \]

\[ \text{DF}_\text{local}[^N] = \text{successors of } [N] \text{ in the CFG that are not strictly dominated by } [N] \]

\[ \text{DF}_\text{up}[c] = \text{nodes in } \text{DF}[c] \text{ that are not strictly dominated by parent}(c) \text{ of the dominator tree} \]
If I have a re-definition in B0:
• No need for a phi

If I have a re-definition in B1:
• I need to add a phi in B4

This is how mem2reg decides where to inject phis for alloca that can be safely removed
Outline

• CFA and a first example: dominators

• Another example of CFA: dominance frontier

• 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

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

Existing LLVM pass: simplifycfg
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

... If (b == 2){
    return;
}
#endif CRAZY
printf(“Yep”);
#endif
return;

clang myfile.c –DCRAZY –o myprog

... if (b == 2)
  return
... b == 2
  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.

```
If (...){
  while (...){
    ...
  }
}
A()
```
Always have faith in your ability

Success will come your way eventually

Best of luck!