Final exam

• Final exam will cover \textit{all} material covered in course – i.e., all compiler phases
  – Focus will be on after-midterm material

• There will be some theoretical questions on the special topics

• 2hr exam, similar in format to midterm

• P6, H10 assigned soon
Optimization Frameworks
Roadmap

• Last time:
  – Optimization overview
    • Soundness and completeness
  – Simple optimizations
    • Peephole
    • LICM

• This time:
  – More Optimization
    • Analysis frameworks
Outline

• Review Dominators
• Introduce some more advanced concepts
  – SSA
  – Dataflow propagation
DOMINATOR REVIEW
Dominator terms

• Domination (A dominates B):
  – to reach block B, you must have gone through block A

• Strict Domination (A strictly dominates B)
  – A dominates B and A is not B

• Immediate Domination (A immediately dominates B)
  – A immediately dominates B if A dominates B and has no intervening dominators
Dominator example
Dominance frontier

- Definition: For a block $X$, the set of nodes $Y$ such that $X$ dominates an immediate predecessor of $Y$ but does not strictly dominate $Y$
STATIC SINGLE ASSIGNMENT
SSA Key idea

- We’d like to build an intermediate representation of the program in which each variable gets a value in at most 1 program point:

\[\begin{align*}
    x &= 1 \\
    x &= 2 \\
    y &= 3 \\
    x &= 1 \\
    z &= 2 \\
    y &= 3 \\
    x &= y \\
    z &= y \\
    w &= z \\
    i &= 0; \\
    \text{while}(i < 10)\{ \\
       k &= i + 1; \\
    \} \\
\end{align*}\]
Conversion

• We’ll make new variables to carry over the effect of the original program

\[
\begin{align*}
x &= 1 \\
x &= x \\
y &= x
\end{align*}
\]

\[
\begin{align*}
x_1 &= 1 \\
x_2 &= x_1 \\
y_1 &= x_2
\end{align*}
\]
Benefits

- There are some obvious advantages to this format for program analysis
  - Easy to see the *live range* of a variable at a program point (i.e. the definitions that could reach that point)
    - Places with the variable on the LHS
  - Easy to see when a variable is *dead* (i.e. unused)
    - Never used on the RHS of a statement
Optimizations where SSA helps

• Dead Code Elimination

```c
int a = 9;
int b = 2;
if (g < 12) {
    a = 1;
} else {
    if (b < 4) {
        a = 2;
    } else {
        a = 3;
    }
}
}
b = a;
return 2;
```
Optimizations where SSA helps

• Constant propagation / constant folding

```c
int a = 30;
int b = 9; - (a / 5);
int c;
c = b; 4;
if (true) {
c = a; 10;
}
return c * 2;
```
What about conditionals

\[
x = 5 \\
x = x - 1 \\
x < 3
\]

\[
y = x \times 2 \\
w = y
\]

\[
w = x - y \\
z = x + y
\]

\[
x_1 = 5 \\
x_2 = x_1 - 1 \\
x_2 < 3
\]

\[
y_1 = x_2 \times 2 \\
w_1 = y_1
\]

\[
w_2 = y_1 - x \\
z = x + y
\]

Which y to use?
Phi functions

- We’ll introduce a special symbol $\Phi$ that represents a join like this
- Takes arguments of all variables to join
- Returns the “correct” one
- Do we need a $\Phi$ for $x$?
  - Nope!
Computing phi functions

• Intuitively, we want to figure out cases where there are multiple assignments that can reach a node

• To be safe, we can place $\Phi$ functions for each assignment at every node in the *dominance frontier*
Pruned phi functions

• This causes a bunch of useless $\Phi$ functions
  – Cases where the result is never used

• **Pruned SSA** is a version where statically removable $\Phi$ nodes are never used
DATAFLOW FRAMEWORKS
Dataflow framework idea

• Many analyses can be formulated as how data is transformed over the control flow graph
  – Propagate static information from:
    • the beginning of a single basic block
    • the end of a single basic block
    • The join points of multiple basic blocks
Dataflow framework idea

• Meet Lattice
• Transfer function
  – How data is propagated from one end of a basic block to the other
• Meet operation
  – Means of combining lattice between blocks
Dataflow analysis direction

• Forward analysis
  – Start at the beginning of a function’s CFG, work along the control edges

• Backwards analysis
  – Start at the end of a function’s CFG, work against the control edges

• Continuously propagate values until there is no change
Dataflow example 1

- Available Expression analysis
  - Whether an expression that has been previously computed may be reused
  - Forward dataflow problem: from expression to points of re-use
  - Meet Lattice:
    - Meet operation:
      - AND of all predecessors
    - At the beginning of each block, everything is True

* This causes some problems for loops
Dataflow example 2

- Very Busy Expression analysis
  - An expression is very busy at a point $p$ if it is guaranteed that it will be computed at some time in the future

- Backwards dataflow problem: from computation to use
- Meet Lattice:

```
True
|   
|False
```

- Meet operation: AND
The end: or is it?

• Covered a broad range of topics
  – Some formal concepts
  – Some practical concepts

• What we skipped
  – Linking and loading
  – Interpreters
  – Register allocation
  – Performance analysis / Proofs