**Final Review**

**Today**
- Overview of what we have learned so far
- SSA review
- pointer and alias analysis
- interprocedural analysis and optimization
- loop transformations and Fourier-Motzkin
- affine partitionings for parallelism
- induction variable elimination review

**Studying**
- make sure to review terminology
  - (i.e. what does flow-sensitive mean?)
- do lots of examples

---

**Questions to think about**

Possible big picture questions
- For this class, which of the five criteria for evaluating optimizations have we used and how? (safety, profitability, opportunity, compile-time, implementation complexity)
- What is the scope of different analyses/optimizations that we have studied? (peephole, local, global, interprocedural)
- Where could a representation for loops (polyhedral or presburger sets) fit into the MiniJava compiler?
- How would you design an experiment to compare a set of alias/pointer analysis algorithms?

---

**Structure of the MiniJava Compiler (CodeGenAssem.java)**

**Analysis**
- character stream
- lexical analysis
- tokens ("words")
- syntactic analysis
- AST ("sentences")
- semantic analysis

**Parser.parse**
- BuildSymTable
- CheckTypes
- AST and symbol table (minijava.node/SymTable/)

**Synthesis**
- IR code generation
- IRT Tree/instruction selection
- Mips/Codegen
- Assem
- optimization
- code generation
- MIPS

**Translate**
- Project 4

---

**Topics**

I. Introduction
- Scanning and parsing

II. Compiling for OOP and Garbage Collection

III. Low-Level Optimizations
- Register allocation
- Instruction scheduling

IV. Data-Flow and Control-Flow Analysis and Optimization
- Dataflow analysis
  - Theoretic framework built on lattices
  - Control flow analysis: control-flow graphs, dominators, dominance frontiers, irreducibility
  - Program optimizations: dead-code elimination, constant propagation, CSE, loop-invariant code motion, copy propagation, PRE, induction variable elimination (*)

V. Static Single Assignment (*)
- SSA Form: types of data dependences, how to translate to minimal SSA
- global value numbering
VI. Parallelism and Locality (*)
  – Dependence analysis
  – Loop transformations
    – unimodular transformation framework
    – Kelly and Pugh transformation framework
  – Tiling and Unroll and Jam: when is tiling legal?
  – Fourier Motzkin elimination for code generation
  – Affine partitionings for parallelism

IV. Interprocedural Analysis and Optimization (*)
  – Aliases
    – how do data-flow analysis algorithms use aliasing information?
    – how do we characterize alias analysis algorithms?
  – Interprocedural Analysis
    – how do different levels of context information affect analysis results?

Loop Transformations

Original code

```
  do i = 1,6
    do j = 1,5
      A(i,j) = A(i-1,j+1)+1
    enddo
  enddo
```

Distance vector:  \((1,-1)\)
Which loop can we parallelize and why?
What transformation can enable parallelism?
What is a synchronization-free affine partitioning for this loop?

Synchronization-free parallelism

```
  do i = 1 to N
    do j = 0 to M
      A(i,j) = A(i-1,j)
    enddo
  enddo
```

SSA

```
f = read()
x = 3
y = 5
if (f>6)
  z = 7
else
  z = 0
  x = y - 2
print x+y+z
```
**Induction Variable Elimination**

\[ i = 0 \]
\[ j = 0 \]
\[ k = 0 \]

**loop:**
\[ i = i + 1 \]
\[ j = i \times 4 \]
\[ k = i + 10 \]

if \( i < 10 \) goto **loop**

**exit:**

---

**Alias/Pointer Analysis Example**

```c
main() {
    int *a, b, c, d;
    a = &c;
    b = &d;
    foo(&a, &a);
    foo(&b, &a);
}

void boo(int** x, int** y) {
    *x = *y;
    **x = 3;
}

void foo(int** p, int** q) {
    boo(p, q);
}
```

Which analyses are relevant? FICI, FICS, FSCI, and/or FSCS

How do the analysis results differ based on a call stack size of 0, 1, or 2?

---

**Jump Functions for Figure 12.11 in book**

```c
int id(int p) { return p; }

if (a==1) { x = id(2); y = id(3); }
else { x = id(3); y = id(2); }
z = x*ty;
```

---

**Example (Data dependence analysis)**

**Sample code**

```c
do i = 1, 6
do j = 1, 5
    A(2i,j) = A(i,j-1)
endo
dendo
```

**Dependence**

\(-2j_1 + j_2 = 0, j_1 - j_2 = 1, \text{ solution: YES}\)

**Distance/Direction Vector**

\(-(i_1,j_1) + (d_i, d_j) = (i_2,j_2), d_i = 1, d_j = 1, d = (<,1)\)

**Dependence Relation**

\(-\{i, j\} \rightarrow \{2i, j\} + 1 \mid | i < n \wedge i \leq 3 \wedge 1 \leq j \leq 4 \)
Tiling

Sample code

do i = 1,6
  do j = 1,5
    A(2i,j) = A(i,j-1)
  enddo
enddo

Dependence Relation

\{ [i, j] -> [2i, j + 1] | 1 \leq i \leq 3 , 1 \leq j \leq 4 \}

Tiling Both Loops with tile size 4

\{ [i, j] -> [(j-1)/4, (i-1)/4, i, j] \}