CS 4120
Introduction to Compilers
Ross Tate
Cornell University
Lecture 16: Basic blocks, CFGs, traces

Where we are

abstract syntax tree
syntax-directed translation (IR generation)
intermediate code
syntax-directed translation (flattening)
reordering with traces
canonical intermediate code
instruction selection
abstract assembly code
assembly code

IR lowering

• We lower the IR to a canonical form in which code is a sequence of statements, each containing a single side effect.
• Done by transformations that lift side-effecting statements to the top of the IR tree.
• \( L[s] = s_1...s_n \)
• \( L[e] = s_1...s_n; e' \)
  • Side effects of \( e \) in \( s_i \). Value of \( e \) computed by side-effect-free \( e' \)

Conditional jumps

• IR is now just a linear list of statements with one side effect per statement
• Still contains CJUMP nodes: two-way branches
• Real machines: fall-through branches (e.g. JZ, JNZ)

Simple Solution

• Translate CJUMP into conditional branch followed by unconditional branch

\[
\text{CJUMP}(\text{TEMP}(t1)==\text{TEMP}(t2), t, f) \quad \text{CMP} \ t_1, t_2 \\
JZ \ t \\
\text{JMP} \ f
\]

• JMP is usually gratuitous
• Code can be reordered so jump goes to next statement

Basic blocks

• Unit of reordering is a basic block
• A sequence of statements that is always begun at its start and always exits at the end:
  • starts with a \( \text{LABEL}(n) \) statement (or beginning of all statements)
  • ends with a \( \text{JUMP}, \text{CJUMP} \), or \( \text{RETURN} \) statement, or just before a \( \text{LABEL} \) statement
  • contains no other \( \text{JUMP} \) or \( \text{CJUMP} \) statement
  • contains no interior \( \text{LABEL} \) used as a jump target
  • No point to breaking up a basic block during reordering
### Basic block example

- CJUMP(e, L2, L3)
- LABEL(L1)
- MOVE(TEMP(x), TEMP(y))
- LABEL(L2)
- MOVE(TEMP(x), TEMP(y) + TEMP(z))
- JUMP(NAME(L1))
- LABEL(L3)
- EXP(Call(NAME(f)), TEMP(x))

### Control-flow graph
- Control-flow graph has basic blocks as nodes
- Edges show control flow between basic blocks

### Fixing conditional jumps
- Reorder basic blocks so that (if possible)
  - the “false” direction of two-way jumps goes to the very next block
  - JUMPs go to the next block (are deleted)
- What if not satisfied?
  - For CJUMP add another JUMP immediately after to go to the right basic block
  - How to find such an ordering of the basic blocks?

### Traces
- Idea: order blocks according to a possible trace: a sequence of blocks that might (naively) be executed in sequence, never visiting a block more than once
- Algorithm:
  - pick an unmarked block (begin w/ start block)
  - run a trace until no more unmarked blocks can be visited, marking each block on arrival
  - repeat until no more unmarked blocks

### Example
- Possible traces?

### Arranging by traces
- Can use profiling information, heuristics to choose which branch to follow
Reordered code

Reversing sense of jumps

1. CJUMP(e, L2, [L3])
   - LABEL(L2)
     - MOVE(TEMP(x), TEMP(y) + TEMP(z))
     - JUMP(L1)
   - LABEL(L1)
     - EXP(CALL(NAME(f)), TEMP(x))

2. CJUMP(e, L2, [L3])
   - LABEL(L2)
     - MOVE(TEMP(x), TEMP(y) + TEMP(z))
     - JUMP(L1)
   - LABEL(L1)
     - EXP(CALL(NAME(f)), TEMP(x))

3. CJUMP(NOT(e), L3, [L2])
   - LABEL(L3)
     - EXP(CALL(NAME(f)), TEMP(x))

4. CJUMP(¬e, L3, [L2])
   - LABEL(L3)
     - EXP(CALL(NAME(f)), TEMP(x))