COSE312: Compilers

Lecture 10 — Intermediate Representation

Hakjoo Oh
2023 Spring

Common Intermediate Representations

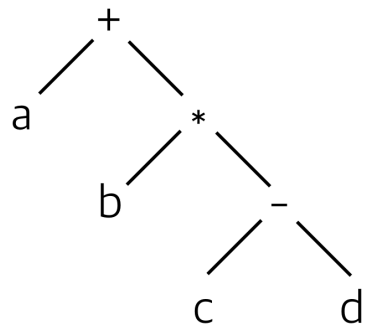
- Three-address code
- Control-flow graph

Three-Address Code

- Instructions with at most one operator on the right side.
- Temporary variables are needed in translation, e.g., $x + y * z$:

$$\begin{aligned} t_1 &= y * z \\ t_2 &= x + t_1 \end{aligned}$$

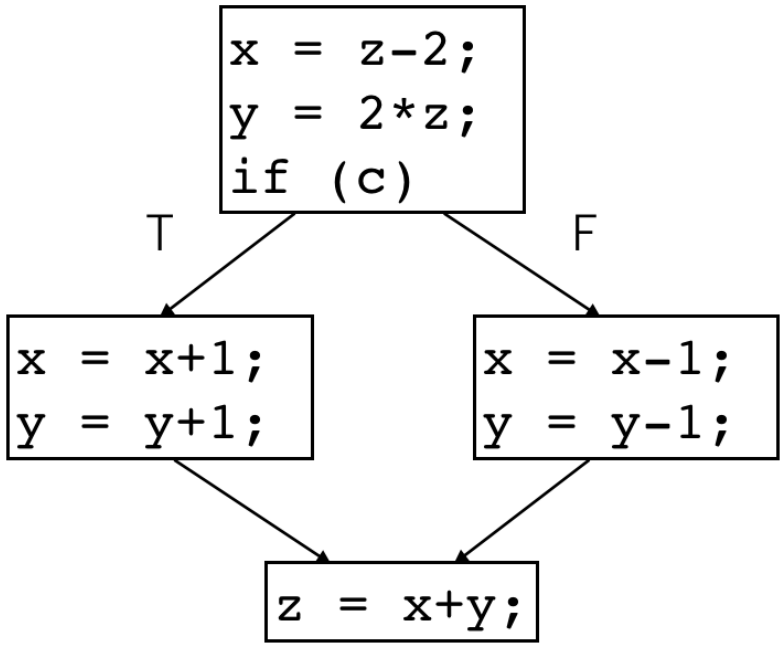
- A linearized representation of a syntax tree, where temporary variables correspond to the internal nodes of the tree: e.g.,



Control-Flow Graph

- Control-Flow Graph (CFG): graph representation of the program
  - A commonly used form for static analysis and optimization
  - Nodes are basic blocks
  - Edges represent control flows

```
x = z-2;
y = 2*z;
if (c) {
  x = x+1;
  y = y+1;
} else {
  x = x-1;
  y = y-1;
}
z = x+y;
```



Basic Blocks

- Maximal sequences of consecutive, branch-free instructions.

```
    x = 1
    y = 1
    z = x + y
L:  t1 = z + 1
    t1 = t1 + 1
    z = t1
    goto L
```

- Properties:
  - Instructions in a basic block are always executed together.
  - No jumps to the middle of a basic block.
  - No jumps out of a basic block, except for the last instruction.

Partitioning Instructions into Basic Blocks

Given a sequence of instructions:

- Determine *leaders*, the first instructions in some basic block.
  1. The first instruction is a leader.
  2. Any instruction that is the target of a conditional or unconditional jump is a leader.
  3. Any instruction that immediately follows a conditional or unconditional jump is a leader.
- For each leader, its basic block consists of itself and all instruction up to but not including the next leader or the end of the program.

Example

```
    i = 1
L1: j = 1
L2: t1 = 10 * i
    t2 = t1 + j
    t3 = 8 * t2
    t4 = t3 - 88
    a[t4] = 0
    j = j + 1
    if j <= 10 goto L2
    i = i + 1
    if i <= 10 goto L1
    i = 1
L3: t5 = i - 1
    t6 = 88 * t5
    a[t6] = 1
    i = i + 1
    if i <= 10 goto L3
```

Control-Flow Graph

A graph representation of intermediate code:

- A directed graph $\boldsymbol{G = (N, \hookrightarrow)}$, where each node $\boldsymbol{n \in N}$ is a basic block and an edge $\boldsymbol{(n_1, n_2) \in (\hookrightarrow)}$ indicates a possible control flow of the program.
- $\boldsymbol{n_1 \hookrightarrow n_2}$ iff
  - there is a conditional or unconditional jump from the end of $\boldsymbol{n_1}$ to the beginning of $\boldsymbol{n_2}$, or
  - $\boldsymbol{n_2}$ immediately follows $\boldsymbol{n_1}$ in the original program, and $\boldsymbol{n_1}$ does not end in an unconditional jump.
- Often, control-flow graphs have unique *entry* and *exit* nodes.

Example

```
    i = 1
L1: j = 1
L2: t1 = 10 * i
    t2 = t1 + j
    t3 = 8 * t2
    t4 = t3 - 88
    a[t4] = 0
    j = j + 1
    if j <= 10 goto L2
    i = i + 1
    if i <= 10 goto L1
    i = 1
L3: t5 = i - 1
    t6 = 88 * t5
    a[t6] = 1
    i = i + 1
    if i <= 10 goto L3
```

CFG Construction for High-level Languages

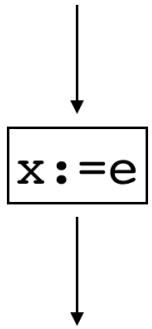
- High-level statements:

$$S \rightarrow x := e \mid S_1; S_2 \mid \mathit{if}\ e\ S_1\ S_2 \mid \mathit{while}\ e\ S$$

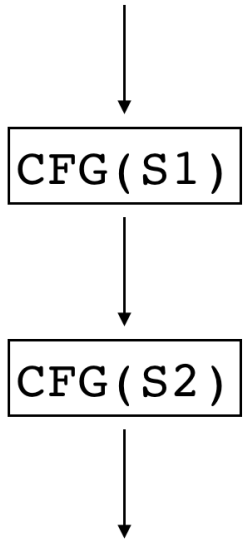
- $\boldsymbol{CFG(S)}$: control-flow graph of $\boldsymbol{S}$
- $\boldsymbol{CFG(S)}$ is recursively defined

CFG Construction for High-level Languages

- $x := e$

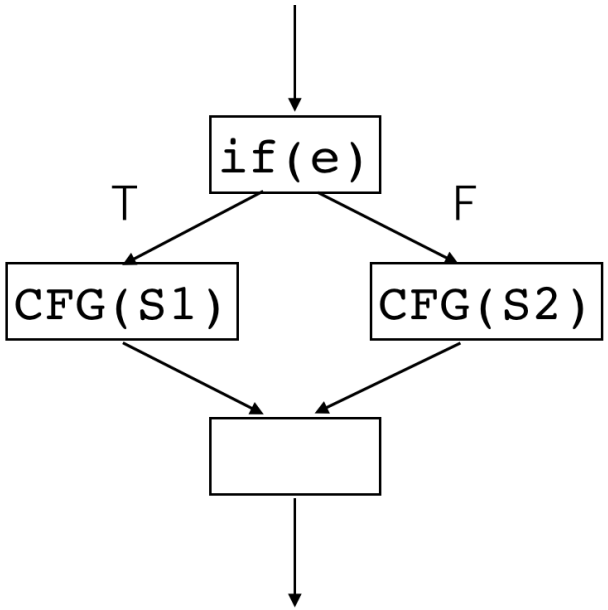


- $S_1; S_2$

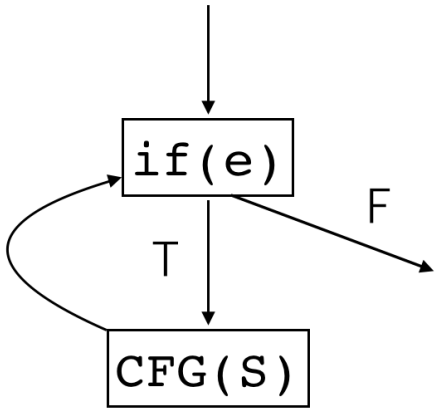


CFG Construction for High-level Languages

- $if\ e\ S_1\ S_2$



- $while\ e\ S$



Example

```
while (c) {
  x = y;
  y = 2;
  if(d) x = y;
  else y = x;
  z = 1;
}
z = x
```

Summary

Intermediate Representations:

- Three-address code
- Control-flow graph