EC3204: Programming Languages and Compilers

Lecture 15 — IR Translation (2): *Control-Flow Graph*

Sunbeom So
Fall 2024

Overview

- `T` program is an IR called **three-address code**.
- **Three-address code**: is an instruction with at most three operands. For example, given $\boldsymbol{x + y * z}$, we produce the three-address code with compiler-generated temporary variables ($\boldsymbol{t_1}$, $\boldsymbol{t_2}$):

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

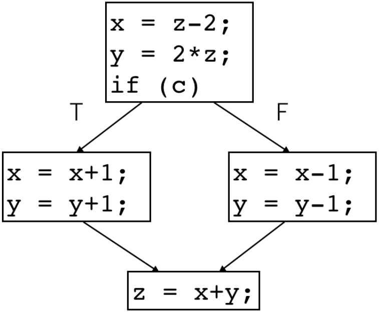
- To perform more code optimizations effectively (global optimization), we transform this IR into another IR, called **Control-Flow Graph (CFG)**.

We will learn the concept of CFG and how to construct it.

Control-Flow Graph

- **Control-Flow Graph (CFG)**: graph representation of the program
    - Common representation for optimization and static analysis
- In this course, we assume
    - Nodes are **basic blocks**, and
    - Edges represent **control flows** between basic blocks

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



```
x = z-2;
y = 2*z;
if (c)
```

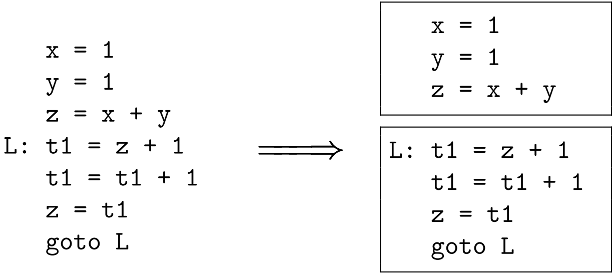
```
x = x+1;
y = y+1;
```

```
x = x-1;
y = y-1;
```

```
z = x+y;
```

Basic Blocks

- Maximal sequences of branch-free instructions that are executed together.
- Properties of basic blocks:
  - No jumps to the middle of a basic block. The control flow can only enter the basic block through its first instruction.
  - No jumps out of a basic block, except for the last instruction.



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

⟹

```
   x = 1
   y = 1
   z = x + y
```

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

Construction of CFG

1. Construct basic blocks by partitioning instructions.
2. Add edges (control-flows) between nodes (basic blocks).

Step 1. Node Construction

Given a sequence of instructions,

1. Determine a **leader**, which will be the first instruction of each basic block.
   - The first instruction is a leader.
   - Any instruction that is the target of a conditional (`if` x `goto` L, `ifFalse` x `goto` L) or unconditional jump (`goto` L) is a leader.
   - Any instruction that immediately follows a conditional or unconditional jump is a leader.
2. For each leader, its basic block consists of itself and all next instructions before the next leader or up to the end of the program.

Example

```
L1  : i = 1
L2  : j = 1
L3  : t1 = 10 * i
L4  : t2 = t1 + j
L5  : t3 = 8 * t2
L6  : t4 = t3 - 88
L7  : a[t4] = 0
L8  : j = j + 1
L9  : if j <= 10 goto L3
L10 : i = i + 1
L11 : if i <= 10 goto L2
L12 : i = 1
L13 : t5 = i - 1
L14 : t6 = 88 * t5
L15 : a[t6] = 1
L16 : i = i + 1
L17 : if i <= 10 goto L13
```

Example

```
L1  : i = 1
L2  : j = 1
L3  : t1 = 10 * i
L4  : t2 = t1 + j
L5  : t3 = 8 * t2
L6  : t4 = t3 - 88
L7  : a[t4] = 0
L8  : j = j + 1
L9  : if j <= 10 goto L3
L10 : i = i + 1
L11 : if i <= 10 goto L2
L12 : i = 1
L13 : t5 = i - 1
L14 : t6 = 88 * t5
L15 : a[t6] = 1
L16 : i = i + 1
L17 : if i <= 10 goto L13
```

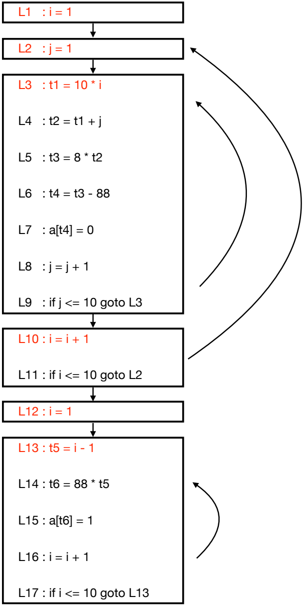
Example

```
L1   : i = 1

L2   : j = 1

L3   : t1 = 10 * i
L4   : t2 = t1 + j
L5   : t3 = 8 * t2
L6   : t4 = t3 - 88
L7   : a[t4] = 0
L8   : j = j + 1
L9   : if j <= 10 goto L3

L10 : i = i + 1
L11 : if i <= 10 goto L2

L12 : i = 1

L13 : t5 = i - 1
L14 : t6 = 88 * t5
L15 : a[t6] = 1
L16 : i = i + 1
L17 : if i <= 10 goto L13
```

Step 2. Edge Construction

- CFG is a directed graph $G = (N, \hookrightarrow)$, where each node $n \in N$ is a basic block and an edge $(n_1, n_2) \in (\hookrightarrow)$ indicates a possible control flow of the program.
- $n_1 \hookrightarrow n_2$ iff
  - there is a conditional or unconditional jump from the end of n_1 to the beginning of n_2, or
  - n_1 does not end in an unconditional jump, and n_2 immediately follows n_1 in the original program.

Example



```
L1  : i = 1

L2  : j = 1

L3  : t1 = 10 * i
L4  : t2 = t1 + j
L5  : t3 = 8 * t2
L6  : t4 = t3 - 88
L7  : a[t4] = 0
L8  : j = j + 1
L9  : if j <= 10 goto L3

L10 : i = i + 1
L11 : if i <= 10 goto L2

L12 : i = 1

L13 : t5 = i - 1
L14 : t6 = 88 * t5
L15 : a[t6] = 1
L16 : i = i + 1
L17 : if i <= 10 goto L13
```

Summary

We use IRs to conduct optimizations or static analyses. In particular, we explored two types of IRs.

- Three-address code: an instruction with at most three operands.
- Control-flow graph: a graph representation of the program.