COSE312: Compilers

Lecture 11 — Translation (3)

Hakjoo Oh
2025 Spring

Common Intermediate Representations

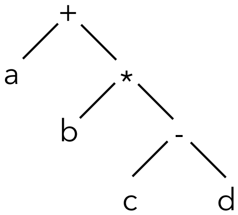
- Three-address code
- Control-flow graph
- Static single assignment form

Three-Address Code

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

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

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

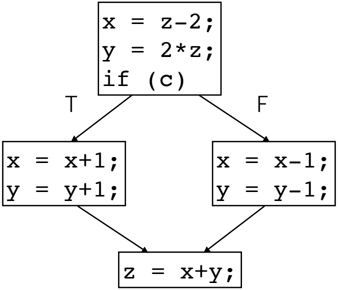


- Suitable for analysis and optimization, e.g., explicit evaluation order

Control-Flow Graph

- Control-Flow Graph (CFG): a graph representation of programs
  - 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
```

Source Language: S

$$
\begin{array}{rcl}
\textit{program} & \to & \textit{block} \\
\textit{block} & \to & \textit{decls}\ \textit{stmts} \\
\textit{decls} & \to & \textit{decls}\ \textit{decl} \mid \epsilon \\
\textit{decl} & \to & \textit{type}\ x \\
\textit{type} & \to & \texttt{int} \mid \texttt{int[}n\texttt{]} \\
\textit{stmts} & \to & \textit{stmts}\ \textit{stmt} \mid \epsilon \\
\\
\textit{stmt} & \to & \textit{lv}\ \texttt{=}\ e \\
& \mid & \texttt{if}\ e\ \textit{stmt}\ \textit{stmt} \\
& \mid & \texttt{while}\ e\ \textit{stmt} \\
& \mid & \texttt{do}\ \textit{stmt}\ \texttt{while}\ e \\
& \mid & \texttt{read}\ x \\
& \mid & \texttt{print}\ e \\
& \mid & \textit{block} \\
\\
\textit{lv} & \to & x \mid x\texttt{[}e\texttt{]} \\
\\
e & \to & n \qquad \text{integer} \\
& \mid & \textit{lv} \qquad \text{l-value} \\
& \mid & e\texttt{+}e \mid e\texttt{-}e \mid e\texttt{*}e \mid e\texttt{/}e \mid \texttt{-}e \qquad \text{airthmetic operation} \\
& \mid & e\texttt{==}e \mid e\texttt{<}e \mid e\texttt{<=}e \mid e\texttt{>}e \mid e\texttt{>=}e \qquad \text{conditional operation} \\
& \mid & \texttt{!}e \mid e\texttt{||}e \mid e\texttt{\&\&}e \qquad \text{boolean operation}
\end{array}
$$

Target Language: G

A directed graph $G = (N, \hookrightarrow)$:

- Each node $n \in N$ contains a command (denoted $cmd(n)$):

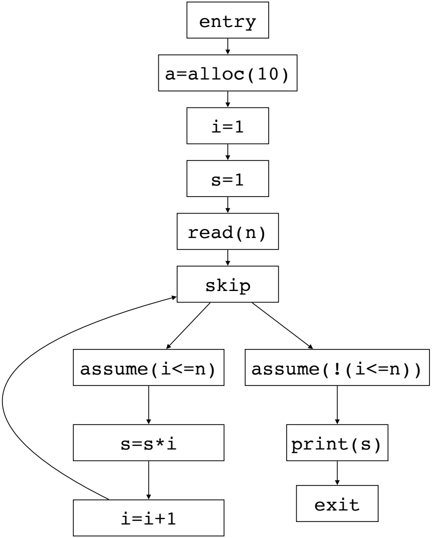
$$c \to x = alloc(n) \mid lv = e \mid assume(e) \mid skip \mid read\ x \mid print\ e$$

- An edge $(n_1, n_2) \in (\hookrightarrow)$ indicates a possible control flow.
- Unique *entry* and *exit* nodes.

Example

```
{
  int n;
  int i;
  int s;
  int[10] a;

  i = 1;
  s = 1;
  read (n);
  while (i <= n) {
    s = s * i;
    i++;
  }
  print (s);
}
```



Semantics of G

- Semantics of command: $M \vdash c \Rightarrow M'$

$$\frac{(l,0),\ldots,(l,n-1) \notin Dom(M)}{M \vdash x = alloc(n) \Rightarrow M[x \mapsto (l,n), (l,0) \mapsto 0, \ldots, (l,n-1) \mapsto 0]}\; n > 0$$

$$\frac{M \vdash lv \Rightarrow l \quad M \vdash e \Rightarrow v}{M \vdash lv = e \Rightarrow M[l \mapsto v]} \qquad \frac{M \vdash e \Rightarrow n}{M \vdash assume(e) \Rightarrow M}\; n \neq 0$$

$$\frac{}{M \vdash skip \Rightarrow M} \qquad \frac{}{M \vdash read\ x \Rightarrow M[x \mapsto n]} \qquad \frac{M \vdash e \Rightarrow n}{M \vdash print\ e \Rightarrow M}$$

- Transition relation $(\rightsquigarrow) \subseteq (N \times Mem) \times (N \times Mem)$:

$$(n_1, M_1) \rightsquigarrow (n_2, M_2) \iff n_1 \hookrightarrow n_2 \wedge M_1 \vdash cmd(n_2) \Rightarrow M_2$$

- Semantic function $\mathcal{G}[\![G]\!] : Mem \to Mem$:

$$\mathcal{G}[\![G]\!](M) = \begin{cases} m' & \text{if } (entry, M) \rightsquigarrow^* (exit, M') \\ \textbf{undef} & \text{otherwise} \end{cases}$$

CFG Construction

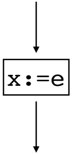
- High-level statements:

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

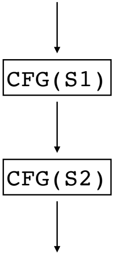
- $CFG(S)$: control-flow graph of S
- $CFG(S)$ is recursively defined
- Assume a node include a single instruction

CFG Construction

- $x := e$

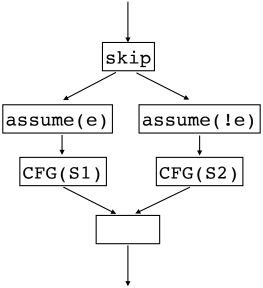


- $S_1; S_2$

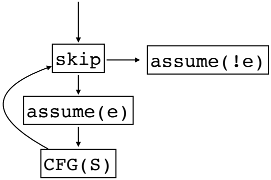


CFG Construction

- $if\ e\ S_1\ S_2$



- $while\ e\ S$



Example

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

Static Single Assignment Form

- An intermediate representation suitable for many code optimizations.
- A program is in SSA iff
  1. each definition has a distinct name, and
  2. each use refers to a single definition.
- Example) Convert the following code into SSA form:

```
p = a + b
q = p - c
p = q * c
p = e - p
q = p + q
```

Static Single Assignment Form

The SSA form of the following:

```
if (flag) x = -1; else x = 1;
y = x * a;
```

needs a ϕ-function:

$$
\begin{array}{l}
\texttt{if (flag) } \texttt{x}_1 \texttt{ = -1; else } \texttt{x}_2 \texttt{ = 1;} \\
\texttt{x}_3 \texttt{ = } \phi(\texttt{x}_1\texttt{, } \texttt{x}_2)\texttt{;} \\
\texttt{y = } \texttt{x}_3 \texttt{ * a;}
\end{array}
$$

Here, $\phi(x_1, x_2)$ has the value x_1 if the control flow passes through the true branch and the value x_2 otherwise.

Exercise

Convert the following code into an SSA form:

```
i = 1
j = 1
k = 0
while (1) {
  if (k < 100) {
    if (j < 20)
      j = i
      k = k + 1
    else
      j = k
      k = k + 2
  }
  else return j
}
```

How to Convert a Program into SSA?

Cytron et al.
Efficiently Computing Static Single Assignment Form and the Control Dependence Graph.
ACM Transactions on Programming Languages and Systems (TOPLAS), Volume 13 Issue 4, Pages 451-490

Summary

Intermediate Representations:

- Three-address code
- Control-flow graph
- Static single assignment form