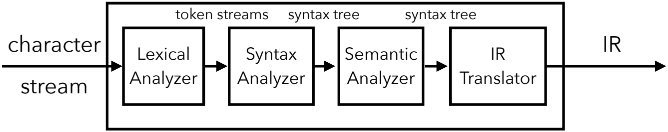
COSE312: Compilers

Lecture 10 — Translation (2)

Hakjoo Oh
2025 Spring

Translation from AST to IR



Why do we use IR?

- The direct translation from AST to the executable is not easy.
- IR is more suitable for analysis and optimization.
- IR reduces the complexity of compiler design: e.g., m source languages and n target languages.

S: The Source Language

- ```
  {
    int x;
    x = 0;
    print (x+1);
  }
  ```
- ```
  {
    int x;
    x = -1;
    if (x) { print (-1); }
    else   { print (2);  }
  }
  ```
- ```
  {
    int x;
    read (x);
    if (x == 1 || x == 2) print (x); else print (x+1);
  }
  ```
  ```

S: The Source Language

- ```
  { int sum;  int i;
    i = 0; sum = 0;
    while (i < 10) {
      sum = sum + i;
      i++;
    }
    print (sum);
  }
  ```
- ```
  {  int[10] arr;  int i;
    i = 0;
    while (i < 10) {
      arr[i] = i;
      i++;
    }
    print (i);
  }
  ```

T: The Intermediate Language

```
{
  int x;
  x = 0;
  print (x+1);
}
```

```
0 : x = 0
0 : t1 = 0
0 : x = t1
0 : t3 = x
0 : t4 = 1
0 : t2 = t3 + t4
0 : write t2
0 : HALT
```

T: The Intermediate Language

```
{
  int x;
  x = -1;

  if (x) {
    print (-1);
  } else {
    print (2);
  }
}
```

```
0 : x = 0
0 : t2 = 1
0 : t1 = -t2
0 : x = t1
0 : t3 = x
0 : if t3 goto 2
0 : goto 3
2 : SKIP
0 : t5 = 1
0 : t4 = -t5
0 : write t4
0 : goto 4
3 : SKIP
0 : t6 = 2
0 : write t6
0 : goto 4
4 : SKIP
0 : HALT
```

T: The Intermediate Language

```
{
  int x;
  read (x);

  if (x == 1 || x == 2)
     print (x);
  else print (x+1);
}
```

```
0 : x = 0
0 : read x
0 : t3 = x
0 : t4 = 1
0 : t2 = t3 == t4
0 : t6 = x
0 : t7 = 2
0 : t5 = t6 == t7
0 : t1 = t2 || t5
0 : if t1 goto 2
0 : goto 3
2 : SKIP
0 : t8 = x
0 : write t8
0 : goto 4
3 : SKIP
0 : t10 = x
0 : t11 = 1
0 : t9 = t10 + t11
0 : write t9
0 : goto 4
4 : SKIP
0 : HALT
```

T: The Intermediate Language

```
{
  int sum;
  int i;

  i = 0;
  sum = 0;
  while (i < 10) {
    sum = sum + i;
    i++;
  }

  print (sum);
}
```

```
0 : sum = 0
0 : i = 0
0 : t1 = 0
0 : i = t1
0 : t2 = 0
0 : sum = t2
2 : SKIP
0 : t4 = i
0 : t5 = 10
0 : t3 = t4 < t5
0 : iffalse t3 goto 3
0 : t7 = sum
0 : t8 = i
0 : t6 = t7 + t8
0 : sum = t6
0 : t10 = i
0 : t11 = 1
0 : t9 = t10 + t11
0 : i = t9
0 : goto 2
3 : SKIP
0 : t12 = sum
0 : write t12
0 : HALT
```

T: The Intermediate Language

```
{
  int[10] arr;
  int i;

  i = 0;
  while (i < 10) {
    arr[i] = i;
    i++;
  }
  print (i);
}
```

```
0 : arr = alloc (10)
0 : i = 0
0 : t1 = 0
0 : i = t1
2 : SKIP
0 : t3 = i
0 : t4 = 10
0 : t2 = t3 < t4
0 : iffalse t2 goto 3
0 : t5 = i
0 : t6 = i
0 : arr[t5] = t6
0 : t8 = i
0 : t9 = 1
0 : t7 = t8 + t9
0 : i = t7
0 : goto 2
3 : SKIP
0 : t10 = i
0 : write t10
0 : HALT
```

Abstract Syntax of S

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

Semantics of S

A statement changes the memory state of the program: e.g.,

```
int i;
int[10] arr;
i = 1;
arr[i] = 2;
```

The memory is a mapping from locations to values:

$$
\begin{array}{rcl}
l \in \boldsymbol{Loc} & = & \boldsymbol{Var} + \boldsymbol{Addr} \times \boldsymbol{Offset} \\
v \in \boldsymbol{Value} & = & \mathbb{N} + \boldsymbol{Addr} \times \boldsymbol{Size} \\
\boldsymbol{Offset} & = & \mathbb{N} \\
\boldsymbol{Size} & = & \mathbb{N} \\
m \in \boldsymbol{Mem} & = & \boldsymbol{Loc} \rightarrow \boldsymbol{Value} \\
a \in \boldsymbol{Addr} & = & \text{Address}
\end{array}
$$

Semantics Rules

$\boxed{M \vdash decl \Rightarrow M'}$

$$\frac{}{M \vdash \texttt{int}\ x \Rightarrow M[x \mapsto 0]}$$

$$\frac{(a,0),\ldots,(a,n-1) \notin Dom(M)}{M \vdash \texttt{int}[n]\ x \Rightarrow M[x \mapsto (a,n), (a,0) \mapsto 0, \ldots, (a,n-1) \mapsto 0]}\ n > 0$$

$\boxed{M \vdash stmt \Rightarrow M'}$

$$\frac{M \vdash lv \Rightarrow l \qquad M \vdash e \Rightarrow v}{M \vdash lv\ \texttt{=}\ e \Rightarrow M[l \mapsto v]}$$

$$\frac{M \vdash e \Rightarrow n \qquad M \vdash stmt_1 \Rightarrow M_1}{M \vdash \texttt{if}\ e\ stmt_1\ stmt_2 \Rightarrow M_1}\ n \neq 0 \qquad \frac{M \vdash e \Rightarrow 0 \qquad M \vdash stmt_2 \Rightarrow M_1}{M \vdash \texttt{if}\ e\ stmt_1\ stmt_2 \Rightarrow M_1}$$

$$\frac{M \vdash e \Rightarrow 0}{M \vdash \texttt{while}\ e\ stmt \Rightarrow M} \qquad \frac{\begin{array}{c} M \vdash e \Rightarrow n \qquad M \vdash stmt \Rightarrow M_1 \\ M_1 \vdash \texttt{while}\ e\ stmt \Rightarrow M_2 \end{array}}{M \vdash \texttt{while}\ e\ stmt \Rightarrow M_2}\ n \neq 0$$

$$\frac{M \vdash stmt \Rightarrow M_1 \qquad M_1 \vdash e \Rightarrow 0}{M \vdash \texttt{do}\ stmt\ \texttt{while}\ e \Rightarrow M_1} \qquad \frac{\begin{array}{c} M \vdash stmt \Rightarrow M_1 \qquad M_1 \vdash e \Rightarrow n \\ M_1 \vdash \texttt{do}\ stmt\ \texttt{while}\ e \Rightarrow M_2 \end{array}}{M \vdash \texttt{do}\ stmt\ \texttt{while}\ e \Rightarrow M_2}\ n \neq 0$$

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

Semantics Rules

$\boxed{M \vdash lv \Rightarrow l}$

$$\frac{}{M \vdash x \Rightarrow x} \qquad \frac{M \vdash e \Rightarrow n_1}{M \vdash x[e] \Rightarrow (a, n_1)} \; M(x) = (a, n_2), 0 \leq n_1 < n_2$$

$\boxed{M \vdash e \Rightarrow v}$

$$\frac{}{M \vdash n \Rightarrow n} \qquad \frac{M \vdash lv \Rightarrow l}{M \vdash lv \Rightarrow M(l)} \qquad \frac{M \vdash e_1 \Rightarrow n_1 \quad M \vdash e_2 \Rightarrow n_2}{M \vdash e_1 \texttt{ + } e_2 \Rightarrow n_1 + n_2}$$

$$\frac{M \vdash e_1 \Rightarrow n_1 \quad M \vdash e_2 \Rightarrow n_2}{M \vdash e_1 \texttt{ / } e_2 \Rightarrow n_1 / n_2} \; n_2 \neq 0 \qquad \frac{M \vdash e \Rightarrow n}{M \vdash \texttt{-}e \Rightarrow -n}$$

$$\frac{M \vdash e_1 \Rightarrow n_1 \quad M \vdash e_2 \Rightarrow n_2}{M \vdash e_1 \texttt{ == } e_2 \Rightarrow 1} \; n_1 = n_2 \qquad \frac{M \vdash e_1 \Rightarrow n_1 \quad M \vdash e_2 \Rightarrow n_2}{M \vdash e_1 \texttt{ == } e_2 \Rightarrow 0} \; n_1 \neq n_2$$

$$\frac{M \vdash e_1 \Rightarrow n_1 \quad M \vdash e_2 \Rightarrow n_2}{M \vdash e_1 \texttt{ > } e_2 \Rightarrow 1} \; n_1 > n_2 \qquad \frac{M \vdash e_1 \Rightarrow n_1 \quad M \vdash e_2 \Rightarrow n_2}{M \vdash e_1 \texttt{ > } e_2 \Rightarrow 0} \; n_1 \leq n_2$$

$$\frac{M \vdash e_1 \Rightarrow n_1 \quad M \vdash e_2 \Rightarrow n_2}{M \vdash e_1 \texttt{ || } e_2 \Rightarrow 1} \; n_1 \neq 0 \vee n_2 \neq 0$$

$$\frac{M \vdash e_1 \Rightarrow n_1 \quad M \vdash e_2 \Rightarrow n_2}{M \vdash e_1 \texttt{ \&\& } e_2 \Rightarrow 1} \; n_1 \neq 0 \wedge n_2 \neq 0$$

$$\frac{M \vdash e \Rightarrow 0}{M \vdash \texttt{!}e \Rightarrow 1} \qquad \frac{M \vdash e \Rightarrow n}{M \vdash \texttt{!}e \Rightarrow 0} \; n \neq 0$$

Runtime Errors in S

Runtime errors = undefined semantics.

- Type errors, e.g.,
  - `int [-10] a;`
  - `int[10] a; int i; i[a] = 0;`
  - `int[10] a; if (a) { ... }`
  - `int i; int[10] a; print(a); print(a+i);`
- Divide-by-zero, e.g.,
  - ```
    int i; i = 10;
    while (i > 0) {
      i = i - 1;
    }
    print(5 / i);
    ```
- Buffer-overrun, e.g.,
  - ```
    int[10] a; int i;
    while (i < 10) {
      i = i + 1;
    }
    a[i] = 0;
    ```

These errors will be detected by a semantic analyzer.

Syntax of T

$$
\begin{array}{rcl}
\mathit{program} & \rightarrow & \mathit{LabeledInstruction}^* \\
\mathit{LabeledInstruction} & \rightarrow & \mathit{Label} \times \mathit{Instruction} \\
\mathit{Instruction} & \rightarrow & \texttt{skip} \\
& | & x = \texttt{alloc}(n) \\
& | & x = y \ \mathit{bop} \ z \\
& | & x = y \ \mathit{bop} \ n \\
& | & x = \mathit{uop} \ y \\
& | & x = y \\
& | & x = n \\
& | & \texttt{goto} \ L \\
& | & \texttt{if} \ x \ \texttt{goto} \ L \\
& | & \texttt{ifFalse} \ x \ \texttt{goto} \ L \\
& | & x = y[i] \\
& | & x[i] = y \\
& | & \texttt{read} \ x \\
& | & \texttt{write} \ x \\
& | & \texttt{halt} \\
\mathit{bop} & \rightarrow & \texttt{+} \mid \texttt{-} \mid \texttt{*} \mid \texttt{/} \mid \texttt{>} \mid \texttt{>=} \mid \texttt{<} \mid \texttt{<=} \mid \texttt{==} \mid \texttt{\&\&} \mid \texttt{||} \\
\mathit{uop} & \rightarrow & \texttt{-} \mid \texttt{!}
\end{array}
$$

Semantics

$$
\begin{array}{rcl}
l \in \boldsymbol{Loc} & = & \boldsymbol{Var} + \boldsymbol{Addr} \times \boldsymbol{Offset} \\
v \in \boldsymbol{Value} & = & \mathbb{N} + \boldsymbol{Addr} \times \boldsymbol{Size} \\
\boldsymbol{Offset} & = & \mathbb{N} \\
\boldsymbol{Size} & = & \mathbb{N} \\
m \in \boldsymbol{Mem} & = & \boldsymbol{Loc} \to \boldsymbol{Value} \\
a \in \boldsymbol{Addr} & = & \text{Address}
\end{array}
$$

$$\frac{}{M \vdash \texttt{skip} \Rightarrow M}$$

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

$$\frac{}{M \vdash x = y \ bop \ z \Rightarrow M[x \mapsto M(y) \ bop \ M(z)]}$$

$$\frac{}{M \vdash x = y \ bop \ n \Rightarrow M[x \mapsto M(y) \ bop \ n]}$$

$$\frac{}{M \vdash x = uop \ y \Rightarrow M[x \mapsto uop \ M(y)]}$$

$$\frac{}{M \vdash x = y \Rightarrow M[x \mapsto M(y)]} \qquad \frac{}{M \vdash x = n \Rightarrow M[x \mapsto n]}$$

$$\frac{}{M \vdash \texttt{goto} \ L \Rightarrow M} \qquad \frac{}{M \vdash \texttt{if} \ x \ \texttt{goto} \ L \Rightarrow M} \qquad \frac{}{M \vdash \texttt{ifFalse} \ x \ \texttt{goto} \ L \Rightarrow M}$$

$$\frac{M(y) = (l,s) \qquad M(i) = n \qquad 0 \leq n \land n < s}{M \vdash x = y[i] \Rightarrow M[x \mapsto M((l,n))]}$$

$$\frac{M(x) = (l,s) \qquad M(i) = n \qquad 0 \leq n \land n < s}{M \vdash x[i] = y \Rightarrow M[(l,n) \mapsto M(y)]}$$

$$\frac{}{M \vdash \texttt{read} \ x \Rightarrow M[x \mapsto n]} \qquad \frac{M(x) = n}{M \vdash \texttt{write} \ x \Rightarrow M}$$

Execution of a T Program

1. Set $instr$ to the first instruction of the program.
2. $M = \lambda x.0$
3. Repeat:
   1. If $instr$ is HALT, terminate the execution.
   2. Update M by M' such that $M \vdash instr \Rightarrow M'$
   3. Update $instr$ by the next instruction.
      - ★ When the current instruction is `goto L`, `if x goto L`, or `ifFalse x goto L`, the next instruction is `L`.
      - ★ Otherwise, the next instruction is what immediately follows.

Translation of Expressions

Examples:

- 2 ⇒ `t = 2`, where `t` holds the value of the expression (label is omitted)
- `x` ⇒ `t = x`
- `x[1]` ⇒ `t1 = 1, t2 = x[t1]`
- `2+3` ⇒ `t1 = 2, t2 = 3, t3 = t1 + t2`
- `-5` ⇒ `t1 = 5, t2 = -t1`
- `(x+1)+y[2]` ⇒ `t1=x, t2=1, t3=t1+t2, t4=2, t5=y[t4], t6=t3+t5`

Translation of Expressions

$$\textbf{trans}_e : e \to \mathit{Var} \times \mathit{LabeledInstruction}^*$$

$$
\begin{array}{rcll}
\textbf{trans}_e(n) & = & (t, [t = n]) & \cdots \text{new t} \\
\textbf{trans}_e(x) & = & (t, [t = x]) & \cdots \text{new t} \\
\textbf{trans}_e(x[e]) & = & \text{let } (t_1, code) = \textbf{trans}_e(e) & \\
 & & \text{in } (t_2, code@[t_2 = x[t_1]]) & \cdots \text{new } t_2 \\
\textbf{trans}_e(e_1 + e_2) & = & \text{let } (t_1, code_1) = \textbf{trans}_e(e_1) & \\
 & & \text{let } (t_2, code_2) = \textbf{trans}_e(e_2) & \\
 & & \text{in } (t_3, code_1@code_2@[t_3 = t_1 + t_2]) & \cdots \text{new } t_3 \\
\textbf{trans}_e(-e) & = & \text{let } (t_1, code_1) = \textbf{trans}_e(e) & \\
 & & \text{in } (t_2, code_1@[t_2 = -t_1]) & \cdots \text{new } t_2
\end{array}
$$

Translation of Statements

Examples:

- `x=1+2` $\Rightarrow$ $\boldsymbol{t_1 = 1; t_2 = 2; x = t_1 + t_2}$
- `x[1]=2` $\Rightarrow$ $\boldsymbol{t_1 = 1; t_2 = 2; x[t_1] = t_2}$
- `if (1) x=1; else x=2;` $\Rightarrow$
- `while (x<10) x++;` $\Rightarrow$

Translation of Statements

$$\textbf{trans}_s : stmt \to LabeledInstruction^*$$

$$
\begin{array}{rcl}
\textbf{trans}_s(x = e) & = & \text{let } (t_1, code_1) = \textbf{trans}_e(e) \\
 & & code_1 @ [x = t_1] \\
\textbf{trans}_s(x[e_1] = e_2) & = & \text{let } (t_1, code_1) = \textbf{trans}_e(e_1) \\
 & & \text{let } (t_2, code_2) = \textbf{trans}_e(e_2) \\
 & & \text{in } code_1 @ code_2 @ [x[t_1] = t_2] \\
\textbf{trans}_s(\texttt{read } x) & = & [\texttt{read } x] \\
\textbf{trans}_s(\texttt{print } e) & = & \text{let } (t_1, code_1) = \textbf{trans}_e(e) \\
 & & \text{in } code_1 @ [\texttt{write } t_1]
\end{array}
$$

Translation of Statements

$$
\begin{array}{l}
\textbf{trans}_s(\texttt{if}\ e\ stmt_1\ stmt_2) = \\
\quad \text{let } (t_1, code_1) = \textbf{trans}_e(e) \\
\quad \text{let } code_t = \textbf{trans}_s(stmt_1) \\
\quad \text{let } code_f = \textbf{trans}_s(stmt_2) \\
\quad \text{in } code_1 @ \qquad\qquad\qquad\qquad \cdots \text{new } l_t, l_f, l_x \\
\qquad [\texttt{if}\ t_1\ \text{goto}\ l_t] @ \\
\qquad [\text{goto}\ l_f] @ \\
\qquad [(l_t, \texttt{skip})] @ \\
\qquad\quad code_t @ \\
\qquad\quad [\text{goto}\ l_x] @ \\
\qquad [(l_f, \texttt{skip})] @ \\
\qquad\quad code_f @ \\
\qquad\quad [\text{goto}\ l_x] @ \\
\qquad [(l_x, \texttt{skip})]
\end{array}
$$

Translation of Statements

$$\textbf{trans}_s(\texttt{while}\ e\ stmt) =$$

$$\begin{array}{ll} \text{let } (t_1, code_1) = \textbf{trans}_e(e) & \\ \text{let } code_b = \textbf{trans}_s(stmt) & \\ \text{in } [(l_e, \texttt{skip})] @ & \cdots \text{new } l_e, l_x \\ \quad code_1 @ & \\ \quad [\texttt{ifFalse}\ t_1\ l_x] @ & \\ \quad code_b @ & \\ \quad [\text{goto } l_e] @ & \\ \quad [(l_x, \texttt{skip})] & \end{array}$$

$$\textbf{trans}_s(\texttt{do}\ stmt\ \texttt{while}\ e) =$$
$$\textbf{trans}_s(stmt) @ \textbf{trans}_s(\texttt{while}\ e\ stmt)$$

Others

Declarations:

$$\begin{array}{rcl} \textbf{trans}_d(\texttt{int}\ x) & = & [x = 0] \\ \textbf{trans}_d(\texttt{int}[n]\ x) & = & [x = \texttt{alloc}(n)] \end{array}$$

Blocks:

$$\begin{array}{l} \textbf{trans}_b(d_1, \ldots, d_n\ s_1, \ldots, s_m) = \\ \quad \textbf{trans}_d(d_1)@\cdots@\textbf{trans}_d(d_n)@\textbf{trans}_s(s_1)@\cdots@\textbf{trans}_s(s_m) \end{array}$$

Summary

- Translation from source language (S) to target language (T).
- Every automatic translation from language S to T is done *recursively* on the structure of the source language S, while preserving some *invariant* during the translation.