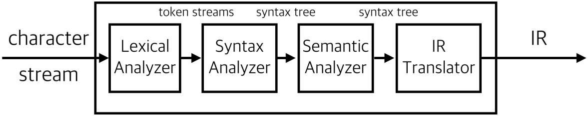
EC3204: Programming Languages and Compilers

Lecture 14 — IR Translation (1): *Automatic Translation*

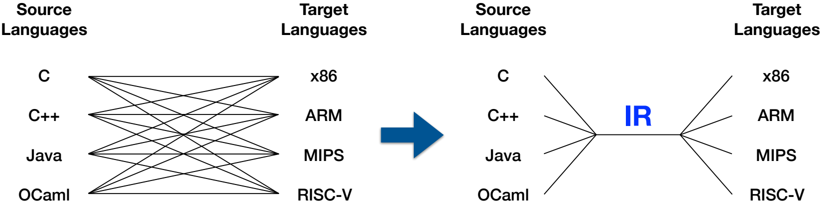
Sunbeom So
Fall 2024

Translation from AST to IR



Why do we use IR?

- IR reduces the complexity of compiler design.



4×4 **compilers**

4 **frontends +** 4 **backends**

Translation Example (1)

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

$\xRightarrow{\text{Translation}}$

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

Translation Example (2)

```
{
  int sum;
  int i;

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

  print (sum);
}
```

$\xRightarrow{\text{Translation}}$

```
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
```

Contents

Goal: define a translation procedure that converts a S program into a semantically equivalent T program.

- Define our source language (S) and target language (T).
- Define an automatic translation procedure from S to T.

Programming Language

A programming language is defined by:

- **Syntax**: a set of rules that define the structure of a program
- **Semantics**: a set of rules that define the meaning of a program execution

The syntax is divided into two kinds:

- **Concrete Syntax**: defines the full structure of a program.
  - Used when writing a program or reading a program.

$$\texttt{if}\ (b)\ \{c_1\}\ \texttt{else}\ \{c_2\}$$

- **Abstract Syntax**: defines the abstract, core structure of a program.
  - Used when automatically generating, analyzing, and optimizing a program.

$$\texttt{if}\ b\ c_1\ c_2$$

Concrete Syntax of S

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

Abstract Syntax of S

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

Semantic Domain of S

A memory state $\boldsymbol{m}$ is a mapping from locations ($\boldsymbol{Loc}$) to values ($\boldsymbol{Value}$).

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

cf) $+$ denotes a disjoint union operator.

Inference Rule

We define program executions using **big-step operational semantics**, where the meanings are specified based on overall execution results. In particular, we define the semantics using **inference rules**. An inference rule is of the form:

$$\frac{\boldsymbol{A}}{\boldsymbol{B}}$$

- Interpreted as: "if A is true then B is also true".
- $\boldsymbol{A}$: hypothesis (antecedent)
- $\boldsymbol{B}$: conclusion (consequent)
- Inference rules without hypotheses are called axioms (e.g., $\boldsymbol{B}$):

$$\overline{\boldsymbol{B}}$$

- The hypothesis may contain multiple statements, e.g.,

$$\frac{\boldsymbol{A} \quad \boldsymbol{B}}{\boldsymbol{C}}$$

  Interpreted as: "If both A and B are true then so is C".

Example: Semantics of S

The execution rules of if-statements:

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

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

- The semantics consists of judgments of the form:

  $$M \vdash stmt \Rightarrow M'$$

  which can be read as "Executing $stmt$ under M results in a new memory M'".
- Similarly, we have the following judgments for $decl$, lv, and e:

  $$M \vdash decl \Rightarrow M', \qquad M \vdash e \Rightarrow v, \qquad M \vdash lv \Rightarrow l$$

Semantics of S

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

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

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

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

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

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

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

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

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

Semantics of S

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

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

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

$$\frac{}{M \vdash n \Rightarrow n} \qquad \frac{M \vdash lv \Rightarrow l}{M \vdash lv \Rightarrow M(l)}$$

$$\frac{M \vdash e_1 \Rightarrow n_1 \quad M \vdash e_2 \Rightarrow n_2}{M \vdash e_1 \texttt{ + } e_2 \Rightarrow n_1 + n_2} \qquad \frac{M \vdash e \Rightarrow n}{M \vdash \texttt{-}e \Rightarrow -n}$$

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

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

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

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

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

Syntax of T

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

Semantic Domain of T

A T program is executed under the following semantic domain (the same with that of S).

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

Semantics of T

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

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

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

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

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

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

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

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

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

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

Execution of a `T` Program

1. Set $\boldsymbol{instr}$ to the first instruction of the program.
2. Set the initial memory state $\boldsymbol{M}$ to the empty mapping, i.e., $\boldsymbol{M} = []$.
3. Repeat:
   1. If $\boldsymbol{instr}$ is HALT, terminate the execution.
   2. Update $\boldsymbol{M}$ by $\boldsymbol{M'}$ such that $\boldsymbol{M} \vdash \boldsymbol{instr} \Rightarrow \boldsymbol{M'}$
   3. Update $\boldsymbol{instr}$ 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

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

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

The first component (e.g., t) in the output is a temporary variable that stores the value of an original expression.

cf) We omit instruction labels if they are not relevant to the discussion.

Examples

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

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

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

Translation of Statements

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

$$
\begin{aligned}
&\textbf{trans}_s(\texttt{do}\ stmt\ \texttt{while}\ e) = \\
&\quad \textbf{trans}_s(stmt)@\textbf{trans}_s(\texttt{while}\ e\ stmt)
\end{aligned}
$$

Declarations and Block (Program)

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:

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

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++;` $\Rightarrow$

Summary

Define a translation procedure that converts a S program into a semantically equivalent T program.

- Defined a source language (S) and a target language (T).
  - syntax (concrete syntax, abstract syntax), semantics
- Defined an automatic translation procedure from S to T.
  - Key principle: every automatic translation from language S to T is done *recursively* on the structure of the source language S.