Intermediate Code Generation

Lecture 9

Exercise
Question?

\[ S \rightarrow \text{repeat } S \text{ until } E \text{ end} \]

- The above grammar defines \textit{repeat-until} loops, where the loop body is executed at least once; we exit loop when its condition is true.

- Add the required action symbols and write the required semantic routines for such loops. Generate three address codes of the given example.

\[
\text{repeat} \\
\quad a := a-1 \\
\quad b := b+1 \\
\text{until } (a-b) \quad \text{end}
\]
Control statements (repeat-until)

\[
S \rightarrow \text{repeat } S \text{ until } E \text{ end}
\]

Input Example:

```
repeat
    a := a-1
    b := b+1
until (a-b)   end
```
Control statements (repeat-until)

\[ S \rightarrow \text{repeat } \#\text{label } S \text{ until } E \text{ end } \]

*conditional jump*: Destination of jump should be saved in SS by \#label.

(to be used when compiler reaches to the end of loop)

\#label: begin

\[
\text{push}(i)
\]

end

Input Example:

- repeat
- Conditional jump
- \( a := a - 1 \)
- \( b := b + 1 \)
- until (a-b) end
Control statements (repeat-until)

\[ S \rightarrow \text{repeat} \ #\text{label} \ S \ 	ext{until} \ E \ #\text{until} \ 	ext{end} \]

At the end of repeat-until, a conditional jump to the start of loop's body (saved by \#label) is generated by \#until. (No need to Back Patching)

\#until: begin
  PB[i] \leftarrow (\text{jpf}, \text{ss(top)}, \text{ss(top-1)}, );
  i \leftarrow i + 1;
  \text{pop(2)}
end

Input Example:

repeat
  a := a-1
  b := b+1
until (a-b) • end
Control statements (repeat-until)

\[ S \rightarrow \text{repeat} \ #\text{label} \ S \ \text{until} \ E \ #\text{until} \ \text{end} \]

Program Block:

<table>
<thead>
<tr>
<th>i</th>
<th>PB[i]</th>
<th>Semantic Actions</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>((-, a, #1, t1))</td>
<td>#sub</td>
</tr>
<tr>
<td>1</td>
<td>((:=, t1, a, ))</td>
<td>#assign</td>
</tr>
<tr>
<td>2</td>
<td>((+, b, #1, t2))</td>
<td>#add</td>
</tr>
<tr>
<td>3</td>
<td>((:=, t2, b, ))</td>
<td>#assign</td>
</tr>
<tr>
<td>4</td>
<td>((-, a, b, t3))</td>
<td>#sub</td>
</tr>
<tr>
<td>5</td>
<td>((jpf, t3, 0, ))</td>
<td>#until</td>
</tr>
</tbody>
</table>

Input Example:

\[
\begin{align*}
\text{repeat} & \quad \text{Conditional jump} \\
a := a-1 & \quad \text{a := a-1} \\
b := b+1 & \quad \text{b := b+1} \\
\text{until} (a-b) & \quad \text{until (a-b)} \\
\text{end} & \quad \text{end}
\end{align*}
\]
The following grammar defines syntax of for loops. Add the required action symbols and write the required semantic routines for such loops. Generate three address codes of the given example.

\[ S \rightarrow \text{for id := } E_1 \text{ to } E_2 \ A \text{ do } S \text{ end} \]
\[ A \rightarrow \text{by } E_3 \]
\[ A \rightarrow \varepsilon \]

Input Example:

\[ \text{for } j := b+c \text{ to } a* b \text{ by } c*a \text{ do} \]
\[ d := d+j \]

end

\[ b+c : \text{ loop variable (j) initial value} \]
\[ a*b : \text{ loop variable (j) limit (constant)} \]
\[ c*d : \text{ loop variable (j) step (constant)} \]
Control Statements (for loop)

\[
S \rightarrow \text{for } \#pid \ #pid \text{ id } := \ E_1 \ \#assign \text{ to } \ E_2 \ \ A \ \text{do } S \ \text{end}
\]

\[
A \rightarrow \text{by } E_3
\]

\[
A \rightarrow \epsilon
\]

**Input Example:**

2 \ #pid put 2 copies of id’s address in SS; one copy is used and popped by \#assign.

The second copy is later (after seeing \(E_2\)) used for comparison with limit of loop’s variable.

\[
\#pid : \text{begin} \ 
p \leftarrow \text{findaddr} \text{(input)}; 
push(p) \ 
\text{end}
\]

\[
\#assign : \text{begin} \ 
PB[i] \leftarrow (:=, \text{ss} \text{(top)}, \text{ss} \text{(top-1)}); 
i \leftarrow i + 1; \text{pop(2)} \ 
\text{end}
\]
Control Statements (for loop)

\[ S \rightarrow \text{for} \ #\text{pid} \ #\text{pid id} := E_1 \ #\text{assign to} \ E_2 \ #\text{save} \ A \ \text{do} \ S \ \text{end} \]
\[ A \rightarrow \text{by} \ E_3 \]
\[ A \rightarrow \varepsilon \]

A place for conditional jump is saved to be later used (by back patching).

\#\text{save}: \text{begin}
    \text{push}(i), \ i \leftarrow i + 1
\text{end}

Input Example:

for \ j := b+c \ \text{to} \ a*b \ \text{by} \ c*a \ \text{do}
    \text{d := d+j}
end
Control Statements (for loop)

\[
S \rightarrow \text{for } \#\text{pid } \#\text{pid id} := E_1 \#\text{assign to } E_2 \#\text{save A do } S \#\text{for end}
\]

\[
A \rightarrow \text{by } E_3
\]

\[
A \rightarrow \varepsilon
\]

**Input Example:**

\[
\text{for } j := b+c \text{ to } a\times b \text{ by } c\times a \text{ do } d := d+j \text{ end}
\]

In the end of loop and by semantic routine \#for:

- Loop’s variable should be increased by step,
- An unconditional jump to the start loop is generated, and
- The place saved by \#save should be filled by a conditional jump
Control Statements (for loop)

Input Example:

\[ S \rightarrow \text{for } \#pid \#pid \text{id} := E_1 \#assign \text{to } E_2 \#save \text{ } A \text{ do } S \#for \text{ end} \]

A \rightarrow \text{by } E_3

A \rightarrow \varepsilon

\#for: begin

PB[i] \leftarrow (+, \text{ss(top)}, \text{ss(top-3)}, \text{ss(top-3)}); i \leftarrow i+1;

PB[i] \leftarrow (\text{jp}, \text{ss(top-1)}-1, , ); i \leftarrow i+1;

PB[\text{ss(top-1)}] \leftarrow (\leq, \text{ss(top-3)}, \text{ss(top-2)}, i, );

\text{Pop}(4)

end

Unconditional jump

Conditional jump

for \text{j} := b+c \text{ to } a*b \text{ by } c*a \text{ do }

\text{d} := d+j

end
Control Statements (for loop)

\[
S \rightarrow \text{for } \#\text{pid } \#\text{pid id} := E_1 \text{ } \#\text{assign to } E_2 \text{ } \#\text{save A} \text{ do } S \#\text{for end}
\]

\[
A \rightarrow \text{by } E_3
\]

\[
A \rightarrow \#\text{step1}
\]

Input Example:

If there is not an explicit step, (\(A \rightarrow \epsilon\) is used), the step should be set to the default value of 1 (by \#\text{step1}).

\#\text{step1}: begin
\[
t \leftarrow \text{gettemp}
\]
\[
PB[i] \leftarrow (:=, \#1, t, )
\]
\[
i \leftarrow i + 1, \text{push}(t)
\]
end

Unconditional jump

conditional jump

for \(j := b + c\) to \(a \times b\) by \(c \times a\) do
\[
d \leftarrow d + j
\]
end
Control Statements (for loop)

Input Example:

\[
S \rightarrow \text{for} \ #\text{pid} \ #\text{pid} \ \text{id} :\ E_1 \ #\text{assign} \ \text{to} \ E_2 \ \text{save} \ A \ \text{do} \ S \ \#\text{for} \ \text{end}
\]

\[
A \rightarrow \text{by} \ E_3
\]

\[
A \rightarrow \#\text{step1}
\]

Semantic Stack:

After \(E_1\): 

\[
\begin{array}{c|c}
\text{pid} & \text{pid} \\
\hline
\text{t1} & \\
\text{j} & \\
\hline
\text{j} & \\
\end{array}
\]

After \#assign:

\[
\begin{array}{c|c}
\text{pid} & \text{pid} \\
\hline
\text{t1} & \\
\text{j} & \\
\hline
\text{j} & \\
\end{array}
\]

Unconditional jump

conditional jump
Control Statements (for loop)

Input Example:

\[
\begin{align*}
S & \rightarrow \text{for} \ #\text{pid} \ #\text{pid id} \ := \ E_1 \ #\text{assign to} \ E_2 \ #\text{save} \ A \ \text{do} \ S \ #\text{for end} \\
A & \rightarrow \text{by} \ E_3 \\
A & \rightarrow \ #\text{step1}
\end{align*}
\]

Semantic Stack:

\[
\begin{array}{c}
\text{After } E_2 \\
\hline
\text{After } \#\text{save}
\end{array}
\]

Unconditional jump

for \( j := b+c \) to \( a*b \) by \( c*a \) do

\( d := d+j \)

end

conditional jump
Control Statements (for loop)

\[
S \rightarrow \text{for } \#\text{pid } \#\text{pid id} := E_1 \#\text{assign to } E_2 \#\text{save } A \text{ do } S \#\text{for end}
\]
\[
A \rightarrow \text{by } E_3
\]
\[
A \rightarrow \#\text{step1}
\]

Input Example:

for j := b+c to a*b by c*a do
d := d+j
end

Unconditional jump

conditional jump

Semantic Stack:

After \(E_3\):

\[
\begin{array}{c}
t3 \\
3 \\
t2 \\
j
\end{array}
\]

After loop Body:

\[
\begin{array}{c}
t3 \\
3 \\
t2 \\
j
\end{array}
\]
Control Statements (for loop)

\[ S \rightarrow \text{for } \#\text{pid } \#\text{pid id := } E_1 \text{ #assign to } E_2 \text{ #save } A \text{ do } S \text{ #for end} \]
\[ A \rightarrow \text{by } E_3 \]
\[ A \rightarrow \#\text{step1} \]

<table>
<thead>
<tr>
<th>i</th>
<th>PB[i]</th>
<th>Semantic Actions</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>(+, b, c, t1)</td>
<td>#add (by E₁)</td>
</tr>
<tr>
<td>1</td>
<td>(:=, t1, j, )</td>
<td>#assign</td>
</tr>
<tr>
<td>2</td>
<td>(*, a, b, t2)</td>
<td>#mult (by E₂)</td>
</tr>
<tr>
<td>3</td>
<td>(&lt;=, j, t2, ?=9)</td>
<td>#for</td>
</tr>
<tr>
<td>5</td>
<td>(*, c, a, t3)</td>
<td>#mult (by E₃)</td>
</tr>
<tr>
<td>6</td>
<td>(+, d, j, t4)</td>
<td>#add (by S)</td>
</tr>
<tr>
<td>7</td>
<td>(:=, t4, d, )</td>
<td>#assign (By S)</td>
</tr>
<tr>
<td>8</td>
<td>(+ , t3, j, j)</td>
<td>#for</td>
</tr>
<tr>
<td>9</td>
<td>(jp, 2, , )</td>
<td>#for</td>
</tr>
</tbody>
</table>

Input Example:

```
for j := b+c to a*b by c*a do
  d := d+j
end
```

Program Block

Unconditional jump

Conditional jump
Control Statements (for loop without step)

$S \rightarrow \text{for } \#\text{pid } \#\text{pid id} := E_1 \ #\text{assign to } E_2 \ #\text{save A do } S \ #\text{for end}$

$A \rightarrow \text{by } E_3$

$A \rightarrow \#\text{step1}$

<table>
<thead>
<tr>
<th>i</th>
<th>PB[i]</th>
<th>Semantic Actions</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>(+, b, c, t1)</td>
<td>#add (by $E_1$)</td>
</tr>
<tr>
<td>1</td>
<td>(:=, t1, j, )</td>
<td>#assign</td>
</tr>
<tr>
<td>2</td>
<td>(*, a, b, t2)</td>
<td>#mult (by $E_2$)</td>
</tr>
<tr>
<td>3</td>
<td>(&lt;=, j, t2, ?=9)</td>
<td>#for</td>
</tr>
<tr>
<td>5</td>
<td>(:=, #1, t3, )</td>
<td>#step1</td>
</tr>
<tr>
<td>6</td>
<td>(+, d, j, t4)</td>
<td>#add (by S)</td>
</tr>
<tr>
<td>7</td>
<td>(:=, t4, d, )</td>
<td>#assign (By S)</td>
</tr>
<tr>
<td>8</td>
<td>(+, t3, j, j)</td>
<td>#for</td>
</tr>
<tr>
<td>9</td>
<td>(jp, 2, , )</td>
<td>#for</td>
</tr>
</tbody>
</table>

Input Example:

for $j := b+c$ to $a*b$ do
\[d := d+j\]
end

Program Block

Unconditional jump

Conditional jump
The following grammar defines syntax of case statements, where at most one of case statements is to be executed.

Can we generate intermediate code for these statements by just using a semantic stack to store the addresses that are required for back-patching?

Example:

```plaintext
case (c * d) of
  a: a := a + 1;
  b: b := b + 2;
  c: c := c + 3;
  otherwise: e := c * d
end
```

```
S  →  case E of L end
L  →  id: S B
B  →  ε | otherwise S | ; is : S B
```
Control Statements (Switch)

\[ S \rightarrow \text{case } E \text{ of } L \text{ end} \]
\[ L \rightarrow \text{id: } S \ B \]
\[ B \rightarrow \epsilon \mid \text{otherwise } S \mid ; \text{ is } : S \ B \]

\[
\text{case } (c \ast d) \text{ of} \\
\quad a: a := a + 1; \\
\quad b: b := b + 2; \\
\quad c: c := c + 3; \\
\quad \text{otherwise: } e := c*d
\end{array}
\]

end

Note: At most one statement is executed.

Difficulty in generating unconditional jumps: Variable number of statements

Solution 1: Link unconditional jumps (Inefficient!)
Control Statements (Switch)

S → case E of L end
L → id: S B
B → ε | otherwise S | ; is : S B

Note: At most one statement is executed.

Difficulty in generating unconditional jumps: Variable number of statements

Solution 2: Jump backward! (Only two jumps are needed for exit)
Control Statements (Switch)

\[ S \rightarrow \text{case } E \text{ of } L \text{ end} \]
\[ L \rightarrow \text{id: } S \ B \]
\[ B \rightarrow \varepsilon \mid \text{otherwise } S \mid ; \text{ is: } S \ B \]

\[
\begin{align*}
\text{case (c * d) of} & \\
\text{(:=, #out t, ,)} & \\
\text{Conditional jump} & \\
\text{a: } a & := a + 1; \\
\text{b: } b & := b + 2; \\
\text{c: } c & := c + 3; \\
\text{otherwise: } e & := c*d \\
\text{end} & \\
\text{#out} & \\
\end{align*}
\]

Note: At most one statement is executed.

Difficulty in generating unconditional jumps: Variable number of statements

Solution 3: Jump indirectly! (Only one jump is needed for exit)
C-minus Switch Statements (in Assignments)

In the absence of a 'break' statement, subsequent case statements, with a satisfied condition, will be executed.

```
switch(x){
    case 0:
        a = 0;
        break;
    case 1:
        a = 1;
        break;
    case 3:
        a = 3;
        break;
    default: a = 4;
}
```

This is different from switches in C

Conditional jump

Unconditional jumps (only for break statements)

We may have duplicate labels