Exercises: Translation

Text adapted from:
Alessandro Artale, Free University of Bolzano

Exercise I

• Consider the SDD in the next slide, where
  – newlabel() generates a fresh symbolic label
  – newtemp() generates a fresh variable name
  – gen() generates strings
  – || is the string concatenation operator
  – code is the attribute with three-address code
Exercise I

• (continued)
  – id.place is the name of the variable associated to the token id
  – E.place is the temporary variable storing the computed value at E
  – num.val is the value associated to the token num
  – relop.op is a comparison operator (<, <=, =, …)

<table>
<thead>
<tr>
<th>Productions</th>
<th>Semantic rules</th>
</tr>
</thead>
<tbody>
<tr>
<td>Prog → S</td>
<td>S.next = newlabel();</td>
</tr>
<tr>
<td>S → S₁ ; S₂</td>
<td>Prog.code = S.code</td>
</tr>
<tr>
<td></td>
<td>S₁.next = newlabel(); S₂.next = S.next;</td>
</tr>
<tr>
<td></td>
<td>S.code = S₁.code</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>S → while Test do { S₁ }</td>
<td>Test.begin = newlabel();</td>
</tr>
<tr>
<td></td>
<td>Test.true = newlabel(); Test.false = S.next;</td>
</tr>
<tr>
<td></td>
<td>S₁.next = Test.begin;</td>
</tr>
<tr>
<td></td>
<td>S.code = gen(Test.begin ‘:’)</td>
</tr>
<tr>
<td></td>
<td>Test.code</td>
</tr>
<tr>
<td></td>
<td>S₁.code</td>
</tr>
</tbody>
</table>
## Exercise I

<table>
<thead>
<tr>
<th>Productions</th>
<th>Semantic rules</th>
</tr>
</thead>
</table>
| $S \rightarrow \text{id} = E$ | $S\.code = E\.code$
| $\text{Test} \rightarrow \text{id}_1 \text{ relop } \text{id}_2$ | $\text{Test}.code = \text{gen}(\text{‘if’} \ \text{id}_1\.place \text{ relop } \text{id}_2\.place \text{ ‘goto’ } \text{Test}.true)$
| $E \rightarrow E_1 + \text{id}$ | $E\.place = \text{newtemp}();$
| $E \rightarrow E_1 - \text{id}$ | $E\.place = \text{newtemp}();$

| $E \rightarrow E_1 + \text{num}$ | $E\.place = \text{newtemp}();$
| $E \rightarrow E_1 - \text{num}$ | $E\.place = \text{newtemp}();$
| $E \rightarrow \text{id}$ | $E\.Place = \text{id}.place; \ E\.code = ‘’$
Exercise I

• Consider the input

  while x < y do {
    y = y - 1; x = x + 1
  }

Exercise I

• Provide the annotated parse tree (without the code attribute) for the input together with the values of the attributes
Exercise I

while Test do { S }

while x < y do { y = y - 1; x = x + 1 }

while x < y do { y = y - 1; x = x + 1 }
Exercise I

• Provide the value of the place and the code attributes at each node of the tree (this is the 3AC translation of the given input produced by the semantic rules)

while x < y do { y = y - 1; x = x + 1 }
Exercise I

\[
\text{code} = \\
\quad t1 = y - 1 \\
\quad y = t1 \\
\]

Exercise I

\[
\text{Prog} \\
\quad \text{while } \text{Test} \text{ do } \\
\quad \quad \{ \text{S} \} \\
\quad \quad \text{S} \\
\quad \quad E - \text{id} \\
\quad \quad \text{id} \\
\quad \quad \text{id} \\
\quad \quad \text{id} \\
\quad \quad y = t1 \\
\quad \quad \text{code} = t1 = y - 1 \\
\]

while \( x < y \) do \{ \( y = y - 1; \) \( x = x + 1 \) \}
Exercise I

code =
\[
t2 = x + 1 \\
x = t2
\]

place = x

code = t2 = x + 1

place = x

code = ''

place = x

val = 1

Exercise I

```
while x < y do { y = y - 1; x = x + 1 }
```
Exercise I

```plaintext
code =
  t1 = y - 1
  y = t1
L4 :
  t2 = x + 1
  x = t2
```

Exercise I

```plaintext
while x < y do { y = y - 1; x = x + 1 }
```
Exercise II

- Consider the SDD in the next slides, where
  - `newlabel()` generates a fresh symbolic label
  - `newtemp()` generates a fresh variable name
  - `gen()` generates strings
  - `||` is the string concatenation operator
  - `code` is the attribute containing three-address code
Exercise II

• (continued)
  – \textit{id}.place is the name of the variable associated to the token \texttt{id}
  – \textit{E}.place is the temporary variable storing the computed value at \textit{E}
  – \texttt{relop}.\texttt{op} is a comparison operator (<, <=, =, …)

<table>
<thead>
<tr>
<th>Productions</th>
<th>Semantic rules</th>
</tr>
</thead>
<tbody>
<tr>
<td>\textit{Prog} \rightarrow \textit{S}</td>
<td>\textit{S}.\textit{next} = \texttt{newlabel}();</td>
</tr>
<tr>
<td>\textit{S} \rightarrow \textit{S}_1 ; \textit{S}_2</td>
<td>\textit{Pro}.\textit{code} = \textit{S}.\textit{code} \parallel \texttt{gen(}\textit{S}.\textit{next} \texttt{‘;’)}</td>
</tr>
<tr>
<td>\textit{S} \rightarrow \textbf{if} \ \textit{Test} \ \textbf{then}{\ \textit{S}_1 }</td>
<td>\textit{S}_1.\textit{next} = \texttt{newlabel}(); \textit{S}_2.\textit{next} = \textit{S}.\textit{next};</td>
</tr>
<tr>
<td>\textit{S} \rightarrow \textbf{id} = \textit{E}</td>
<td>\textit{S}.\textit{code} = \textit{S}_1.\textit{code} \parallel \texttt{gen(}\textit{S}_1.\textit{next} \texttt{‘;’}) \parallel</td>
</tr>
<tr>
<td>\textit{Prog}.\textit{code} = \textit{S}.\textit{code} \parallel \texttt{gen(}\textit{S}.\textit{next} \texttt{‘;’})</td>
<td></td>
</tr>
<tr>
<td>\textit{S}.\textit{code} = \textit{S}_1.\textit{code} \parallel \texttt{gen(}\textit{S}_1.\textit{next} \texttt{‘;’}) \parallel</td>
<td></td>
</tr>
<tr>
<td>\textit{S}.\textit{code} = \textit{E}.\textit{code} \parallel \texttt{gen(}\textit{id}.\textit{place} \texttt{‘=’} \textit{E}.\textit{place})</td>
<td></td>
</tr>
</tbody>
</table>
Exercise II

Productions

<table>
<thead>
<tr>
<th>Production</th>
<th>Semantic rules</th>
</tr>
</thead>
<tbody>
<tr>
<td>$Test \rightarrow \text{id}_1 \ \text{relop} \ \text{id}_2$</td>
<td>( Test.code = \text{gen}('\text{if}' \ \text{id}_1.\text{place} \ \text{relop}.\text{op} \ \text{id}_2.\text{place} '\text{goto}' \ Test.\text{true}) \parallel \text{gen}('\text{goto}' \ Test.\text{false}) )</td>
</tr>
</tbody>
</table>
| $E \rightarrow E_1 + \ \text{id}$ | $E.\text{place} = \text{newtemp}();$  
| | \( E.\text{code} = E_1.\text{code} \parallel \text{gen}(E.\text{place} '=' E_1.\text{place} '++' \ \text{id}.\text{place}) \) |
| $E \rightarrow \ \text{id}$ | $E.\text{Place} = \text{id}.\text{place}; E.\text{code} = ''$ |

Exercise II

- Consider the input

```plaintext
if y > w then {
    y = x + z;
    x = z + v
```
Exercise II

• Provide the annotated parse tree for the input, along with the values of the attributes (with the exclusion of the code attribute)

```
if y > w then { y = x + z }; x = z + v
```
Exercise II

\[
\text{if } y > w \text{ then } \begin{cases} y = x + z; & x = z + v \\ \end{cases}
\]

Exercise II

- Provide the value of the place and the code attributes at each node of the tree (this is the 3AC translation of the given input produced by the semantic rules)
Exercise II

Prog

if Test then { S } id = E id + id id

if y > w then { y = x + z }; x = z + v

code =
if y > w goto L1
goto L2
L1:
t1 = x + z
y = t1

code =
tl = x + z
y = tl

code =
if y > w goto L1
goto L2
L1:
t1 = x + z
y = t1

place = t1

code = t1 = x + z

place = x
code = ""

place = x

place = y

op = >

place = y

place = w

place = t1

place = z

place = x
Exercise II

if y > w then { y = x + z }; x = z + v
Exercise III

- Provide the definition of inherited attribute
- For the rule

\[
S \rightarrow \textbf{if } Test \textbf{ then } \{ S_1 \}
\]

show what are the synthesized attributes and what are the inherited attributes
Exercise III

- An attribute of a nonterminal $A$ is called
  - *Inherited* if it is derived only from operations on attributes of $A$’s parent or siblings in the parsing tree (the semantic rule is associated with a production with $A$ in the right-hand side)
  - *Synthesized* if it is derived only from the attributes of the children of $A$ (the semantic rule is associated with a production with $A$ in the left-hand side)

**Exercise III**

<table>
<thead>
<tr>
<th>Production</th>
<th>Semantic rules</th>
</tr>
</thead>
</table>
| $S \rightarrow \text{if Test then } \{ S_1 \}$ | $\begin{align*}
  \text{Test.true} &= \text{newlabel}(); & \text{Synthesized (convention)} \\
  \text{Test.false} &= S.next; & \text{Inherited} \\
  S_1.next &= S.next; & \text{Inherited} \\
  S_.code &= \text{Test.code} \parallel \text{gen(Test.true ' :')} \parallel \ll S_1_.code & \text{Synthesized}
\end{align*}$ |
Exercise IV

Consider the SDD in the next slide, where

- `newlabel()` generates a fresh symbolic label
- `newtemp()` generates a fresh variable name
- `gen()` generates strings
- `||` is the string concatenation operator
- `code` is the attribute with three-address code

Exercise IV

(continued)

- `id.place` is the name of the variable associated to the token `id`
- `E.place` is the temporary variable storing the computed value at `E`
- `num.val` is the value associated to the token `num`
- `relop.op` is a comparison operator (`<`, `<=`, `=`, …)
## Exercise IV

### Productions

<table>
<thead>
<tr>
<th>Productions</th>
<th>Semantic rules</th>
</tr>
</thead>
<tbody>
<tr>
<td>Prog → S</td>
<td>S.next = newlabel();</td>
</tr>
<tr>
<td>S → S₁ ; S₂</td>
<td>Prog.code = S.code</td>
</tr>
<tr>
<td>S → while Test do { S₁ }</td>
<td>S₁.next = newlabel();</td>
</tr>
<tr>
<td></td>
<td>S₂.next = S.next;</td>
</tr>
<tr>
<td></td>
<td>S.code = S₁.code</td>
</tr>
<tr>
<td></td>
<td>Test.begin = newlabel();</td>
</tr>
<tr>
<td></td>
<td>Test.true = newlabel();</td>
</tr>
<tr>
<td></td>
<td>Test.false = S.next;</td>
</tr>
<tr>
<td></td>
<td>S₁.next = Test.begin;</td>
</tr>
<tr>
<td></td>
<td>S.code = gen(Test.begin ':')</td>
</tr>
</tbody>
</table>

### Semantic rules

<table>
<thead>
<tr>
<th>Productions</th>
<th>Semantic rules</th>
</tr>
</thead>
<tbody>
<tr>
<td>S → id := E</td>
<td>S.code := E.code</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>Test → id₁ relop id₂</td>
<td>Test.code = gen('if' id₁.place relop.op id₂.place 'goto' Test.true)</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>E → E₁ + id</td>
<td>E.place = newtemp();</td>
</tr>
<tr>
<td></td>
<td>E.code = E₁.code</td>
</tr>
<tr>
<td>E → id</td>
<td>E.Place = id.place; E.code = &quot;&quot;</td>
</tr>
</tbody>
</table>
Exercise IV

Given the input:
while x < y do
z := a + b

- Show the annotated parse tree (without the code attribute) for the input together with the values of the attributes
Exercise IV

The three-address code produced by the semantic actions for the given input

Code =
L1 :
if x < y goto L2
goto L0
L2 :
t1 = a + b
Z = t1
goto L1
L0 :