Reading Assignments

• For lecture on 11/14/2013 (Code generation): Dragon book 8.1–8.4 + 8.6 (34p).

Homework Assignment The homework assignment below consists of two problems with a total of 30 points plus an optional 6 bonus points!

1 Control Flow

Consider the new programming language construct

\[ E \rightarrow \text{stabilize } \text{id} = E_1 \text{ from } E_2 \quad \text{(fix)} \]

The semantics of the construct is as follows:

• Set the variable (id) to the value of \( E_2 \).

• Execute the assignment \( \text{id} = E_1 \) repeatedly until the computation stabilizes (so all subsequent executions will give the same result).

• Return the stable value.

Question 1.1 (8 points). Write a fragment of an SDT that translates stabilize constructs to intermediate code (either the intermediate language of the Dragon book or the TIGER intermediate language from the project). Make sure you explain what attributes are used.

Question 1.2 (4 points). Explain how you would type-check the stabilize construct.

Question 1.3 (optional for 6 bonus points). Extend your SDT to type-check the stabilize construct.

2 Array Translation

Consider the SDT in Figure 1 with incremental code attribute generation for expressions with arrays.

That an SDT is “incremental” for the code attribute just means that we make concatenation of the code attribute implicit: we write

\[ E \rightarrow E_1 + E_2 \quad \text{E.addr = new Temp(); gen(E.addr ‘=’ E_1.addr ‘+’ E_2.addr)} \]

instead of

\[ E \rightarrow E_1 + E_2 \quad \text{E.addr = new Temp(); E.code = E_1.code || E_2.code || gen(E.addr ‘=’ E_1.addr ‘+’ E_2.addr)} \]

http://cs.nyu.edu/courses/fall13/CSCI-GA.2130-001/hw7.pdf
<table>
<thead>
<tr>
<th>PRODUCTION</th>
<th>SEMANTIC RULES</th>
</tr>
</thead>
<tbody>
<tr>
<td>$S \rightarrow \text{id} = E$</td>
<td>$\text{gen(top.get(id.lexeme) \text{‘=’} E.addr)}$</td>
</tr>
<tr>
<td>$</td>
<td>L = E$</td>
</tr>
</tbody>
</table>
| $E \rightarrow E_1 + E_2$ | $\text{E.addr} = \text{new Temp();}$  
| | $\text{gen(E.addr \text{‘=’} E_1.addr + E_2.addr)}$ |
| $| -E_1$ | $\text{E.addr} = \text{new Temp();}$  
| | $\text{gen(E.addr \text{‘=’} \text{‘0’ \text{‘+’} E_2.addr)}$ |
| $| (E_1)$ | $\text{E.addr} = E_1.addr;$  
| $| \text{num}$ | $\text{E.addr} = \text{new Temp();}$  
| | $\text{gen(Temp \text{‘=’} \text{num.lexeme})}$ |
| $| \text{id}$ | $\text{E.addr} = \text{top.get(id.lexeme)}$ |
| $| L$ | $\text{E.addr} = \text{new Temp();}$  
| | $\text{gen(E.addr \text{‘=’} L.array.base [L.addr])}$ |
| $L \rightarrow \text{id}[E]$ | $\text{L.array} = \text{top.get(id.lexeme);}$  
| | $\text{L.type} = L.array.type.elem;$  
| | $\text{L.addr} = \text{new Temp();}$  
| | $\text{gen(L.addr \text{‘=’} E.addr \text{‘*’} L.type.width);}$ |
| $| L_1[E]$ | $\text{L.array} = L_1.array;$  
| | $\text{L.type} = L_1.type.elem;$  
| | $t = \text{new Temp();}$  
| | $\text{L.addr} = \text{new Temp();}$  
| | $\text{gen(t \text{‘=’} E.addr \text{‘*’} L.type.width);}$  
| | $\text{gen(L.addr \text{‘=’} L_1.addr \text{‘+’} t);}$ |

Figure 1: Array Translation.

We use the following special attributes:

- *array* attributes have two sub-attributes: *array.base* with the name used to denote the start address of the array, and *array.type* with the element type of the array.

- *type* attributes have three sub-attributes: All have *type.width*, with the size (in bytes) of the type. If the type is an array type, then *type.elem* is the element type of the array and *type.size* is the size.

**Question 2.1** (4 points). Use the SDT to translate the statement $a = (-3)+(a+-2)$.

**Question 2.2** (6 points). Use the SDT to translate the statement $x = a[i][j] + b[a[i]]$.

**Question 2.3** (8 points). Extend the SDT with a new production for array indexing,

$$L \rightarrow \text{id}[E_1, E_2]$$

which denotes the indicated element of the (row-major) two-dimensional array.