Werkcollege Compilerconstructie
Woensdag 2 november 2016

1. (Part of problem 4 at the exam of 15 December, 2014)

   When generating tree-address code for boolean expressions and *flow-of-control* instructions, we can use labels for addresses that Goto-instructions must jump to. During translation, we can pass both the code generated and the labels as attributes of the variables in the grammar.

   Consider the following part of a syntax-directed definition for generating this three-address code:

Production	Semantic Rules
$P \rightarrow S$	$S.next = newlabel()$
	$P.code = S.code \;\; \|\| \;\; label(S.next)$
$S \rightarrow$ **while** $(B)\ S_1$	$begin = newlabel()$
	$B.true = newlabel()$
	$B.false = S.next$
	$S_1.next = begin$
	$S.code = label(begin) \;\; \|\| \;\; B.code \;\; \|\| \;\; label(B.true)$
	$\|\| \;\; S_1.code \;\; \|\| \;\; gen('\texttt{goto}'\ begin)$
$B \rightarrow B_1 \&\& B_2$	$B_1.true = newlabel()$
	$B_1.false = B.false$
	$B_2.true = B.true$
	$B_2.false = B.false$
	$B.code = B_1.code \;\; \|\| \;\; label(B_1.true) \;\; \|\| \;\; B_2.code$
$B_1 \rightarrow$ **id**$_1$ **rel** **id**$_2$	$B_1.code = gen('\texttt{if}'\ \mathbf{id}_1.addr\ \mathbf{rel}.op\ \mathbf{id}_2.addr\ '\texttt{goto}'\ B_1.true)$
	$\|\| \;\; gen('\texttt{goto}'\ B_1.false)$
$S_1 \rightarrow$ **id**$_1$ = **id**$_2$ **binop** **num**	$S_1.code = gen(\mathbf{id}_1.addr = \mathbf{id}_2.addr\ \mathbf{binop}.op\ \mathbf{num}.val)$

   Both P, S and B have an attribute *code*, In addition, S has an attribute *next*, and B has attributes *true* and *false*.

   (a) Draw the parse tree in the above grammar (with start symbol P) for the following 'program':

```
while (x!=y && x<max)
  x = x*2
```

   (b) Use the syntax-directed definition above to annotate the parse tree, i.e., to determine values for the attributes *code*, *next*, *true* and *false* of the variables in the tree (of course only the attributes that each variable has). Use names $L1, L2, \ldots$ for the labels you introduce.

2. (Derived from Exercise 6.7.1(a) from the book)

   Consider the following boolean expression:

   ```
   a==b && (c==d || e==f)
   ```

   (a) Construct the parse tree for the boolean expression.

   (b) Use the translation scheme of Fig. 6.43 to annotate the parse tree. Give the resulting translation of the boolean expression into three-address code.
   You may assume that the address of the first instruction generated is 100.

3. (Extension of Exercise 6.7.1(a) from the book)

   Consider the following 'program':

   ```
   { if (a==b && (c==d || e==f)) x=1; y=x+1; }
   ```

   (a) Construct the parse tree for the program.

   (b) Use the translation scheme of Fig. 6.43 and Fig. 6.46 to annotate the parse tree. Give the resulting translation of the program into three-address code.
   You may assume that the address of the first instruction generated is 100.