CvP - Werkcollege 1

Exercise 1 Consider the following grammar G:

$$
\begin{aligned}
\langle assign \rangle &\to \langle id \rangle = \langle expr \rangle \\
\langle expr \rangle &\to \langle expr \rangle + \langle expr \rangle \mid \langle expr \rangle * \langle expr \rangle \mid (\langle expr \rangle) \mid \langle id \rangle \\
\langle id \rangle &\to A \mid B \mid C
\end{aligned}
$$

(a) List all lexemes, tokens, terminals, and non-terminals used in G.

(b) Give two distinct right-most derivations of "$A = A + B * C$", and conclude the ambiguity of this grammar.

(c) Count the number of sentential forms in each of the derivations in (a).

(d) Rewrite G to make it unambiguous. Argue (no full proof) why your grammar is unambiguous.

Exercise 2 Consider the following grammar G:

$$
\begin{aligned}
\langle assign \rangle &\to \langle id \rangle = \langle expr \rangle \\
\langle expr \rangle &\to \langle expr \rangle + \langle expr \rangle \\
&\mid \langle id \rangle \\
\langle id \rangle &\to A \mid B \mid C \mid D
\end{aligned}
$$

(a) Draw a parse tree of "$A = B + C + D$".

(b) Prove grammar G is ambiguous.

(c) Suppose that + associative. Does this change the ambiguity of the expression "$A = B + C + D$" in G? Explain your answer.

(d) Rewrite G into an unambiguous grammar where + right-associative.

(e) Starting from G, write an attribute grammar, where

- types cannot be mixed in expressions, and
- types of both sides of an assignment match.

Exercise 3 Translate the following rules from EBNF to BNF:

(a) $\langle A \rangle \rightarrow \langle B \rangle [\langle C \rangle]$

(b) $\langle id\_list \rangle \rightarrow \langle id \rangle \{, \langle id \rangle\}$

(c) $\langle T \rangle \rightarrow \langle T \rangle (+ \mid - \mid 9) \langle F \rangle$

Exercise 4 Consider the attribute grammar in Figure 1.

EXAMPLE 3.6 **An Attribute Grammar for Simple Assignment Statements**

1. Syntax rule: <assign> → <var> = <expr>
   Semantic rule: <expr>.expected_type ← <var>.actual_type
2. Syntax rule: <expr> → <var>[2] + <var>[3]
   Semantic rule: <expr>.actual_type ←
   if (<var>[2].actual_type = int) and
   (<var>[3].actual_type = int)
   then int
   else real
   end if
   Predicate: <expr>.actual_type == <expr>.expected_type
3. Syntax rule: <expr> → <var>
   Semantic rule: <expr>.actual_type ← <var>.actual_type
   Predicate: <expr>.actual_type == <expr>.expected_type
4. Syntax rule: <var> → A | B | C
   Semantic rule: <var>.actual_type ← look-up (<var>.string)

The look-up function looks up a given variable name in the symbol table and returns the variable's type.

Figure 1: Attribute grammar.

(a) In this grammar, point out a synthesized attribute, an inherited attribute, and an intrinsic attribute.

(b) Draw the parse tree of the expression "$A = A+B$" and decorate the parse tree (add the flow of attributes to the tree).

(c) Consider the expression "$A = A + B$". What would happen if the actual type of A is int and the actual type of B is real?