Introduction to Language Theory and Compilation Exercises

Session 7: First, Follow, Action tables continued and Semantic Analysis

First, Follow, Action tables continued

Reminders

Action table construction algorithm

```
begin
    $M \leftarrow \times$ ;
    foreach $A \rightarrow \alpha$ do
        foreach $a \in First^1(\alpha)$ do
            $M[A,a] \leftarrow M[A,a] \cup$ Produce($A \rightarrow \alpha$) ;
        if $\varepsilon \in First^1(\alpha)$ then
            foreach $a \in Follow^1(A)$ do
                $M[A,a] \leftarrow M[A,a] \cup$ Produce($A \rightarrow \alpha$) ;
    foreach $a \in T$ do $M[a,a] \leftarrow$ Match ;
    $M[\$, \varepsilon] \leftarrow$ Accept ;
```

Exercise

Ex. 1. Compute the First$^1(A)$ and Follow$^1(A)$ sets for each variable A and then give the action table for the following grammar:

(1)	<S>	$\rightarrow$	<expr> $
(2)	<expr>	$\rightarrow$	$-$ <expr>
(3)	<expr>	$\rightarrow$	(<expr>)
(4)	<expr>	$\rightarrow$	<var> <expr-tail>
(5)	<expr-tail>	$\rightarrow$	$-$ <expr>
(6)	<expr-tail>	$\rightarrow$	ε
(7)	<var>	$\rightarrow$	ID <var-tail>
(8)	<var-tail>	$\rightarrow$	(<expr>)
(9)	<var-tail>	$\rightarrow$	ε

Semantic Analysis

Reminders

The stages of compiling are: Preprocessing, Lexical analysis, Parsing (syntactical analysis), **Semantic analysis**, Code generation, Code optimization and Linking.

The role of *Semantic analysis* is to check if lexical units make sense in their context.

- Construct the table of symbols: add all required information (identifier, type, ...) into the table of symbols.
- Resolve names: association between the labels (lexical unit) and the identifiers (from the table of symbols).
- Check type of expressions: conversion between integer/real, strings, ...
- Check if the constants are well-defined (non-null).

The semantic analysis can be achieved by an attributed grammar which consists of adding attributes[1] to each production rule.

- Objective : *decorate* grammars with *attributes* which will formalize "what" we *generate*. (for instance, compute the value of an arithmetical expression, infer types and handle coercions...)
- This amounts to applying something to the parse tree of the input string.
- We distinguish two kinds of attributes :
  - *Inherited attributes* : the attribute value depends on the attribute values of the node's *ancestors* and *siblings*
  - *Synthesised attributes* : the attribute value depends on the attribute values of the node's *descendants*.
- Attributes have the following shape:

  $$b = f(c_1, c_2, \ldots, c_n)$$

  where b is the new attribute and $c_1 \ldots c_n$ are attributes of the production. f may be a function or a procedure (i.e. returns nothing).

Additionally, one can include *semantic actions* in the right-hand side of the grammar rules, that is, code excerpts of instructions to execute during the parsing when a rule is derived (e.g. evaluations, printing...). In order to handle attributes and actions correctly, one may need to label terminals and non-terminals (for instance a rule written $E \rightarrow E : e1 + E : e2$ stands for a labelling of the first occurrence of E on the right-hand side by $e1$ and by $e2$ for the second).

[1]Also called annotations.

Exercises

Ex. 2. Assuming that the Java grammar contains rules of Figure 1, write the parse tree for the expression $3+4+"."+(1+2*3.0)$ decorated with the type of each node, and precise, for each type attribute, if it synthesised or inherited. Do not forget to respect the precedence of operators, which concerns arithmetic in this context. We assume that the precedence is taken care of by a parser generator, so here you do not need to transform your grammar to take it into account, just to respect it when constructing the tree.

```
public class Exercise3{
   public static void main(String[] args){
   System.out.println(3+4+"."+(1+2*3.0));
   }
}
```

<EXP>	→	<EXP> + <EXP>
<EXP>	→	<EXP> * <EXP>
<EXP>	→	<EXP> - <EXP>
<EXP>	→	<EXP> / <EXP>
<EXP>	→	(<EXP>)
<EXP>	→	ID
<EXP>	→	Bool
<EXP>	→	Real
<EXP>	→	Int
<EXP>	→	String

Figure 1: Some speculative rules about expressions of the Java grammar

Ex. 3. Consider the following attributed grammar:

		Grammar rules	Semantic Rules
<S>	→	<A> <B> <C>	$B.u = S.u \mid A.u = B.v + C.v \mid S.v = A.v$
<A>	→	a	$A.v = 2 * A.u$
<B>	→	b	$B.v = B.u$
<C>	→	c	$C.v = 1$

1. Draw the parse tree for the input *abc*, (the only string for the language), and show the dependency graph for the associated attributes. Describe one correct order for the evaluation of the attributes.
2. Assume that $S.u$ is assigned the value of 3 before starting attribute evaluation. What will be the value of $S.v$ when evaluation has terminated?
3. Consider now the same grammar with slightly different semantic rules :

		Grammar rules	Semantic Rules
<S>	→	<A> <B> <C>	$B.u = S.u \mid A.u = B.v + C.v \mid S.v = A.v \mid C.u = A.v$
<A>	→	a	$A.v = 2 * A.u$
<B>	→	b	$B.v = B.u$
<C>	→	c	$C.v = C.u - 2$

   Can you evaluate $S.v$?

Ex. 4.

1. Rewrite the following grammar in order to account for operator precedence and associativity:

$$\begin{array}{rcl} \text{<E>} & \rightarrow & \text{<E> <op> <E> | (<E>) | int} \\ \text{<op>} & \rightarrow & + \mid - \mid * \mid / \end{array}$$

2. Associate the rules and attributes necessary to compute the value of an expression E. Finally, remove left recursion from the grammar.

Ex. 5. Assuming that the Java grammar contains rules of Figure 1, identify the scope of all variables

- Give the table of symbols (ToS)
- Give the parse tree of each numerical expression
- Annotate the parse trees with changes of the table of symbols

Report any semantic error.

```java
public class Exercise3{
   public static final double PI = 3.141592653589793;
   public static double diameter;
   public static void main(String[] args){
      double perimeter = 50;
      diameter = perimeter / PI;
      final int diameter = Exercise3.diameter;  //15.915494309189533
      System.out.println(((int)(diameter*100))/100.0);
   }
}
```