SAARLAND UNIVERSITY
COMPUTER SCIENCE

Prof. Dr. Reinhard Wilhelm
Jun.-Prof. Dr. Sebastian Hack
Jörg Herter, M.Sc.
Dipl.-Inform. Christoph Mallon

Compiler Construction WS09/10

Exercise Sheet 5

Please hand in the solutions to the theoretical exercises until the beginning of the lecture next Monday 2009-11-30, 10:00. Please write the number of your tutorial group or the name of your tutor on the first sheet of your solution. Solutions submitted later will not be accepted.

Exercise 5.1: Attribute Grammars (Points: 4+2+4)

Let the grammar G_{expr} given by productions

$$\begin{aligned} E &\rightarrow E + T \mid T \\ T &\rightarrow T * id \mid id \end{aligned}$$

describe the concrete syntax of expressions. Further, let the grammar G with productions

$$\begin{aligned} S &\rightarrow DE \\ E &\rightarrow EE \\ E &\rightarrow id \end{aligned}$$

describe the abstract syntax of expressions, where D stands for variable declarations.

For type deduction if necessary use the attributes $E.pre$ which are derived from the definitions of the operands, and $E.post$ which are required by the context. There are the types: inttype, realtype, booltype, and stringtype as well as the operation intplus, realplus, boolplus (or), and stringplus (concatenation), inttimes, realtimes, and booltimes (and). The type of an identifier $id.type$ is given by the environment $E.env$ using the function $identify(id.sym, E.env)$.

1. Expand the grammar G by attribute rules which determine the types of the operators ($E.op$) from the types of the operands. For this use functions $convertible$ and $balance$ which check the convertibility of two types and return the balanced type, respectively.

2. Draw the dependencies of the attribute instances in the derivation tree for the word $id + id * id$.

3. Extent your solution s.t. the type of an expression also depends on the assignment (refer to the updated grammar below). Types of expressions shall propagate to leaves requiring only minimal adjustments.

$$\begin{aligned} S &\rightarrow DA && (1) \\ A &\rightarrow AA && (2) \\ A &\rightarrow id = E && (3) \\ E &\rightarrow EE && (4) \\ E &\rightarrow id && (5) \end{aligned}$$

   where (3) is an assignment.

Exercise 5.2: Static Semantics (Project)

In the next phase of your project, you are to add the analysis of the static semantics. Namely:

- Name analysis, i.e., associate identifiers with declarations. Implement the usual shadowing and hiding rules of Java.
- Type analysis, i.e., associate expressions with types. E.g. '2 < 3' would be associated with the type boolean, '6 * 9' with type int. Also ensure that the program is properly typed. E.g. 'true + 5' is to be rejected.
- Type hierarchy, i.e., construct a tree structure for the declared classes for checking whether fields and methods are hidden and overwritten, respectively. MiniJava does not support method overloading, hence there must only be a single signature for each method name within a class.
- Unreachable statements, i.e., check for unreachable statements without relying on evaluating conditions. E.g. 'return 0; foo(5);'. Further check whether the end of a non-void function can be reached without encountering a return statement.
- Invalid expression statements, i.e., check for statements that are not valid statements in Java (e.g. '42;') as well as illegal assignments (e.g. 'a + b = 47;').

Implement your analyses in such a way that a statement which syntactically can be a print statement only is a print statement if there is no other valid interpretation at the point of its occurance. Also for compatibility to Java and simplification of semantic checking, the reference part of the method invocation expression may be parenthesized[1]. The single parameter of the PrintStatement is of type int.

You do not need to add another switch, just extend the functionality of –ast and –astmin. The switch –tokens should of course stay functional.

Send your solution to the practical exercise to jherter@cs.uni-sb.de until 2009-12-09, 10:00. Please send just one e-mail per project group.

[1] I.e.

$$\begin{aligned} PrintStatement &\rightarrow PrintReference\ .\ println\ (\ Expression\ )\ ; \\ PrintReference &\rightarrow (\ PrintReference\ ) \mid System\ .\ out \end{aligned}$$