CS4201 – Programming Language Design and Implementation

Practical 1: Semantic Analysis

Chris Brown
cmb21@st-andrews.ac.uk

2022

Weighting: 50% of coursework

Deadline: 10th October 2022 at 21:00
Deadlines on MMS are definitive

You are expected to have read and understood all the information in this specification at least a week before the deadline. You must contact the lecturer regarding any queries well in advance of the deadline.

Purpose

This practical will help develop your skills in:

- Writing a semantic analyser for the front-end of a MiniJava compiler.
- Semantic analysis;
- Scope checking;
- Type checking;
- Symbol tables.

Overview and Background

The semantic analysis phase of a compiler connects variable definitions to their uses, checks that each expression has a correct type, and ultimately translates the abstract syntax into a simpler representation suitable for generating machine code. This practical focusses on the scope checking and type-checking phases.

Type-checking MiniJava should proceed in two phases. First, we must build a symbol table, and then we can type-check the statements and expressions. During the second phase, the symbol table is consulted for each identifier that is found. Note that MiniJava declarations (e.g. classes) are mutually recursive. The first phase can be implemented by a visitor/tree walk that visits the nodes of the MiniJava abstract syntax tree and builds a symbol table of properly declared variables. Note the first phase should also detect scoping errors, i.e. if a variable is used where it is not bound, or if a variable is declared more than once, an error message should be returned.

The second phase of the type-checker can be implemented by a visitor that type-checks all statements and expressions. The symbol table will be updated with the appropriate type for

each object and identifier in the program, otherwise an error message detailing the type error will be reported.

When the type-checker detects a type error or an undeclared identifier, it should print an error message and continue, so that the programmer is told about all of the errors in the program. To recover an error, it is sometimes necessary to build data structures if a valid expression has been encountered. For example, type-checking:

```
{int i = new C();

 int j = i + i;

 …

}
```

Although the expression `new C()` is not an integer, it is still required to store a record of `i` in the symbol table as having type `int`, so that the rest of the program can be type-checked.

If the type-checking phase detects errors, then the compiler should not produce a compiled program as a result. This means that the later phases of the compiler (IR, register allocation, etc.) will not be triggered.

The full BNF grammar for MiniJava is supplied on the Cambridge website as a supplement to Appel's book:

https://www.cambridge.org/resources/052182060X/MCIIJ2e/grammar.htm

Tasks

Your task is to implement a type checker and scope analyser for the MiniJava language that constructs a valid symbol table of identifiers, scopes and types. You may use the ANTLR parser that is available on studres for your solution.

https://studres.cs.st-andrews.ac.uk/CS4201/Coursework/MiniJava%20-%20ANTLR/

This requires antlr-4.7.2-complete.jar to be installed, available from

https://www.antlr.org/download/index.html

https://github.com/antlr/website-antlr4/tree/gh-pages/download

```
javac *.java
java minijava.MiniJavaMain "Path\To\Example.java"
```

Your solution should produce error messages about mismatching types or undeclared identifiers.

You must implement your solution in Java (no other languages will be permitted).

There are a small number of example MiniJava programs available as a starter to get you going, which you can freely modify to show different types of semantic errors. You are also expected to produce your own MiniJava programs/examples to aid your testing in order to give strong confidence that your semantic analysis picks up all possible errors.

In the success case, the solution should simply print out the symbol table, as a list of identifiers, types and some indication of which scope they belong to.

https://studres.cs.st-andrews.ac.uk/CS4201/Coursework/minijava_examples/

Submission

You should hand in the sources of your implementations, together with any recipes needed to build them. In your report, briefly discuss key design decisions, any problems or unexpected features you encountered, and the checking results of your implementations. Make a **zip archive** of all of the above and submit via MMS by the deadline.

Marking

I am looking for:

- Good design and understandable code, which is well documented, with major design decisions explained;
- A description of the coverage of your scope analysis and type-checking, with examples and explanations of the behaviours and errors. You are encouraged to come up with MiniJava programs that show interesting behaviours of the system and document them in the report.
- A report giving a critical analysis of the design and applicability of the scope analyser and type-checker to a range of examples.

The standard mark descriptors in the School Student Handbook will apply:

https://info.cs.st-andrews.ac.uk/student-handbook/learning-teaching/feedback.html#Mark_Descriptors

In particular, a very basic implementation of a semantic analyser for one example with very basic scope analysis and type-checking would fit the definition of a reasonable attempt achieving some of the required functionality and could get marks up to 10. More than that would need work on analysis and critical thinking of the algorithms. As usual, marks of 19 or 20 would need an exceptional solution with demonstrated insight into the problem.

Lateness

The standard penalty for late submission applies (Scheme B: 1 mark per 8 hour period, or part thereof):

https://info.cs.st-andrews.ac.uk/student-handbook/learning-teaching/assessment.html#lateness-penalties

Good Academic Practice

The University policy on Good Academic Practice applies:
https://www.st-andrews.ac.uk/students/rules/academicpractice/