DEPARTMENT OF COMPUTER SCIENCE, TECHNISCHE UNIVERSITÄT KAISERSLAUTERN
COMPILERS AND LANGUAGE PROCESSING TOOLS - SS17

Type and Name Analysis for Minijava

Exercise 3

Group 03
Daniele Gadler, Gopal Praveen
Alka Scaria, Stephen Banin Payin

May 28, 2017

OVERVIEW

In the present report, we describe the type and name analyzer built by our group for Minijava. More specifically, in the 'Features' section we explain what our analyzer is capable of doing, whereas in the 'Technical Description', we provide a deeper description of key components of the Analyzer.
The constructed type and name analyzer is thoroughly commented with Javadoc and passes all tests provided for the 'Name' analysis and passes most of the tests for the 'Type' analysis . Due to time constraints, the group members did not manage to make the analyzer pass all test cases.

FEATURES

- **Class Extension checking**: The name analyzer checks classes for valid extension of classes (i.e: it ensures classes being extended are declared) and ensures the non-existence of circular references while extending other classes. (e.g: if class A extends class B and class B extends class A, a circular reference error is raised).
- **Uniqueness of Class names, Methods' names, Fields' names and Parameter names in methods**
- **Type representation and subtyping checking**: Types are represented in a symbol table. We also wrote a method 'SubTypeOff' that checks if a type is a subtype of another type [1].
- **Method overriding**: Methods with the same names as methods defined in the parent are checked for overriding, ensuring the signatures correspond to one another.
- **Type checking and full name analysis**: We implemented type and name checking for local variables, global variables and fields for the following types: int, boolean, int[]. We also implemented type checking for the following operations: if, while, System.out.println, binary and unary assignment expressions, return statements, method calls and method

declarations and class instantiation.
With respect to the next phase, we provide analysis information concerning types through a stack.

Technical Description

Task 1 - Class Hierarchy and Uniqueness of Elements

We made use of a linked list data structure for checking if circular references exist among classes. Instead, for ensuring the uniqueness of classes, methods, fields and methods, we opted for a *Hashmap*, as it allows for the storage of a <Key, Value> pair.

Task 2 - Types representation and Subtyping checking

We represented types with a **symbol table**, which contains three hash maps: hash_main for the main class, hash_class for non-main classes and hash_method for all methods inside non-main classes [2].

- **hash_class**: Upon entering a class scope', the class' fields, and methods' declarations are stored into this data structure. Upon leaving the class' scope, the hash_class is cleared.
- **hash_method**: Upon entering a method's scope, the method's body(block) inside it are stored. Upon leaving the method's scope, the hash_method is cleared.
- **hash_main**: Same as hash_class, but for the class containing the main method and contains all names of declared classes and extended classes.

Task 3 - Method overriding

We ensured that the return type and all parameters' types of a method overriding a parent method are of the same type or a subtype of the parent's by making use of the method 'SubTypeOff' writting in Task 2 for types' subtype checking. We also checked that the amount of parameters in methods' overridden matches their parent's amount.

Task 4 - Name and Type Analysis

Type checking was implemented in the class 'typechecker', where we check for the type rules provided in the assignment's description. Analogously to exercise 2, we made use of matchers for identifying the proper cases to check in statements (e.g: MJReturn or MJWhile).
Name analysis was implemented in the symbol table: when a new identifier (either as a local variable or as a field) occurs, the hash corresponding to the scope (hash_class, hash_method or hash_main) is checked for that identifier, raising appropriate errors in case the latter is not defined.

References

[1] We would like to thank our study colleague Joseff for support with subtyping checking.

[2] Reference for Symbol Table Creation with hash maps. Accessed on 25th May 2017. "http://alumni.cs.ucr.edu/ vladimir/cs152/assignments.html#A5 "