CMSC 22600 **Compilers for Computer Languages** **Project 2**
Autumn 2016 **October 20, 2016**

SOOL typechecker

Due: November 4, 2016

1 Introduction

The second project is to implement a type checker for SOOL, which checks whether or not a parse tree is *statically correct* and produces a *typed abstract syntax tree* (AST). The abstract syntax tree includes information about the binding sites of identifiers and about the types of variables and expressions. The project seed code will provide an ML-ULex based scanner, an ML-Antlr based parser (but you may also use your scanner and parser from Project 1), and modules for implementing the abstract-syntax-tree representation. A companion document (*The* SOOL *Type System*) describes the formal SOOL type system; the goal of Project 2 is to implement a checker for that formal system.[1]

2 Syntactic restrictions

There are a number of syntactic restrictions that should be enforced by the type checker. Some of these are properties that could have been specified as part of the grammar in Project 1, but would have made the grammar much more verbose. Others are properties that could be specified as part the typing rules below, but it is easier to specify them separately.

- Class and interface names in a program must be distinct.
- A declaration of subclass class `C` that is derived from a class `B`

  ```
  class C (···) extends B (···) { ··· }
  ```

  cannot precede the declaration of `B`. This restriction ensures that the class hierarchy will be acyclic. Also note that class `B` is allowed to mention class `C` and its methods.
- The parameter names of a class and of a member-function definition must be distinct.
- The member-variable names of a class must be distinct.
- The member-function names of a class or interface definition must be distinct.

[1] Remember, a *specification* is a description of a property (yes/no question; true/false statement). It does not define (though it may suggest) an *implementation* for deciding whether or not the property holds. A significant component of this project is to develop the skills needed to produce an implementation from a specification.

- Integer literals are restricted to the range $-2^{62}..2^{62}-1$ (*i.e.*, representable as a 63-bit 2's-complement integer).

Your type checker is responsible for checking these properties and reporting an error when they are violated.

3 Implementation strategy

Two passes over declarations:

1. first records class and interface names; method and variable signatures; and checks that superclasses are declared before subclasses.
2. second checks that variable and method declarations are well typed and builds the program's AST

4 Document history

October 20, 2016 Original version.