SWEN430 - Compiler Engineering (2018)

Lecture 6 - Typing I
AST Building, Context Checking

Lindsay Groves & David J. Pearce

School of Engineering and Computer Science
Victoria University of Wellington

Building an Abstract Syntax Tree (AST)

- AST contains the semantically significant parts of a program needed for compiling/interpreting, and for further checking.

  Omit punction, delimiters etc. which are needed to make programs parsable.

- Easy to extend a recursive descent parser to build an AST.
- Each method called builds and returns an AST for the thing it parses.
- The method builds its AST from the ASTs for its components.
- So the whole tree is built bottom up.

Abstract Syntax Tree (AST)

Java example:

```
public class Test {
 private final int f;

 int method(int final x) {
  return 103;
}}
```

What would you expect the AST to look like?

$\Longrightarrow$ See `ast` classes for While compiler.

Error Checking

- The parser only checks context free syntax.

  (Look at where/how the While parser reports errors.)

- Lots of inputs it accepts are not valid programs.
- Need extra checks to eliminate them:
  - Context conditions: Constraints on where symbols can be used.
  - Type conditions: Ensuring operations are applied to arguments of the correct type.
- Compiler may also check for some forms of logical errors;

  e.g. unreachable code, accessing uninitialised variables, deferencing null pointers, dangling pointers, race conditions, array bound errors, ...

Syntax versus Semantics

- An incorrect input file may still parse
- Java example:

```
class Test {
 void method(int[] xs, Test x) {
  for(int i=0;i!=xs.length;i=i+1) {
   System.out.println(x);
  }
  x.someMethod();
 }
}
```

- The above will be successfully parsed into an AST
- An error will be produced during a later compilation phase
- Is this a syntactic or semantic error?

Context Checking

- Variables, types and methods must be declared
- ... and can only be declared once in a given scope
- Methods must be called with the right number of arguments
- Break and continue can only appear within a loop (or switch)
- Switch case labels must be distinct

Ex: What context conditions (should) apply to While programs?

Ex: Where should they be checked??

Type Checking in While

- Condition in while and if must be Boolean.
- Operands of +, - , * and / must be numeric.
- Operands on && and || must be Boolean.
- In $e_1[e_2]$, e_1 must be an array and e_2 an interger.
- In $e.f$, e must be a record with field f.

Ex: What others?

Easy to check these: Just traverse the AST performing required checks.

$\Longrightarrow$ See While type checker.

Java Type Checking

The Java compiler must check:

- That primitive types are used correctly
- That reference types are used correctly
- That methods and fields exist with appropriate types
- That method overriding respects modifiers
- That generic types are used correctly
- That wildcard types are used correctly
- ...

```
class Test implements Inter <? extends Comparable> {
 double f(float f) {
  int x = (int) f;
  return f;
 }
 void g(Test x, Inter <? extends Comparable> y) {
  y = x; // up cast
  x = (Test) y; // down cast
 }
}
```