CS 444 - Compiler Construction **Winter 2020**

Lecture 13: February 24th , 2020

Lecturer: Ondřej Lhoták *Notes By: Harsh Mistry*

13.1 Context-sensitive analysis continued

13.1.1 Name Resolution

- In Java : A2 and A3
  1. Build global environment
  2. Resolve type names
  3. Build/check class hierarchy (methods/files)
  4. Disambiguate ambiguous namespace
     - In Java you can't simply determine the namespace based on location of usage
  5. Resolve "expressions" (Variables, static fields)
  6. Type checking
  7. Resolve methods instance (non-static) fields

- (5) Resolving expressions (Variables)
  - General: search outwards in nested scopes
  - Shadowing

```
{
  int x;
  {
    int y;
  }
  {
    int x; // Not allowed
    int y; //Allowed
  }
}
```

  - Solution
    - New scope/environment for each block
    - when inserting variable into environment, check outwards for duplicates
    - Variables may shadow fields
  - Problem

```
{
  int x;
  x = 1;      //Can not access y or z ye
  int y;
  y  = 2;
  int z;
  z = 3;
}
```

- A Solution: Start a new block for every declaration

```
{
  int x;
  x = 1;      //Can not access y or z
   { int y;
  y  = 2;
  { int z;
  z = 3;
}}}
```

- (6) Type Checking - Will be covered after step 7
- (7) Resolve method instances (Non-static) fields
  - After type checking, every sub expression has a type
  - **Example 13.1** *Usages*
    - *a.b //look for a field named b in a*
    - *a.b(c) //Look for method named b in contain(type of a). Use type of c to select among overloaded methods*
- (6) Type Checking
  - **Definition 13.2** *Type is a collection of values or an interpretation of a sequence of bits (Reference: Luca Cardell: Type Systems)*
  - **Definition 13.3** *Static Type of an expression E is a set containing all possible values of E*
  - **Definition 13.4** *Dynamic Type of a variable is an assertion that the variable will only contain values of that type*
  - **Definition 13.5** *Type Checking: Enforce declared type assertions*
  - **Definition 13.6** *Static Type Checking: Prove that every expression evaluates to a value in its type*
  - **Definition 13.7** *Dynamic Type Checking: Runtime check that the tag of a value is in the declared type of the variable to which the value is assigned*
  - **Definition 13.8** *A program is type correct if type assertions hold in all executions*
  - **Definition 13.9** *A program is statically type correct if the program satisfies a system of type inference rules (type system)*
  - **Definition 13.10** *A type system is sound if statically typed correct* $\implies$ *type correct*