Script **generated by TTT**

Title: Petter: Compiler Construction (25.06.2020) - 42: Introduction to Declaration Use Analysis

Date: Thu Jun 25 10:02:25 CEST 2020

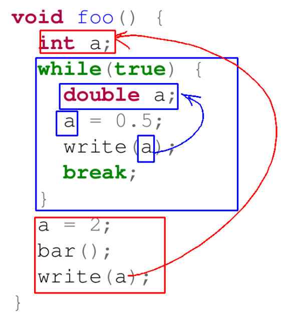
Duration: 05:21 min

Pages: 7

Chapter 2: Decl-Use Analysis

Symbol Bindings and Visibility

Consider the following Java code:



```
void foo() {
  int a;
  while(true) {
    double a;
    a = 0.5;
    write(a);
    break;
  }
  a = 2;
  bar();
  write(a);
}
```

- each *declaration* of a variable v causes memory allocation for v
- using v requires knowledge about its memory location
  $\rightarrow$ determine the declaration v is *bound* to
- a binding is not *visible* when a local declaration of the same name is in scope

  in the example the declaration of a is shadowed by the *local declaration* in the loop body

Scope of Identifiers

```
void foo() {
  int a;
  while (true) {
    double a;
    a = 0.5;
    write(a);
    break;
  }
  a = 2;
  bar();
  write(a);
}
```

scope of **int** a

Scope of Identifiers

```
void foo() {
  int a;
  while (true) {
    double a;
    a = 0.5;
    write(a);
    break;
  }
  a = 2;
  bar();
  write(a);
}
```

scope of **double** a

Scope of Identifiers

```
void foo() {
  int a;
  while (true) {
    double a;
    a = 0.5;
    write(a);
    break;
  }
  a = 2;
  bar();
  write(a);
}
```

scope of **double** a

⚠ administration of identifiers can be quite complicated...

Resolving Identifiers

Observation: each identifier in the AST must be translated into a memory access

Resolving Identifiers

Observation: each identifier in the AST must be translated into a memory access

Problem: for each identifier, find out what memory needs to be accessed by providing *rapid* access to its *declaration*