Comp 311
Principles of Programming Languages
Lecture 4
The Scope of Variables

Corky Cartwright

August 30, 2010

Variables

- What is a variable?
  A legal symbol without a pre-defined (reserved) meaning that can be bound to a value (and perhaps rebound to a different value) during program execution.
  - Examples in Scheme/Java

    ```
    x   y   z
    ```

  - Non-examples in Java

    ```
    +  null true false 7f
    ```

  - Complication in Java: variables vs. fields
- What happens when the same name is used for more than one variable?
  - Example in Scheme:

    ```
    (lambda (x) (x (lambda (x) x)))
    ```

  We use *scoping* rules to distinguish them.

Some scoping examples

- Java:

```
class Foo {
  static void doNothing() {
    int[] a = ...;
    for int i = 0; i < a.length; i++) { ... }
    ...
//   <is a in scope here?  is i in scope here?>
    ...
  }
}
```

What is the scope (part of the program where it can be accessed/referenced) of a?

What is the scope of i?

Formalizing Scope

- Focus on language the pedagogic functional language LC. LC (based on the Lambda Calculus) is the language generated by the root symbol Exp in the following grammar

  ```
  Exp ::= Num | Var | (Exp Exp) | (lambda Var Exp)
  ```

  where **`Var`** is the set of alphanumeric identifiers excluding **`lambda`** and is the set of integers written in conventional decimal radix notation. (LC is very *restrictive*; there are no operators on integers. Later in the course, we will slightly expand it.)

- If we interpret LC as a sub-language of Scheme, it contains only one binding construct: lambda-expressions. In

  ```
  (lambda (a-variable) an-expression)
  ```

  `a-variable` is introduced as a new, unique variable whose scope is the body **`an-expression`** of the lambda-expression (with the exception of possible "holes", which we describe in a moment).

Free and Bound Occurrences

- An important building block in characterizing the scope of variables is defining when a variable *x* ***occurs free in*** an expresssion. For LC, this notion is easy to define inductively.
- Definition (Free occurrence of a variable in LC):
  Let ***x***, ***y*** range over the elements of Var. Let ***M***, ***N*** range over the elements of Exp. Then ***x occurs free in***:
  - ***y*** if ***x*** = *y*;
  - **`(lambda (y) M)`** if ***x*** != ***y*** and ***x*** occurs free in
  - **`(M N)`** if it occurs free either in ***M*** or in ***N***.

  The relation ***x occurs free in*** *y* is the least relation on LC expressions satisfying the preceding constraints.
- It is straightforward but tedious to define when a particular *occurrence* of a variable ***x*** (identified by a path of tree selectors) is *free* or *bound*; the definition proceeds along similar lines to the definition of *occurs free* given above.
- Definition: an *occurrence* of ***x*** is *bound* in ***M*** iff it is **not** free in ***M***.

Static Distance Representation

- The choice of bound variable names in an expression is arbitrary (modulo ensuring distinct, potentially conflicting variables have distinct names).
- We can eliminate explicit variable names by using the notion of "relative addressing" (widely used in machine language and assembly language): a variable reference simply has to identify which lambda abstraction introduces the variables to which it refers. We can number the lambda abstractions enclosing a variable occurrence 1, 2, ... and simply use these indices instead of variable names. Since LC includes integer constants, we will embolden the indices referring to variables to distinguish them from integer constants.
- Examples:
  - `(lambda (x) x)  --->  (lambda 1)`
  - `(lambda (x) (lambda (y) (lambda (z) ((x z)(y z)))))`

    `--->`

    `(lambda (lambda (lambda ((3 1)(2 1)))))`

Generalized Static Distance

- In LC, `lambda` abstractions are unary; only one variable appears in the parameter list.
- In practical programming languages, parameter lists can contain any finite number (within reason) of parameters.
- How can we generalize deBruijn notation to accommodate lambda abstractions of arbitrary arity?
- Does a variable reference have to be a scalar (physics terminology)?