Compilers

Scope

Alex Aiken

- Matching identifier declarations with uses
  - Important static analysis step in most languages
  - Including COOL!

- Example 1

$$\text{let } y: String \leftarrow \text{``abc'' in } y + 3$$
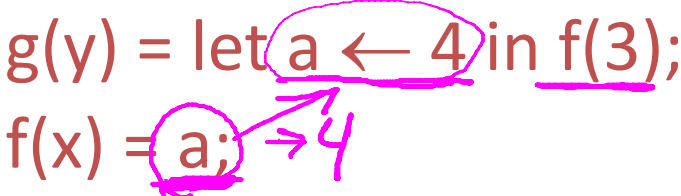
- Example 2

$$\text{let } y: Int \text{ in } x + 3$$

Alex Aiken

- The *scope* of an identifier is the portion of a program in which that identifier is accessible

- The same identifier may refer to different things in different parts of the program
  – Different scopes for same name don't overlap

- An identifier may have restricted scope

- Most languages have *static* scope
  - Scope depends only on the program text, not run-time behavior
  - Cool has static scope

- A few languages are *dynamically* scoped
  - Lisp, SNOBOL
  - Lisp has changed to mostly static scoping
  - Scope depends on execution of the program

let x: Int <- 0 in
   {
       x;
       let x: Int <- 1 in
           x;
       x;
   }

- A dynamically-scoped variable refers to the closest enclosing binding in the execution of the program

- Example

  g(y) = let a ← 4 in f(3);
  f(x) = a; → 4

- More about dynamic scope later . . .

- Cool identifier bindings are introduced by
    - Class declarations (introduce class names)
    - Method definitions (introduce method names)
    - Let expressions (introduce object id's)
    - Formal parameters (introduce object id's)
    - Attribute definitions (introduce object id's)
    - Case expressions (introduce object id's)

Alex Aiken

- Not all identifiers follow the most-closely nested rule

- For example, class definitions in Cool
  - Cannot be nested
  - Are *globally visible* throughout the program

- A class name can be used before it is defined

Class Foo {

    . . . let y: Bar in . . .

};

Class Bar {

    . . .

};

Attribute names are global within the class in which they are defined

Class Foo {

    f(): Int {(a)};

    a: Int ← 0;

}

Alex Aiken

- Method names have complex rules

- A method need not be defined in the class in which it is used, but in some parent class

- Methods may also be redefined (overridden)

Alex Aiken

Choose whether or not each variable use binds to the name on the given line

☐ Line 6 binds to line 2

☐ Line 9 binds to line 7

☐ Line 11 binds to line 2

☐ Line 11 binds to line 14

Scope

```
1    Class Foo {
2        f(x: Int): Int {
3            {
4                let x: Int <- 4 in {
6                    x;
7                    let x: Int <- 7 in
8                        x;
9                    x;
10               };
11               x;
12           };
13       };
14       x: Int <- 14;
15   }
```