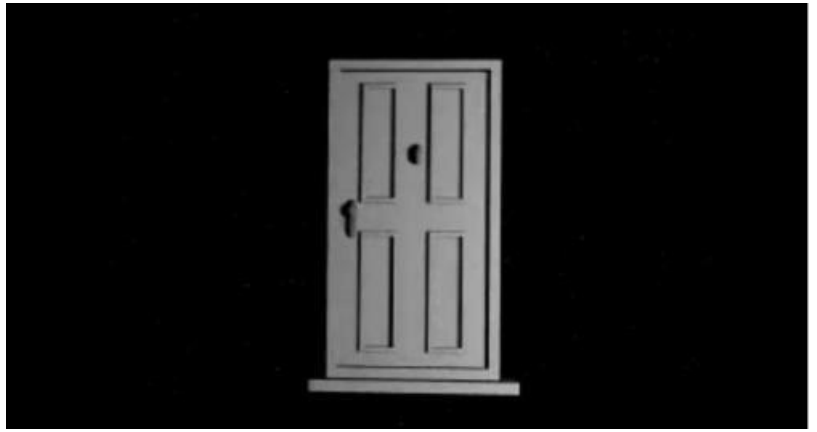
John L. Singleton
Formal Methods @ UCF

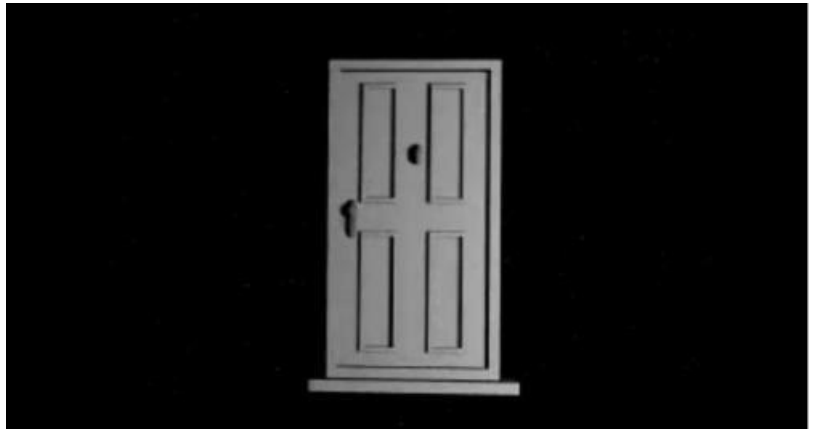
UCF

What is Scoping?



Ideas?

What is Scoping?



- The range of statements where a given variable **binding** is visible.
- “Visible” if it can be referenced in that statement.

```
int x = 1;
int a = x+1; // x is in "scope" here
```

UCF

Scoping vs Binding

- Note that **scoping** != **binding**

What's the difference?

UCF

What is Scoping?

Scope can be thought of as the lifetime of a binding.

Types of Scoping

- Static (Lexical) Scoping
  - Scoping is dependent on the static position of a variable reference in the source code.
  - If we know all the statements a binding may be visible in, we know all the types beforehand and therefore can do static typing.
- Dynamic Scoping
  - Scoping is dependent on the execution path of the program.
  - Therefore, **scope** is determined at runtime.

Is This Program Valid in Static Scoping?

```
(defvar a 100)

(defun foo ()
  (let ((a "uh oh"))
  (print-dyn))
)
(defun bar ()
  (let ((a 1))
  (print-dyn))
)

(defun print-dyn () (print a))

(foo)
(bar)
(print-dyn)
(setq a 10)
(print-dyn)
```

Is This Program Valid in Dynamic Scoping?

```
(defvar a 100)

(defun foo ()
  (let ((a "uh oh"))
  (print-dyn))
)
(defun bar ()
  (let ((a 1))
  (print-dyn))
)

(defun print-dyn () (print a))

(foo)
(bar)
(print-dyn)
(setq a 10)
(print-dyn)
```

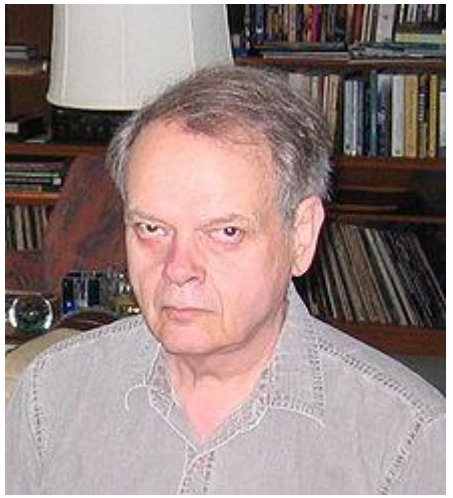
The problem here is that we can't determine which "a" print-dyn is pointing to!

```
(defvar a 100)

(defun foo ()
  (let ((a "uh oh"))
  (print-dyn))
)
(defun bar ()
  (let ((a 1))
  (print-dyn))
)

(defun print-dyn () (print a))

(foo)
(bar)
(print-dyn)
(defvar a 10)
(print-dyn)
```

Contrasting Dynamic and Static Scoping: Implementation

- We can imagine variable bindings existing on a stack
- To find the definition of a given variable, we just look at the top of the stack.

- According to John Reynolds "[Dynamic Scoping] is widely and unequivocally regarded as a design error."



UCF

Highlighting the Problem of Dynamic Scoping

$$[\![(\lambda x'.x'3)(\lambda y.y + x)]\!][\eta \mid x : \iota_{int} 1]$$

- Under static scoping this presents no problems
- Consider α-conversion (renaming)
- However, in dynamic scoping we can no longer rename x' -> x... we will get a type error.

UCF

Obligatory Problem Statement

If the effective binding is something that gets determined at runtime, is it possible use Static Typing in combination with a Dynamically Scoped language?

Obligatory Problem Statement

If the effective binding is something that gets determined at runtime, is it possible use Static Typing in combination with a Dynamically Scoped language?

It’s a commonly held belief that it is not possible!

Insight

- YES! As it turns out a sufficiently powerful type system can handle this.

Solution: Paradox

- As it turns out this has been done before…
- Lewis, Shields, Meijer, Launchbury wrote about in a POPL 2000 paper *Implicit Parameters: Dynamic Scoping with Static Types.*
- Used Haskell (wrote an extension to GHC) which introduced implicit types

UCF

Paradox: A Brief Tour

```
//
// Computes factorial of 10 (and in parallel the sum)
//

fn Int fact(Int n) {

   sum := sum+n;

   if (n=0){
      return n;
   }

   return n*(fact(n-1));
}

Int factOf10;
factOf10 := fact(10);
```

UCF

Paradox: Where it Gets Interesting

```
//
// Computes factorial of 10 (and in parallel the sum)
//

fn Int fact(Int n) implicitly [Int sum] {

   sum := sum+n;

   if (n=0){
      return n;
   }

   return n*(fact(n-1));
}

Int sum;
Int factOf10;

sum :=0;

factOf10 := fact(10);
```

UCF

Paradox: Where it Gets Crazy

```
data IntToInt = (Int -> Int -> Int)


fn Int SomeFunction(Int a, IntToInt n) implicitly [Int b] {
   return n(a,b);
}
```

Paradox: Demonstration

Concluding Remarks

- It’s possible to combine dynamic scoping and static typing
- Challenge ideas that seem obvious!
- Haskell is a great language to write a language project in.

UCF