Programming Paradigms

Names, Scopes, and Bindings (Part 5)

Prof. Dr. Michael Pradel

Software Lab, University of Stuttgart

Summer 2020

Overview

- **Object lifetime and storage management**
- **Scopes**
- **Aliasing and overloading**
- **Binding of referencing environments**

Referencing Environment

Complete set of bindings at a point in the program

- Determined by scoping rules (e.g., static or dynamic scoping)

What if we create a reference to a function?

- When to apply the scoping rules?

Example

Pseudo code:

```
function a() {
  var x = 23;
  function b() {
    console.log(x);
  }
  x = 42;
  return b;
}
b = a();
var x = 5;
b();
```

Example

Pseudo code:

```
function a() {
  var x = 23;
  function b() {
    console.log(x);
  }
  x = 42;
  return b;
}
b = a();
var x = 5;
b();
```

Example

Pseudo code:

```
function a() {
  var x = 23;
  function b() {
    console.log(x);
  }
  x = 42;
  return b;
}
b = a();
var x = 5;
b();
```

Reference to a function

Function called here

Example

Pseudo code:

```
function a() {
  var x = 23;
  function b() {
    console.log(x);
  }
  x = 42;
  return b;
}
b = a();
var x = 5;
b();
```

What memory object is `x` bound to?

Shallow Binding

Referencing environment created when function is called

- Common in languages with dynamic scoping

Shallow Binding

Referencing environment created when function is called

- Common in languages with dynamic scoping

```
function a() {
  var x = 23;
  function b() {
    console.log(x);
  }
  x = 42;
  return b;
}
b = a();
var x = 5;
b();
```

Shallow Binding

Referencing environment created when function is called

- Common in languages with dynamic scoping

```
function a() {
  var x = 23;
  function b() {
    console.log(x);
  }
  x = 42;
  return b;
}
b = a();
var x = 5;
b();
```

x bound to the global variable initialized to 5; code prints 5

Deep Binding

Referencing environment created when the reference to the function is created

- Common in languages with static scoping

```
function a() {
  var x = 23;
  function b() {
    console.log(x);
  }
  x = 42;
  return b;
}
b = a();
var x = 5;
b();
```

Deep Binding

Referencing environment created when the reference to the function is created

- Common in languages with static scoping

```
function a() {
  var x = 23;
  function b() {
    console.log(x);
  }
  x = 42;
  return b;
}
b = a();
var x = 5;
b();
```

x bound to the local variable initialized to 23; code prints 42, as this is the most recent value of x

Closure

- **Implementation of deep binding**
- **Closure = Representation of referencing environment + function itself**
- **When creating reference to function, closure is created**

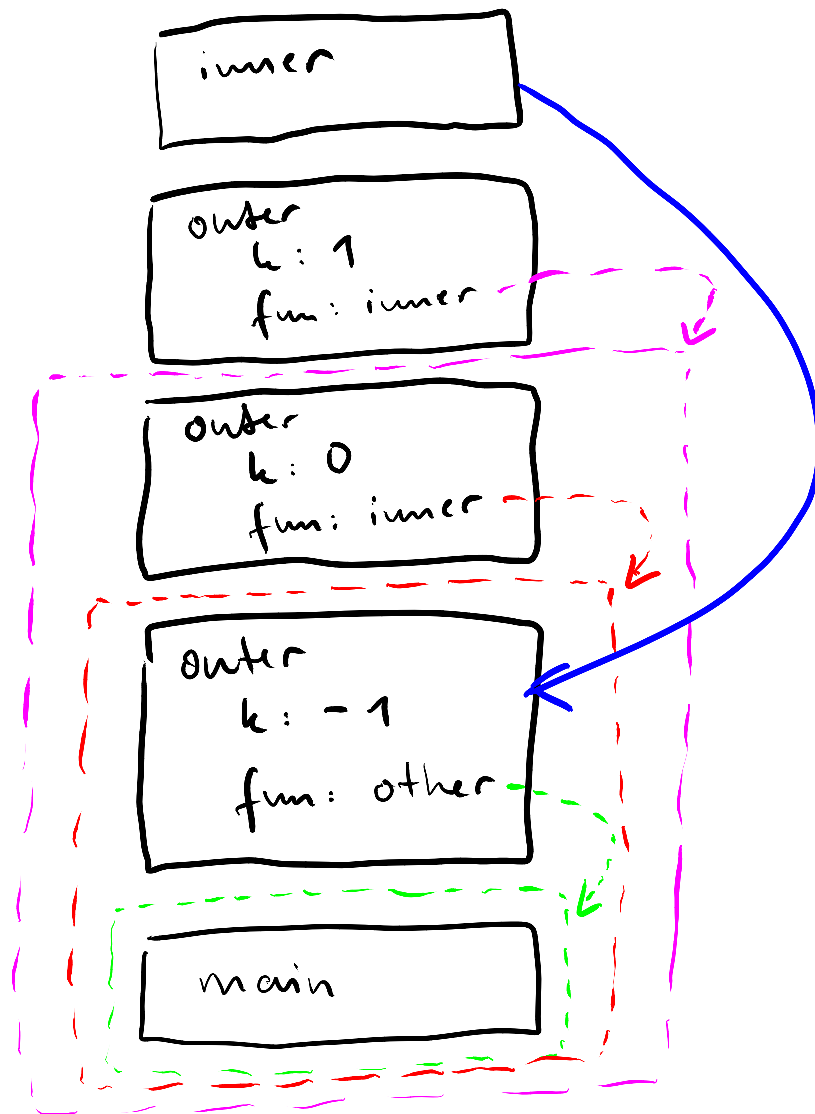
Example: Closures

```
function outer(k, fun) {
    function inner() {
        console.log(k);
    }

    if (k > 0)
        fun();
    else
        outer(k + 1, inner)
}

function other() {}

outer(-1, other);
```



→ .. static

- - → - - → - - → } referencing environment captured by closures

prints 0

Quiz: Scopes and Bindings

Which of the following statements is true?

- Scoping rules (e.g., static or dynamic) determine how names are bound to memory objects.
- Built-in objects can be thought of as an invisible, outer scope.
- Languages with pointers don't have any aliasing.
- Closures implement referencing environments created via shallow binding.

Please vote via Ilias.

Quiz: Scopes and Bindings

Which of the following statements is true?

- Scoping rules (e.g., static or dynamic) determine how names are bound to memory objects.
- Built-in objects can be thought of as an invisible, outer scope.
- ~~Languages with pointers don't have any aliasing.~~
- ~~Closures implement referencing environments created via shallow binding.~~

Please vote via Ilias.