

CSE130/230 - WEEK 5 DI

Interpreters, PA4, and beyond

David Justo

The Plan for Today

1. Interpreters
2. PA 4 Overview
- 3. PA4 Concepts**

The Plan for Today

1. Interpreters
2. PA 4 Overview
3. PA4 Concepts
 - a. Environments
 - b. Closures
 - c. Apps
 - i. Let, Letrec
 - d. Native ops



Obligatory Halloween meme

Interpreters

~Of the realms beyond~

What is an Interpreter?

An interpreter is a program that executes other programs (it can interpret / understand source code) without the need of compiling them.

What is an Interpreter?

An interpreter is a program that executes other programs (it can interpret / understand source code) without the need of compiling them.

Usually, it consists of an **evaluation** loop that recursively resolves the arguments to an operator from expressions to values.

What is an Interpreter?

An interpreter is a program that executes other programs (it can interpret / understand source code) without the need of compiling them.

Usually, it consists of an **evaluation** loop that recursively resolves the arguments to an operator from expressions to values.

```
let rec eval (evn,e) = failwith "to be written"
```

What is an Interpreter?

An interpreter is a program that executes other programs (it can interpret / understand source code) without the need of compiling them.

Usually, it consists of an **evaluation** loop that recursively resolves the arguments to an operator from expressions to values.

```
let rec eval (evn,e) = failwith "to be written"
```



The expression you're currently evaluating

What is an Interpreter?

An interpreter is a program that executes other programs (it can interpret / understand source code) without the need of compiling them.

Usually, it consists of an **evaluation** loop that recursively resolves the arguments to an operator from expressions to values.

```
let rec eval (env,e) = failwith "to be written"
```



The expression you’re currently evaluating

The environment: array of tuples of the form (“var”, “value”)

How to implement an interpreter?

```
let rec eval (env,e) = failwith "to be written"
```

How to implement an interpreter?

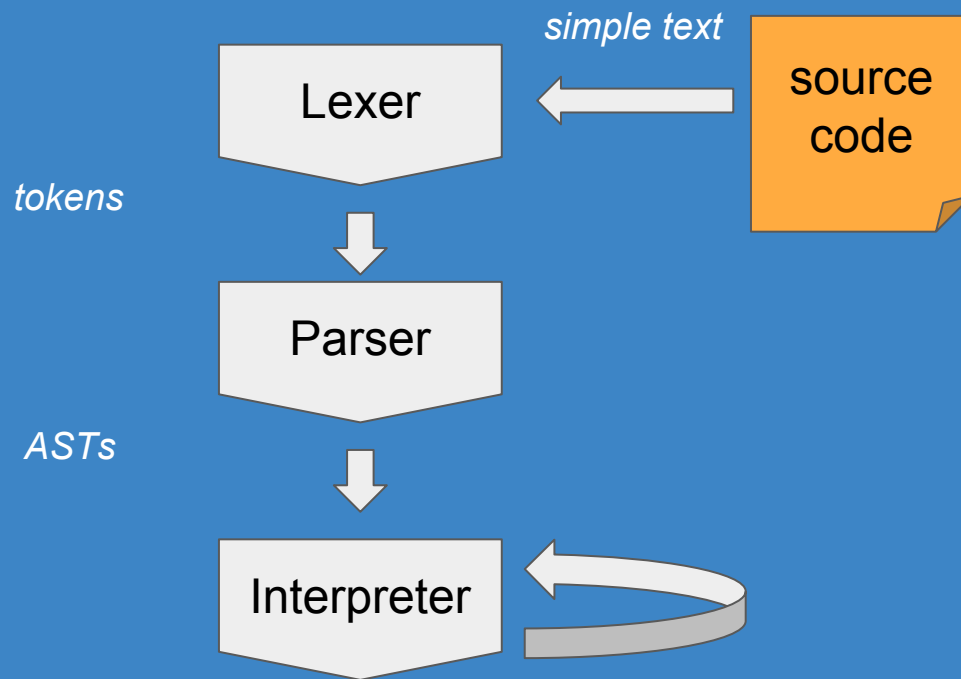
```
let rec eval (env,e) =
```

Pattern match `e` with the data constructors and handle each case;;

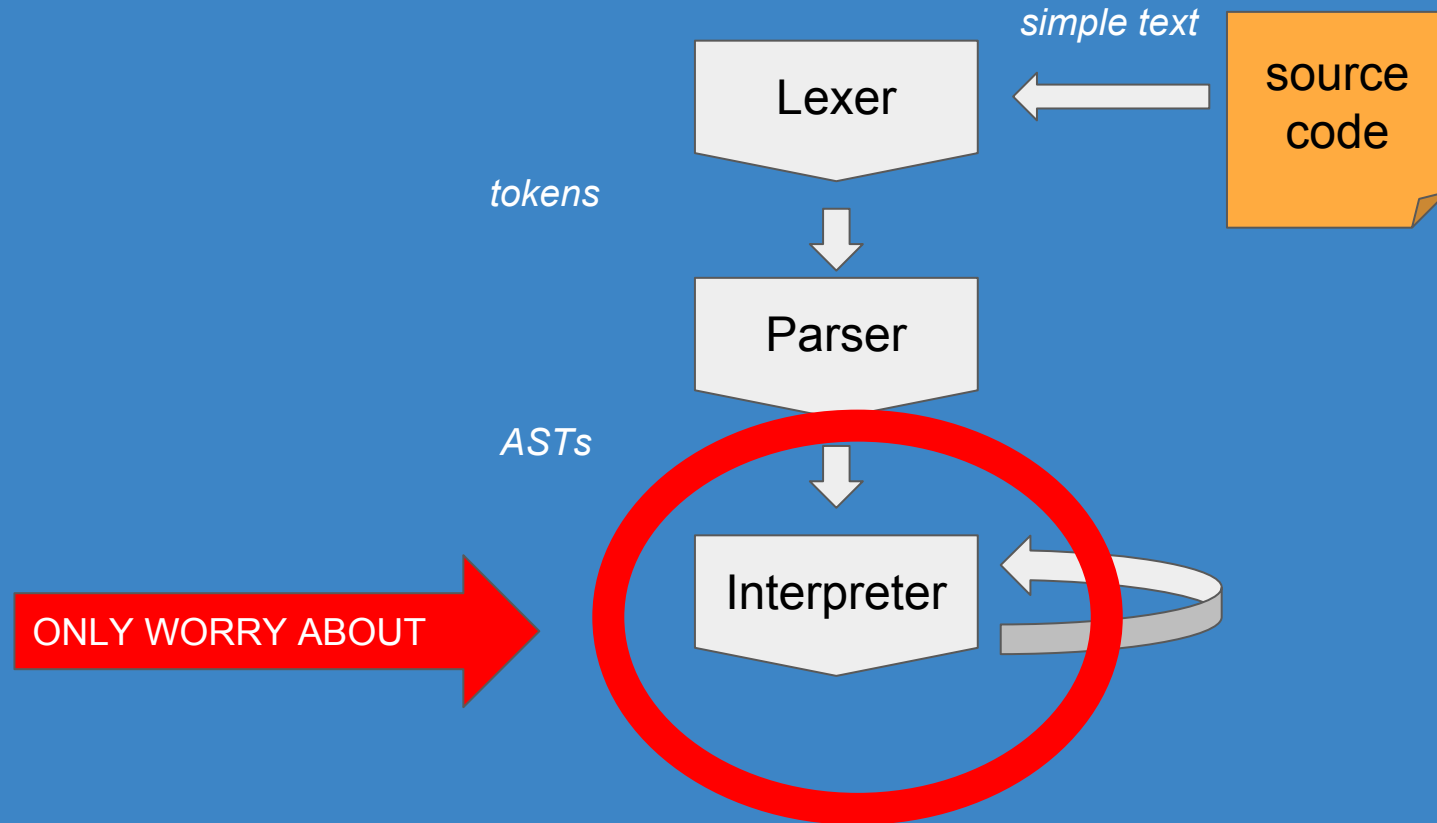
Sometimes add a new variable to `env`

Also check that types are correct: cannot do $4 + \text{"Burger"}$, for example

The Big Picture



The Big Picture



Environments

~Of the undead~

Let's run some code in our heads!

let a = 1 in

let b = 2 in

let a = a + 1 in

a + b

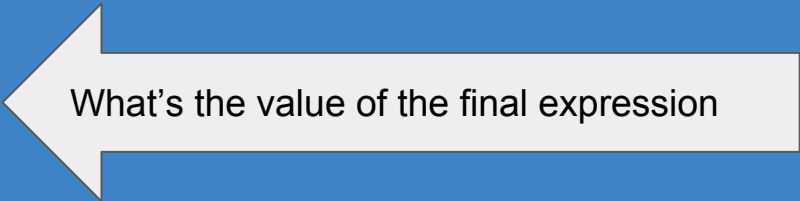
Let's run some code in our heads!

let a = 1 in

let b = 2 in

let a = a + 1 in

a + b



What's the value of the final expression

Let's run some code in our heads!

```
let a = 1 in
```

```
  let b = 2 in
```

```
    let a = a + 1 in
```

```
      a + b // 4
```

Let's run some code in our heads!

let **a** = 1 in

let **b** = 2 in

let **a** = **a** + 1 in

a + **b** // 4



a and **a** are different!

How is the environment filled?

let **a** = 1 in



let **b** = 2 in

let **a** = **a** + 1 in

a + **b** // 4

Environment

(a,1)

How is the environment filled?

let **a** = 1 in

let **b** = 2 in



let **a** = **a** + 1 in

a + **b** // 4

Environment

(b,2) (a,1)

How is the environment filled?

let **a** = 1 in

let **b** = 2 in

let **a** = **a** + 1 in

a + **b** // 4



Environment

(a, 2) (b,2) (a,1)

How is the environment filled?

let **a** = 1 in

let **b** = 2 in

let **a** = **a** + 1 in

a + **b** // 4



Environment

(a, 2) (b,2) (a,1)

How is the environment filled?

let **a** = 1 in

let **b** = 2 in

let **a** = **a** + 1 in

a + **b** // 4



Environment

(a, 2) (b, 2) (a, 1)

ListAssoc finds the left-most definition of any variable in the environment. So **a** + **b** will resolve to $2 + 2 = 4$ instead of $1 + 2 = 3$

Closures

From the beyond

Closures

Construction: **Closure**(env, name, argument, body)

Closures

Construction: **Closure**(env, name, argument, body)



Either: **None** or **Some** 'name'

Closures

Construction: **Closure**(env, name, argument, body)



This is the
representation of a
function in your
environment



Either: **None** or **Some** 'name'

When is a closure created?

let a = 1 in

let b = 2 in

let foo = fun x -> x + 1

in x + b + a + 1

When is a closure created?

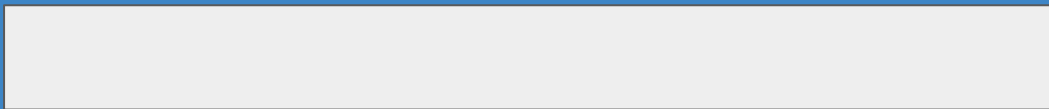
Environment

```
let a = 1 in
```

```
  let b = 2 in
```

```
    let foo = fun x -> x + 1
```

```
      in x + b + a + 1
```



When is a closure created?

Environment

let a = 1 in



let b = 2 in

let foo = fun x -> x + 1

in x + b + a + 1

When is a closure created?

Environment

let a = 1 in

let b = 2 in

let foo = fun x -> x + 1

in x + b + a + 1

(b,2),(a,1)



When is a closure created?

Environment

let a = 1 in

let b = 2 in

let foo = fun x -> x + 1



in x + b + a + 1

(foo, Closure([(b,2), (a,1)], None, x, x+1),(b,2),(a,1))

When is a closure created?

let a = 1 in

let b = 2 in

let foo = fun x -> x + 1

in x + b + a + 1

Environment

(foo, Closure([(b,2),(a,1)], None, x, x+1),(b,2),(a,1))



The name is **None**.
When would it need a name?

App

~Of dark magic~

How do you call a function?

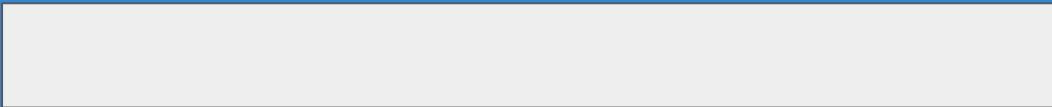
Environment

```
let a = 1 in
```

```
  let b = 2 in
```

```
    let foo = fun x -> x + a
```

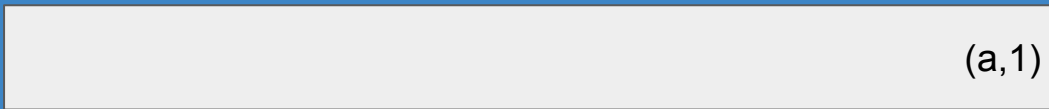
```
      in foo 5
```



How do you call a function?

Environment

let a = 1 in



let b = 2 in

let foo = fun x -> x + a

in foo 5

How do you call a function?

Environment

let a = 1 in

let b = 2 in

let foo = fun x -> x + a

in foo 5



(b,2), (a,1)

How do you call a function?

Environment

let a = 1 in

let b = 2 in

let foo = fun x -> x + a

in foo 5

(Closure([(b,2),(a,1)], None, x, x + a)),(b,2),(a,1)



How do you call a function?

Environment

let a = 1 in

let b = 2 in

let foo = fun x -> x + a

in foo 5



(Closure([(b,2),(a,1)], None, x, x + a)),(b,2),(a,1)

How do you call a function?

Environment

let a = 1 in

(Closure([(b,2),(a,1)], None, x, x + a)),(b,2),(a,1)

let b = 2 in

Assuming we're in eval....

let foo = fun x -> x + a

in foo 5



How do you call a function?

Environment

let a = 1 in

(Closure([(b,2),(a,1)], None, x, x + a)),(b,2),(a,1)

let b = 2 in

let foo = fun x -> x + a

in foo 5



Assuming we're in eval....

1. We take the environment inside the closure

[(b,2), (a,1)]

How do you call a function?

Environment

let a = 1 in

let b = 2 in

let foo = fun x -> x + a

in foo 5



(Closure([(b,2),(a,1)], None, x, x + a)),(b,2),(a,1)

Assuming we're in eval....

1. We take the environment [(b,2), (a,1)] inside the closure
2. Then you bind the parameter (x,5) to the passed value

How do you call a function?

Environment

let a = 1 in

let b = 2 in

let foo = fun x -> x + a

in foo 5



(Closure([(b,2),(a,1)], None, x, x + a)),(b,2),(a,1)

Assuming we're in eval....

1. We take the environment [(b,2), (a,1)] inside the closure
2. Then you bind the parameter (x,5) to the passed value
3. Then you pass the new bind [(x,5),(b,2),(a,1)] to the environment

How do you call a function?

Environment

let a = 1 in

let b = 2 in

let foo = fun x -> x + a

in foo 5



(Closure([(b,2),(a,1)], None, x, x + a)),(b,2),(a,1)

Assuming we're in eval....

1. We take the environment [(b,2), (a,1)] inside the closure
2. Then you bind the parameter (x,5) to the passed value
3. Then you pass the new bind [(x,5),(b,2),(a,1)] to the environment
4. And you evaluate the body in eval [(x,5),(b,2),(a,1)] the closure with the new (x+a) environment

Whatabout Letrec

Now what happens when you have a recursive function?

Whattabout Letrec

Now what happens when you have a recursive function?

Whoever gets this right gets a TOBLERONE

Whattabout Letrec

Simple.

Whattabout Letrec

If letrec creates a function, make sure that
function has a name!

letrec x = e1 **in** e2

Whattabout Letrec

If letrec creates a function, make sure that
function has a name!

letrec x = e1 **in** e2



Name is 'Some x'

Map, Fold

Ghouls, and Ghosts

To implement the native HOFs

**You need to first build the functions using your
AST constructors!**

To implement the native HOFs

What are the parameters to map?

To implement the native HOFs

What are the parameters to map?

`('a -> 'b) -> 'a list -> 'b list`

To implement the native HOFs

How do you build a closure for a function named 'map' with takes an argument 'f'?

To implement the native HOFs

How do you build a closure for a function named 'map' with takes an argument 'f'?

Closure(**env**, **Some** 'map', 'f', *<body>*)

To implement the native HOFs

Closure(**env**, **Some** 'map', 'f', *<body>*)



The body is an expression. Thus, its
constructed from AST nodes!

To implement the native HOFs

Closure(**env**, **Some** 'map', 'f', **<body>**)



The body is an expression. Thus, its constructed from AST nodes!

Since we have yet to capture the ``a' list`` parameter, we may want to start with the **Fun** constructor. The rest is up to you

HW4 tips

And other undead creatures

HW4 in a slide

Problem #1: evaluate explicit types and binary operations

- Use **BinOp**'s middle argument to find which binary operator (Plus, Minus) is used
- Check for that values have the right type for their operators.
 - Else, **raise (MLFailure "ERROR TEXT")**

HW4 reminders in a slide

Problem #1: evaluate explicit types and binary operations

- Use **BinOp**'s middle argument to find which binary operator (Plus, Minus) is used
- Check for that values have the right type for their operators.
 - Else, **raise (MLFailure "ERROR TEXT")**

Problem #2: Let, Letrec and App

- For the “Lets”, you’ll be updating the environment. Remember to add the **newly named** function in the **letrec** case
- For App, you’ll be updating the environment with function parameter

HW4 reminders in a slide

Problem #1: evaluate explicit types and binary operations

- Use **BinOp**'s middle argument to find which binary operator (Plus, Minus) is used
- Check for that values have the right type for their operators.
 - Else, **raise (MLFailure "ERROR TEXT")**

Problem #2: Let, Letrec and App

- For the “Lets”, you’ll be updating the environment. Remember to add the **newly named** function in the **letrec** case
- For App, you’ll be updating the environment with function parameter

Problem #3: Native ops

- Start early!

Fin.

Happy Halloween!