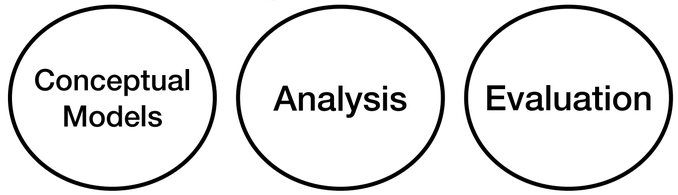
Core Design Concepts Discussed:

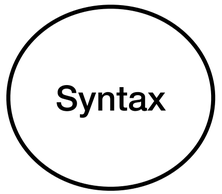


Recursive Functions and their Evaluation

Harley Eades III

Core Design Concepts:

Recursive Functions in OCaml

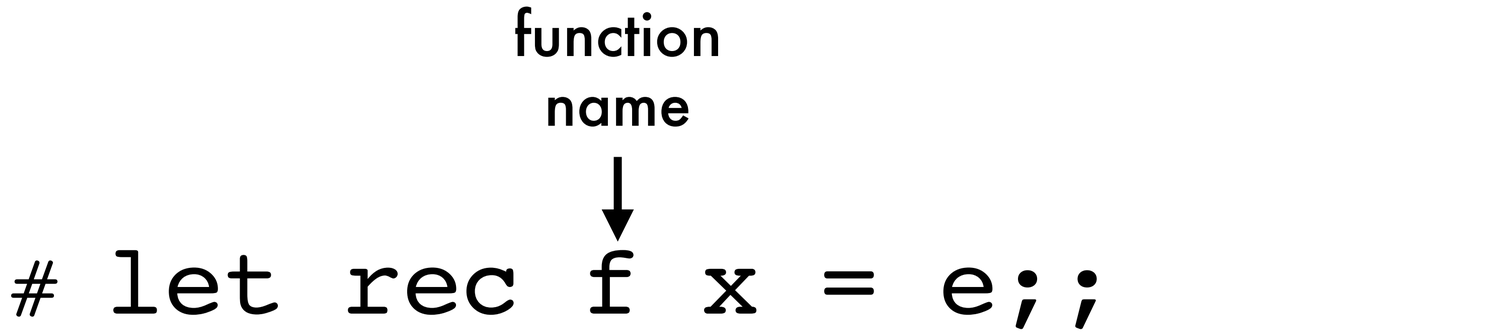



```
# let rec f x = e;;
```

Core Design Concepts:

Syntax

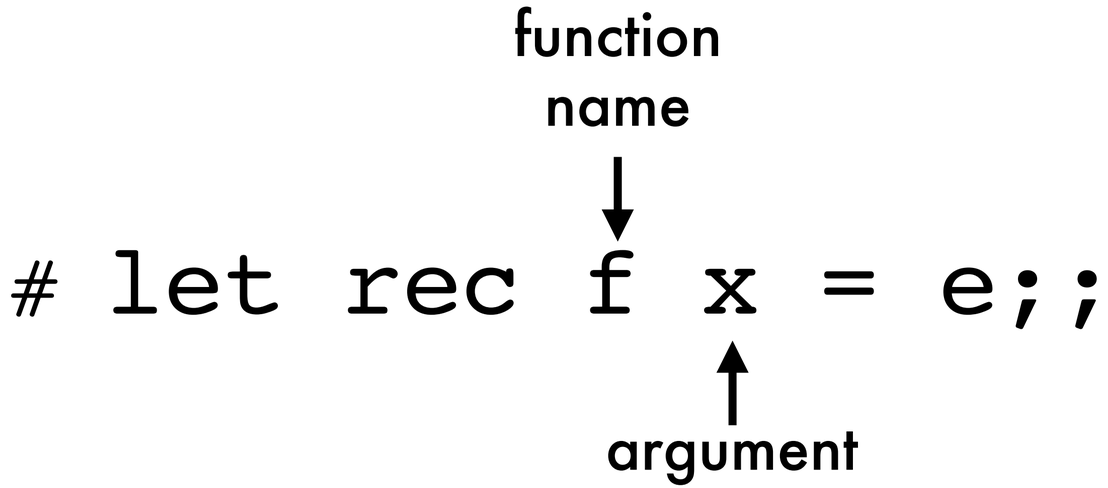
Recursive Functions in OCaml



Core Design Concepts:

Syntax

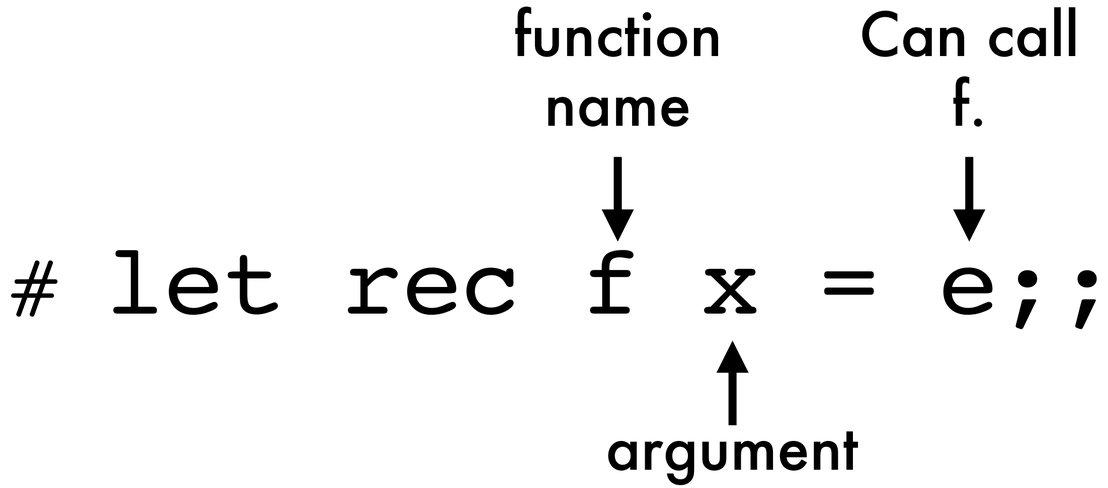
Recursive Functions in OCaml



Core Design Concepts:

Syntax

Recursive Functions in OCaml



Core Design Concepts:

But, what about performance?

Core Design Concepts:

The Function Call Stack

Activation Record:

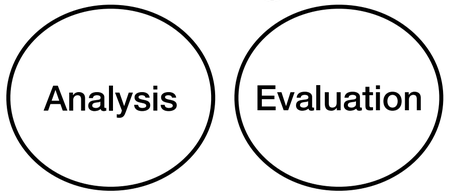
The location in memory where an executing function stores its bindings.

Activation records are sometimes referred to as frames.

Consider evaluating the following function:

```
1: let cube n =
2:   let c = n*n*n in
3:       c;;
4: let main =
5:   let n = 5 in
6:   let ans = cube n in
7:       ans;;
8: main;;
```

Core Design Concepts:



The Function Call Stack

Activation Record:

The location in memory where an executing function stores its binding.

Activation records are sometimes referred to as frames.

Activation Record: program initialization

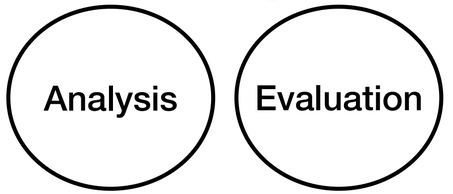
Frame	Symbol	Value
init line: 8	`cube` `main`	`<fun>` `<fun>`

Consider evaluating the following function:

```
1: let cube n =
2:    let c = n*n*n in
3:        c;;
4: let main =
5:    let n = 5 in
6:    let ans = cube n in
7:        ans;;
8: main;;
```

The Function Call Stack

Core Design Concepts:



Activation Record:

The location in memory where an executing function stores its binding.

Activation records are sometimes referred to as frames.

Consider evaluating the following function:

```
1: let cube n =
2:    let c = n*n*n in
3:        c;;
4: let main =
5:    let n = 5 in
6:    let ans = cube n in
7:        ans;;
8: main;;
```

Activation Record: program initialization

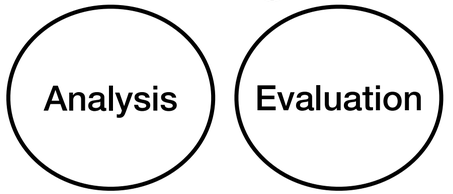
Frame	Symbol	Value
init line: 8	`cube` `main`	`<fun>` `<fun>`

Activation Record: after calling `main`

Frame	Symbol	Value
init line: 8	`cube` `main`	`<fun>` `<fun>`
`main` line: 6	`n` `ans`	`5` `?`

The Function Call Stack

Core Design Concepts:



Activation Record:

The location in memory where an executing function stores its binding.

Activation records are sometimes referred to as frames.

Consider evaluating the following function:

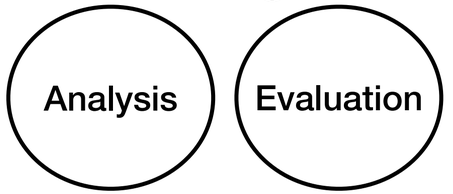
```
1: let cube n =
2:    let c = n*n*n in
3:        c;;
4: let main =
5:    let n = 5 in
6:    let ans = cube n in
7:        ans;;
8: main;;
```

Activation Record: after calling `main`

Frame	Symbol	Value
init line: 8	`cube` `main`	`<fun>` `<fun>`
`main` line: 6	`n` `ans`	`5` `?`
`cube` line: 2	`n` `c`	`5` `125`

The Function Call Stack

Core Design Concepts:



Activation Record:

The location in memory where an executing function stores its binding.

Activation records are sometimes referred to as frames.

Consider evaluating the following function:

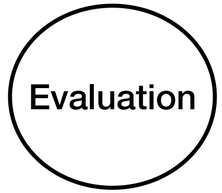
```
1: let cube n =
2:    let c = n*n*n in
3:        c;;
4: let main =
5:    let n = 5 in
6:    let ans = cube n in
7:        ans;;
8: main;;
```

Activation Record: after calling `main`

Frame	Symbol	Value
init line: 8	`cube` `main`	`<fun>` `<fun>`
`main` line: 6	`n` `ans`	`5` `125`
`cube` line: 2	`n` `c`	`5` `125`

Evaluating Recursive Functions

Core Design Concepts:



Consider evaluating the following recursive function:

```
1. let rec ackermann m n =
2.  if m == 0
3.  then let ret = n + 1 in ret
4.  else if n == 0
5.       then let ack  = ackermann (m - 1) 1 in ack
6.       else let ack1 = ackermann m (m - 1) in
7.            let ack2 = ackermann (m - 1) ack1 in
8.              ack2
9.
10. let main =
11.   let m = 1 in
12.    let n = 0 in
13.      let ans1 = ackermann m n in
14.        let ans2 = ackermann m m in
15.          ans1 + ans2
16.
17. main;;
```

Evaluating Recursive Functions

Core Design Concepts:

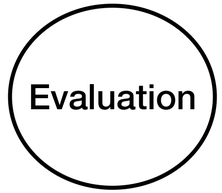
Consider evaluating the following recursive function:

```
1. let rec ackermann m n =
2.  if m == 0
3.  then let ret = n + 1 in ret
4.  else if n == 0
5.       then let ack  = ackermann (m - 1) 1 in ack
6.       else let ack1 = ackermann m (m - 1) in
7.            let ack2 = ackermann (m - 1) ack1 in
8.              ack2
9.
10. let main =
11.   let m = 1 in
12.    let n = 0 in
13.      let ans1 = ackermann m n in
14.        let ans2 = ackermann m m in
15.          ans1 + ans2
16.
17. main;;
```

Frame	Symbol	Value
init line: 17	ackermann main	<fun> <fun>

Evaluating Recursive Functions

Core Design Concepts:



Consider evaluating the following recursive function:

```
1. let rec ackermann m n =
2.   if m == 0
3.   then let ret = n + 1 in ret
4.   else if n == 0
5.        then let ack  = ackermann (m - 1) 1 in ack
6.        else let ack1 = ackermann m (m - 1) in
7.             let ack2 = ackermann (m - 1) ack1 in
8.               ack2
9.
10. let main =
11.   let m = 1 in
12.    let n = 0 in
13.      let ans1 = ackermann m n in
14.        let ans2 = ackermann m m in
15.          ans1 + ans2
16.
17. main;;
```

Frame	Symbol	Value
init line: 17	ackermann main	<fun> <fun>
main line: 14	m n ans1 ans2	1 0 ? ?

Evaluating Recursive Functions

Core Design Concepts:

Consider evaluating the following recursive function:

```
1. let rec ackermann m n =
2.   if m == 0
3.   then let ret = n + 1 in ret
4.   else if n == 0
5.        then let ack  = ackermann (m - 1) 1 in ack
6.        else let ack1 = ackermann m (m - 1) in
7.             let ack2 = ackermann (m - 1) ack1 in
8.               ack2
9.
10. let main =
11.    let m = 1 in
12.     let n = 0 in
13.       let ans1 = ackermann m n in
14.         let ans2 = ackermann m m in
15.           ans1 + ans2
16.
17. main;;
```

Frame	Symbol	Value
init line: 17	ackermann main	<fun> <fun>
main line: 14	m n ans1 ans2	1 0 ? ?
ackermann line: 5	m n ack	1 0 ?

Evaluating Recursive Functions

Core Design Concepts:

Consider evaluating the following recursive function:

```
1. let rec ackermann m n =
2.   if m == 0
3.   then let ret = n + 1 in ret
4.   else if n == 0
5.        then let ack  = ackermann (m - 1) 1 in ack
6.        else let ack1 = ackermann m (m - 1) in
7.             let ack2 = ackermann (m - 1) ack1 in
8.               ack2
9.
10. let main =
11.    let m = 1 in
12.     let n = 0 in
13.       let ans1 = ackermann m n in
14.         let ans2 = ackermann m m in
15.           ans1 + ans2
16.
17. main;;
```

Frame	Symbol	Value
init line: 17	ackermann main	<fun> <fun>
main line: 14	m n ans1 ans2	1 0 ? ?
ackermann line: 5	m n ack	1 0 ?
ackermann line: 3	m n ret	0 1 2

Evaluating Recursive Functions

Core Design Concepts:

Consider evaluating the following recursive function:

```
1. let rec ackermann m n =
2.   if m == 0
3.   then let ret = n + 1 in ret
4.   else if n == 0
5.        then let ack  = ackermann (m - 1) 1 in ack
6.        else let ack1 = ackermann m (m - 1) in
7.             let ack2 = ackermann (m - 1) ack1 in
8.               ack2
9.
10. let main =
11.    let m = 1 in
12.     let n = 0 in
13.       let ans1 = ackermann m n in
14.         let ans2 = ackermann m m in
15.           ans1 + ans2
16.
17. main;;
```

Frame	Symbol	Value
init line: 17	ackermann main	<fun> <fun>
main line: 14	m n ans1 ans2	1 0 ? ?
ackermann line: 5	m n ack	1 0 2
ackermann line: 3	m n ret	0 1 2

Evaluating Recursive Functions

Core Design Concepts:

Consider evaluating the following recursive function:

```
1. let rec ackermann m n =
2.   if m == 0
3.   then let ret = n + 1 in ret
4.   else if n == 0
5.        then let ack  = ackermann (m - 1) 1 in ack
6.        else let ack1 = ackermann m (m - 1) in
7.             let ack2 = ackermann (m - 1) ack1 in
8.               ack2
9.
10. let main =
11.    let m = 1 in
12.     let n = 0 in
13.       let ans1 = ackermann m n in
14.         let ans2 = ackermann m m in
15.           ans1 + ans2
16.
17. main;;
```

Frame	Symbol	Value
init line: 17	ackermann main	<fun> <fun>
main line: 14	m n ans1 ans2	1 0 2 ?
ackermann line: 5	m n ack	1 0 2
ackermann line: 3	m n ret	0 1 2

Evaluating Recursive Functions

Core Design Concepts:

Consider evaluating the following recursive function:

```
1. let rec ackermann m n =
2.  if m == 0
3.  then let ret = n + 1 in ret
4.  else if n == 0
5.       then let ack  = ackermann (m - 1) 1 in ack
6.       else let ack1 = ackermann m (m - 1) in
7.            let ack2 = ackermann (m - 1) ack1 in
8.              ack2
9.
10. let main =
11.   let m = 1 in
12.    let n = 0 in
13.      let ans1 = ackermann m n in
14.        let ans2 = ackermann m m in
15.          ans1 + ans2
16.
17. main;;
```

Frame	Symbol	Value
main line: 14	m n ans1 ans2	1 0 2 ?
ackermann line: 7	m n ack1 ack2	1 1 ? ?

Evaluating Recursive Functions

Core Design Concepts:

Consider evaluating the following recursive function:

```
1. let rec ackermann m n =
2.  if m == 0
3.  then let ret = n + 1 in ret
4.  else if n == 0
5.       then let ack  = ackermann (m - 1) 1 in ack
6.       else let ack1 = ackermann m (m - 1) in
7.            let ack2 = ackermann (m - 1) ack1 in
8.              ack2
9.
10. let main =
11.   let m = 1 in
12.    let n = 0 in
13.      let ans1 = ackermann m n in
14.        let ans2 = ackermann m m in
15.          ans1 + ans2
16.
17. main;;
```

Frame	Symbol	Value
main line: 14	m n ans1 ans2	1 0 2 ?
ackermann line: 7	m n ack1 ack2	1 1 ? ?
ackermann line: 5	m n ack	1 0 ?

Evaluating Recursive Functions

Core Design Concepts:

Consider evaluating the following recursive function:

```
1. let rec ackermann m n =
2.  if m == 0
3.  then let ret = n + 1 in ret
4.  else if n == 0
5.       then let ack  = ackermann (m - 1) 1 in ack
6.       else let ack1 = ackermann m (m - 1) in
7.            let ack2 = ackermann (m - 1) ack1 in
8.              ack2
9.
10. let main =
11.   let m = 1 in
12.    let n = 0 in
13.      let ans1 = ackermann m n in
14.        let ans2 = ackermann m m in
15.          ans1 + ans2
16.
17. main;;
```

Frame	Symbol	Value
main line: 14	m n ans1 ans2	1 0 2 ?
ackermann line: 7	m n ack1 ack2	1 1 ? ?
ackermann line: 5	m n ack	1 0 ?
ackermann line: 3	m n ret	0 1 2

Evaluating Recursive Functions

Core Design Concepts:

Consider evaluating the following recursive function:

```
1. let rec ackermann m n =
2.  if m == 0
3.  then let ret = n + 1 in ret
4.  else if n == 0
5.       then let ack  = ackermann (m - 1) 1 in ack
6.       else let ack1 = ackermann m (m - 1) in
7.            let ack2 = ackermann (m - 1) ack1 in
8.              ack2
9.
10. let main =
11.   let m = 1 in
12.    let n = 0 in
13.      let ans1 = ackermann m n in
14.        let ans2 = ackermann m m in
15.          ans1 + ans2
16.
17. main;;
```

Frame	Symbol	Value
main line: 14	m n ans1 ans2	1 0 2 ?
ackermann line: 7	m n ack1 ack2	1 1 ? ?
ackermann line: 5	m n ack	1 0 2
ackermann line: 3	m n ret	0 1 2

Evaluating Recursive Functions

Core Design Concepts:

Consider evaluating the following recursive function:

```
1. let rec ackermann m n =
2.  if m == 0
3.  then let ret = n + 1 in ret
4.  else if n == 0
5.       then let ack  = ackermann (m - 1) 1 in ack
6.       else let ack1 = ackermann m (m - 1) in
7.            let ack2 = ackermann (m - 1) ack1 in
8.              ack2
9.
10. let main =
11.   let m = 1 in
12.    let n = 0 in
13.      let ans1 = ackermann m n in
14.        let ans2 = ackermann m m in
15.          ans1 + ans2
16.
17. main;;
```

Frame	Symbol	Value
main line: 14	m n ans1 ans2	1 0 2 ?
ackermann line: 7	m n ack1 ack2	1 1 2 ?
ackermann line: 5	m n ack	1 0 2
ackermann line: 3	m n ret	0 1 2

Evaluating Recursive Functions

Core Design Concepts:

Consider evaluating the following recursive function:

```
1. let rec ackermann m n =
2.  if m == 0
3.  then let ret = n + 1 in ret
4.  else if n == 0
5.       then let ack  = ackermann (m - 1) 1 in ack
6.       else let ack1 = ackermann m (m - 1) in
7.            let ack2 = ackermann (m - 1) ack1 in
8.              ack2
9.
10. let main =
11.   let m = 1 in
12.    let n = 0 in
13.      let ans1 = ackermann m n in
14.        let ans2 = ackermann m m in
15.          ans1 + ans2
16.
17. main;;
```

Frame	Symbol	Value
main line: 14	m n ans1 ans2	1 0 2 ?
ackermann line: 7	m n ack1 ack2	1 1 2 ?
ackermann line: 5	m n ack	1 0 2
ackermann line: 3	m n ret	0 1 2
ackermann line: 3	m n ret	0 2 3

Evaluating Recursive Functions

Core Design Concepts:

Consider evaluating the following recursive function:

```
1. let rec ackermann m n =
2.   if m == 0
3.   then let ret = n + 1 in ret
4.   else if n == 0
5.        then let ack  = ackermann (m - 1) 1 in ack
6.        else let ack1 = ackermann m (m - 1) in
7.             let ack2 = ackermann (m - 1) ack1 in
8.               ack2
9.
10. let main =
11.    let m = 1 in
12.     let n = 0 in
13.       let ans1 = ackermann m n in
14.         let ans2 = ackermann m m in
15.           ans1 + ans2
16.
17. main;;
```

Frame	Symbol	Value
main line: 14	m n ans1 ans2	1 0 2 ?
ackermann line: 7	m n ack1 ack2	1 1 2 3
ackermann line: 5	m n ack	1 0 2
ackermann line: 3	m n ret	0 1 2
ackermann line: 3	m n ret	0 2 3

Evaluating Recursive Functions

Core Design Concepts:

Consider evaluating the following recursive function:

```
1. let rec ackermann m n =
2.  if m == 0
3.  then let ret = n + 1 in ret
4.  else if n == 0
5.       then let ack  = ackermann (m - 1) 1 in ack
6.       else let ack1 = ackermann m (m - 1) in
7.            let ack2 = ackermann (m - 1) ack1 in
8.              ack2
9.
10. let main =
11.   let m = 1 in
12.    let n = 0 in
13.      let ans1 = ackermann m n in
14.        let ans2 = ackermann m m in
15.          ans1 + ans2
16.
17. main;;
```

Frame	Symbol	Value
main line: 14	m n ans1 ans2	1 0 2 3
ackermann line: 7	m n ack1 ack2	1 1 2 3
ackermann line: 5	m n ack	1 0 2
ackermann line: 3	m n ret	0 1 2
ackermann line: 3	m n ret	0 2 3