Université Paris Cité

Call-by-name, Call-by-value, Call-by-need, and the Linear Lambda Calculus

Short Talk

Quentin Schroeder

M1 MPRI - Université Paris-Cité

Idea

Motivation

Goal

- understand evaluation strategies in the lambda calculus

Common strategies

Motivation

Goal

- understand evaluation strategies in the lambda calculus
- help improve current programming languages (e.g. Rust)

Common strategies

Motivation

Goal

- understand evaluation strategies in the lambda calculus
- help improve current programming languages (e.g. Rust)
- advance understanding of models of computation

Common strategies

Motivation

Goal

- understand evaluation strategies in the lambda calculus
- help improve current programming languages (e.g. Rust)
- advance understanding of models of computation

Common strategies

- Call by Name

Motivation

Goal

- understand evaluation strategies in the lambda calculus
- help improve current programming languages (e.g. Rust)
- advance understanding of models of computation

Common strategies

- Call by Name
- Call by Value

Motivation

Goal

- understand evaluation strategies in the lambda calculus
- help improve current programming languages (e.g. Rust)
- advance understanding of models of computation

Common strategies

- Call by Name
- Call by Value
- Call by Need

Call by value (Ocaml)

Nice Example

```
let rec dumb_works n = if false then dumb_works n else 42

let () = print_int (dumb_works 0)
```

Call by value (Ocaml)

Nice Example

```
let rec dumb_works n = if false then dumb_works n else 42

let () = print_int (dumb_works 0)
```

Evil Example

```
let branch b l r = if b then l else r

let rec dumb_breaks n = branch false (dumb_breaks n) 42


let () = print_int (dumb_breaks 0)
```

Call by need (Haskell)

Example

```haskell
dumb n = if False then dumb n else 42

forceEvalBranch b l r = if b then l else r

dumb2 n = forceEvalBranch False (dumb2 n) 42

main = do
    print $ dumb 0
    print $ dumb 1
    print $ dumb2 0
    print $ dumb2 1
```

Call by need (Haskell)

Example

```haskell
dumb n = if False then dumb n else 42

forceEvalBranch b l r = if b then l else r

dumb2 n = forceEvalBranch False (dumb2 n) 42

main = do
    print $ dumb 0
    print $ dumb 1
    print $ dumb2 0
    print $ dumb2 1
```

Example

```haskell
project x y = x

loop x = loop x

main = print $ project 2 (loop 3)
```

Call by Name (???)

Is there a call by name programming language?

Yes … kind of (e.g. Algol 60)

Example

Consider the following:

```
let rec h n = if n = 0 then 1 else n * h (n - 1) * h (n - 1)

let eval4x m =
  let () =  m in
  let () =  m in
  let () =  m in
  let () =  m

let () = eval4x (h 40)
```

Preliminaries

Preliminaries

Lambda Calculus

- model of computation
- basis for functional programming languages

Preliminaries

Linear Logic

- a resource sensitive logic
- can be used to priority in evaluation of proof terms

Preliminaries

Linear Logic

- a resource sensitive logic
- can be used to priority in evaluation of proof terms

A massive leap of faith

- build a linear lambda calculus [Maraist et al.(1995)Maraist, Odersky, Turner, and Wadler,]
- show that it can be used to model execution strategies

Interpretation of Call by Value

Call by Value

- evaluate the argument before the function
- the argument is evaluated at most once

Interpretation of Call by Value

Call by Value

- evaluate the argument before the function
- the argument is evaluated at most once

Call by Value via ”suspending computation”

- suspend computation of all the innermost functions calls
- evaluate from last to first

Interpretation of Call by Value

Example

```
let f x y = x + y
let () = print_int (f (1 + 2) (3 + 4))
```

Interpretation of Call by Value

Example

```
let f x y = x + y
let () = print_int (f (1 + 2) (3 + 4))
```

- suspend `f (1 + 2) (3 + 4)`

Interpretation of Call by Value

Example

```
let f x y = x + y
let () = print_int (f (1 + 2) (3 + 4))
```

- suspend `f (1 + 2) (3 + 4)`
- suspend `1 + 2`

Interpretation of Call by Value

Example

```
let f x y = x + y
let () = print_int (f (1 + 2) (3 + 4))
```

- suspend `f (1 + 2) (3 + 4)`
- suspend `1 + 2`
- suspend `3 + 4`

Interpretation of Call by Value

Example

```
let f x y = x + y
let () = print_int (f (1 + 2) (3 + 4))
```

- suspend `f (1 + 2) (3 + 4)`
- suspend `1 + 2`
- suspend `3 + 4`
- evaluate `3 + 4`

Interpretation of Call by Value

Example

```
let f x y = x + y
let () = print_int (f (1 + 2) (3 + 4))
```

- suspend `f (1 + 2) (3 + 4)`
- suspend `1 + 2`
- suspend `3 + 4`
- evaluate `3 + 4`
- evaluate `1 + 2`

Interpretation of Call by Value

Example

```
let f x y = x + y
let () = print_int (f (1 + 2) (3 + 4))
```

- suspend `f (1 + 2) (3 + 4)`
- suspend `1 + 2`
- suspend `3 + 4`
- evaluate `3 + 4`
- evaluate `1 + 2`
- evaluate `f 3 7`

Conclusion

Conclusion

In short

- Call by Name
- Call by Value
- Call by Need

Conclusion

In short

- Call by Name
- Call by Value
- Call by Need

For the next time

- Model Call by Name and Call by Need in this model
- Show interpretations have nice properties (soundness, completeness, etc.)

Thank you

References I

John Maraist, Martin Odersky, David N. Turner, and Philip Wadler.

Call-by-name, Call-by-value, Call-by-need, and the Linear Lambda Calculus.
Electron. Notes Theor. Comput. Sci., 1:370–392, January 1995.
ISSN 1571-0661.
doi: 10.1016/S1571-0661(04)00022-2.