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

Short Talk

Quentin Schroeder

MPRI - Université Paris-Cité

Idea

Motivation

Problem

We want to understand how to evaluate expressions in the lambda calculus.

- Call by Name

Motivation

Problem

We want to understand how to evaluate expressions in the lambda calculus.

- Call by Name
- Call by Value

Motivation

Problem

We want to understand how to evaluate expressions in the lambda calculus.

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

Call by value (Python)

Normal example

```
def f(x):
  return x + x

print(f(2 + 3))
```

Call by value (Python)

Normal example

```
def f(x):
  return x + x

print(f(2 + 3))
```

Evil example

```
def project(x, y):
  return x

def loop():
  return loop()

print(project(2, loop()))
```

Call by value (Ocaml)

Examples

```
let rec dumb_works n = if (n = 0 && n = 1) then dumb n else 42

let force_eval_branch b l r = if b then l else r

let rec dumb_breaks n = force_eval_branch (n = 0 && n = 1) (dumb_breaks n) 4

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

Call by need (Haskell)

Example

```
dumb n = if (n == 0 && n == 1) then dumb n else 42

forceEvalBranch b l r = if b then l else r

dumb2 n = forceEvalBranch (n == 0 && n == 1) (dumb2 n) 42

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

Call by need (Haskell)

Example

```
dumb n = if (n == 0 && n == 1) then dumb n else 42

forceEvalBranch b l r = if b then l else r

dumb2 n = forceEvalBranch (n == 0 && n == 1) (dumb2 n) 42

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

Example

```
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 . . .

Example

Consider the following:

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

let printalot m =
  let () = print_endline (string_of_int m) in
  let () = print_endline (string_of_int m) in
  let () = print_endline (string_of_int m) in
  let () = print_endline (string_of_int m)

let () = printNTimes (horrible 40)
```

Conclusion

Thank you

References I

[?,] [?,]

Thibaut Balabonski, Antoine Lanco, and Guillaume Melquiond.
A strong call-by-need calculus.
Logical Methods in Computer Science, 19(1):39, March 2023.
doi: 10.46298/lmcs-19(1:21)2023.
URL https://inria.hal.science/hal-03409681.

Yannick Forster, Fabian Kunze, and Marc Roth.
The weak call-by-value -calculus is reasonable for both time and space.
Proc. ACM Program. Lang., 4(POPL), dec 2019.
doi: 10.1145/3371095.
URL https://doi.org/10.1145/3371095.

References II

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.

G. D. Plotkin.

Call-by-name, call-by-value and the λ-calculus.

Theoret. Comput. Sci., 1(2):125–159, December 1975.

ISSN 0304-3975.

doi: 10.1016/0304-3975(75)90017-1.