Technology Group

Complexity

Simplified

Functional Programming

Authored & Presented By : **Amit Mulay**

Technical Evangelist

Philosophy of Laziness

- More work you do chances are that you do more mistake
- So avoid doing work
- And if you do work then only do once!

Why waste life in doing same thing again and again?

How about this philosophy in program execution??

ANYWAYS for next few slides keep in mind following scenario

Think of application where you need to lead loads of data and process it.

Only issue is that data is so huge that loading memory takes time

And you are also facing lots of out of memory issues!

Laziness

- Lazy evaluation

```
let donothing f = ()
let rec fact n =
    match n with
    | 0 | 1-> 1
    | n  -> n * fact(n-1)
let d = donothing (lazy fact( fact (fact 10000)))
```

```
&& ||
```

```
Employee.smartlevel() > 100 && Employee.hardworkinglevel() > 6 &&
employee. friends() > 100
            {
                        smartness = Employee.smartlevel()
            }
```

Laziness

- Lazy data structure

```
let infinteseq = Seq.unfold(fun x -> Some(x, x+1)) 1

let infinteseqoperated =
        infinteseq
        |> Seq.map(fun x -> x*x)
        |>  Seq.map(fun x -> x+4)

let result = infinteseqoperated |> Seq.take 30 |> Seq.toList
```

LazyList : seq + caching

Laziness

Requirement : Customers have list of resources.

1. count number of resources per client.
2. give me customers with resources less than 3.

```
let resourcesount customer =
                        c.resources
                        |> Seq.scan(fun x y -> x+1) 0
// count
Customers
|> Seq.map(fun c -> cid, resourcecount c)

// < 3
Customers
|> Seq.map(fun c -> cid, resourcecount c)
|> Seq.filter(fun (c, rc) -> !rc.exists(fun c -> c > 3))
```

Stack overflow

- Tail recursion

Any recursive function can be converted to tail recursive version. There are techniques (easy to remember).

Tail recursive functions are internally converted to while loop and they don't use stack!!

```
let fact n = match n with
             | 0 | 1 -> n
             | n -> n * fact(n-1)

let fact n =
        let innerfact res n = match n with
                              | 0 | 1 -> res
                              | n -> innerfact  (res*n) (n-1)
        innerfact 1 n
```

Stack overflow

- Continuous passing style

```
let rec fib = function
                | 0 | 1 -> 1
                | n -> fib (n-1) + fib (n-2)

let rec fib_cps n k =
 match n with
 | 0 | 1 -> k 1
 | n -> fib_cps (n-1) (fun r1 -> fib_cps (n-2) (fun r2 -> k (r1+r2)))
```

Monad

- Monad way
  - Seq
  - Avoid pollution of infra
  - Function signature
- Bind
- Can be used for lots of concepts like async!

```
async {
        let! Data = downloadfromurl("lll")
        processData
}
```

Monad

```
let rec fib = function
                | 0 | 1 -> 1
                | n -> fib (n-1) + fib (n-2)

type ContinuationBuilder() =
    member this.Bind(m, f) = fun c -> m (fun a -> f a c)
    member this.Return x = fun k -> k x

let cont = ContinuationBuilder()

let rec fib n =
    cont {
        match n with
        | 0 | 1 -> return 1
        | n -> let! r1 = fib(n-1)
               let! r2 = fib(n-2)
               return r1+r2
    }
```

General purpose programming

- Elegant
- Superior (quality, productivity, modern hardware)
- Functional programmers are less
- Entire world (frameworks/tools) today is imperative/object oriented.
- Church-Turing thesis

Data Analytics/Big data

- Data oriented designs
- Not dependent on OOPS/existing imperative worlds
- Target programmers are even mathematician/statisticians
- Algorithms can be abstracted from underline hardware.
- Lazy evaluation concepts enable to handle large amounts of data with less technical knowledge

Mobile

- Relatively new and hence no baggage of imperative world
- GPUs

Domains

- Telephony
- Chatting
- Messaging
- Gaming AI
- BFSI

Internet as big source of data

Go grab data from the web, don’t just think of it as text, bring structure into it and then [...] try out different ways of presenting it, but very interactively. [...] Write a little bit of code that may have your specific algorithm for processing that data. [Gates, 2008]

Video

- 48 secs
- 42 secs
- 32 secs

Any Questions?

Thank you!