# Programmazione parallela in Haskell
## strategie e prestazioni

Candidato: Andrea Senese
Relatore: Ugo de'Liguoro

Dipartimento di Informatica - 9 Dicembre 2019

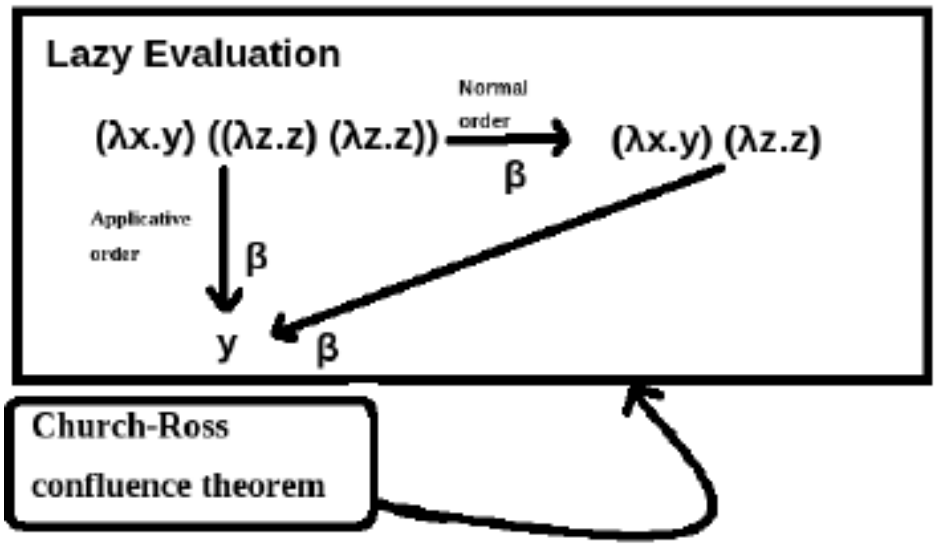
# Table of Contents

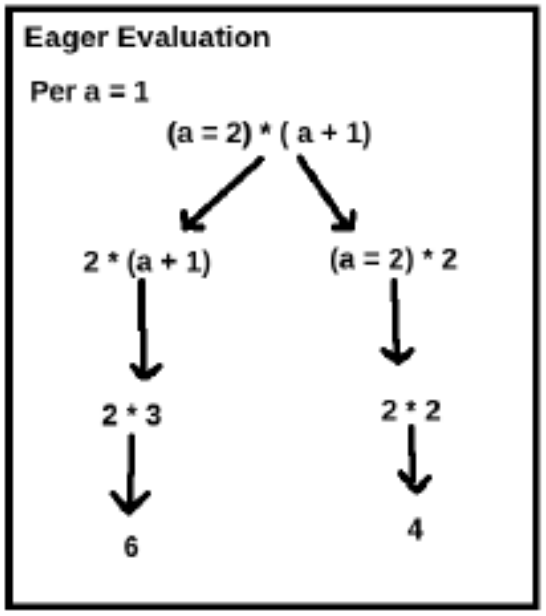


# Cos'è Haskell?

- *Haskell è un linguaggio di programmazione avanzato e puramente funzionale (basato sul sistema formale lambda calcolo) e prende nome dal matematico-logico statunitense Haskell Curry;*
- *Adatto per l'insegnamento, la ricerca, per sviluppare applicazioni e la costruzione di grandi sistemi;*
- *Completamente descritto attraverso la pubblicazione di una sintassi e una semantica formale;*
- *Haskell è un linguaggio per cui la valutazione risulta "lazy";*
- *No effetti collaterali (no side effect) $\Rightarrow$ deterministico e per sua natura è possibile adottare tecniche di parallelismo.*

# Determinismo in Haskell



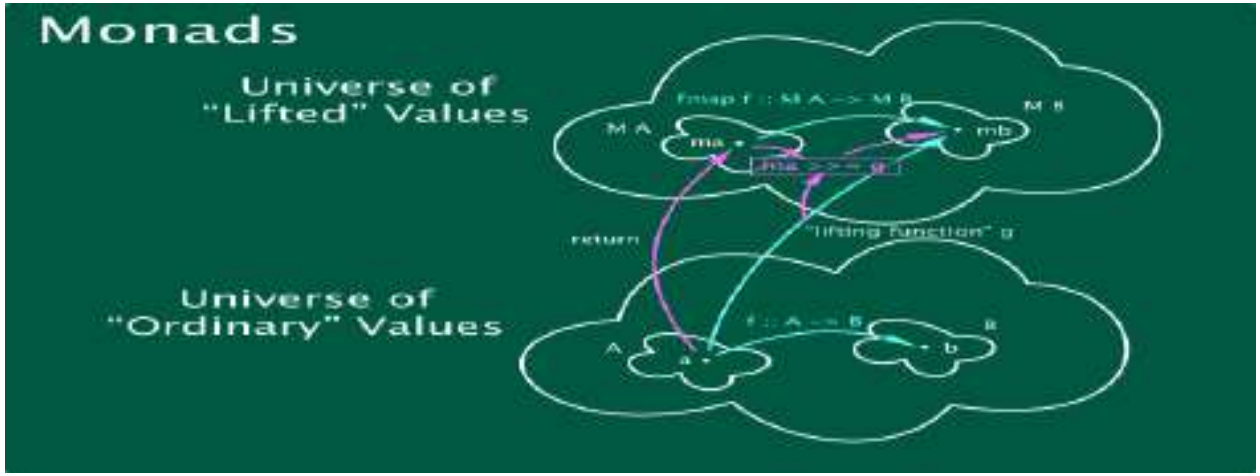

# Monadi in Haskell

- *Le Monadi in Haskell realizzano composizioni di classi di computazioni in modo da poter trasportare dati extra. Questo permette la realizzazione di operazioni come I/O, State, handler, ecc;*
- *Una monade è caratterizzata da:*
  - *Un costruttore di tipo T che crea un tipo monadico T a;*
  - *Un iniettore, spesso chiamato unit o return, che inietta un valore x nella monade;*
  - *Un combinatore, tipicamente chiamato bind e rappresentato con un operatore postfisso (>>=), che scarta il valore prodotto dalla prima azione monadica, e la inserisce in una funzione che farà risultare un nuovo valore monadico.*

# Monadi in Haskell

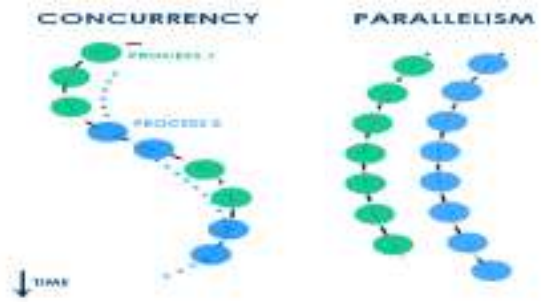
## Monad

```
class Monad m where
        (>>=)  :: m a -> (a -> m b) -> m b
        (>>)   :: m a ->  m b -> m b
        return :: a -> m a
        fail   :: String -> m a
```




# Parallel Haskell

- *Il Parallelismo e la Concorrenza sono concetti distinti:*
  - *Parallelismo significa che diversi thread lavorano sullo stesso task per ridurre un calcolo in forma normale ottimizzando i tempi di risposta ⇒ Determinismo;*
  - *La Concorrenza consiste nell'eseguire task differenti da parte di più thread alternandosi la CPU ⇒ Non Determinismo.*
- *Haskell offre i seguenti strumenti al programmatore per parallelizzare i propri programmi:*
  - *Monade Eval e Strategie;*
  - *Monade Par.*

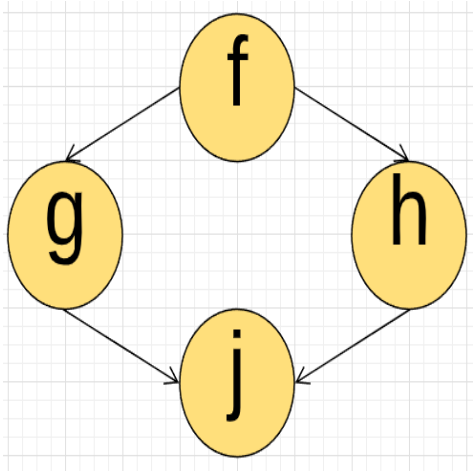


# Parallel Type Inference Algorithm

- *Un esempio che si adatta naturalmente al dataflow model è l'analisi del programma, in cui le informazioni vengono in genere propagate dai punti di definizione ai punti di utilizzo nel programma;*
- *L'inferenza di tipo dà origine a un dataflow graph; ogni binding è un nodo nel grafo con input corrispondente alle variabili libere dell'associazione; il tipo degli identificatori scorrono lungo gli archi e un singolo output rappresenta il tipo derivato per quel binding (ovviamente non si conosce la forma del grafo a priori). Ad esempio, il seguente insieme di bindings può essere rappresentato dal dataflow graph sotto riportato:*

# Parallel Type Inference Algorithm

```
f = ...
g = ... f ...
h = ... f ...
j = ... g ... h ...
```




# Parallel Type Inference Algorithm

*Definiamo la funzione parinfer come segue:*

## parInfer

```
type TypeEnv = Map Var (IVar Type)
infer :: TypeEnv -> (Var,Expr) -> Par ()
parInfer :: [(Var,Expr)] -> [(Var,Type)]
   parInfer bindings  = runPar $ do
   let binders = map fst bindings
   ivars <- replicateM (length binders) new
   let env = Map.fromList (zip binders ivars)
   mapM_ (fork . infer env) bindings
   types <- mapM_ get ivars
   return (zip binders types)
```

# Parallel Type Inference Algorithm: Threadscope

- *In Threadscope:*

Figure: *ParInfer visto in Threadscope su una CPU quad-core con 8-threads.*

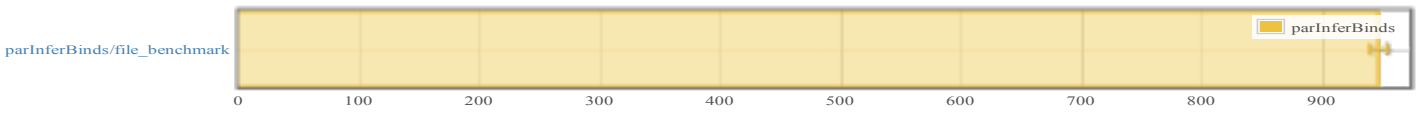
Figure: *ParInfer visto in Threadscope su una CPU quad-core limitata a 2 core con 4 thread*

# Parallel Type Inference Algorithm: Prestazioni con Criterion

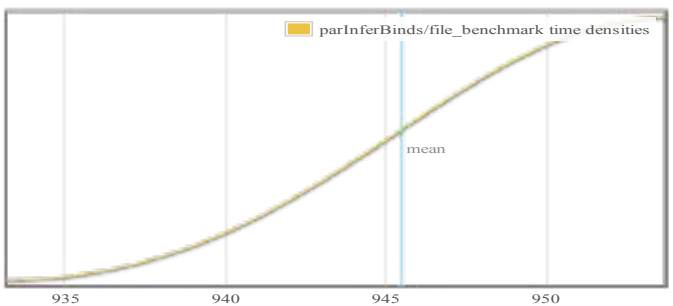
## criterion performance measurements

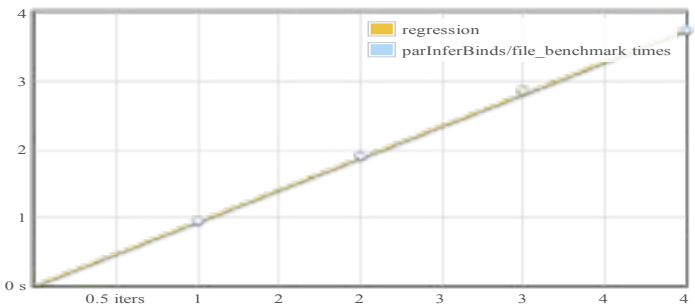
### overview

want to understand this report?




### parInferBinds/file_benchmark




| | lower bound | estimate | upper bound |
|---|---|---|---|
| OLS regression | 884 ms | **934 ms** | 959 ms |
| R² goodness-of-fit | 0.999 | **1.000** | 1.000 |
| Mean execution time | 938 ms | **945 ms** | 950 ms |
| Standard deviation | 1.73 ms | **7.39 ms** | 9.84 ms |

Outlying measurements have moderate (18.8%) effect on estimated standard deviation.

# Ringraziamenti

*GRAZIE A TUTTI PER L'ATTENZIONE*