Strategie di Programmazione Parallela in Haskell

Candidato : Andrea Senese
Relatore : Prof. Ugo de' Liguoro

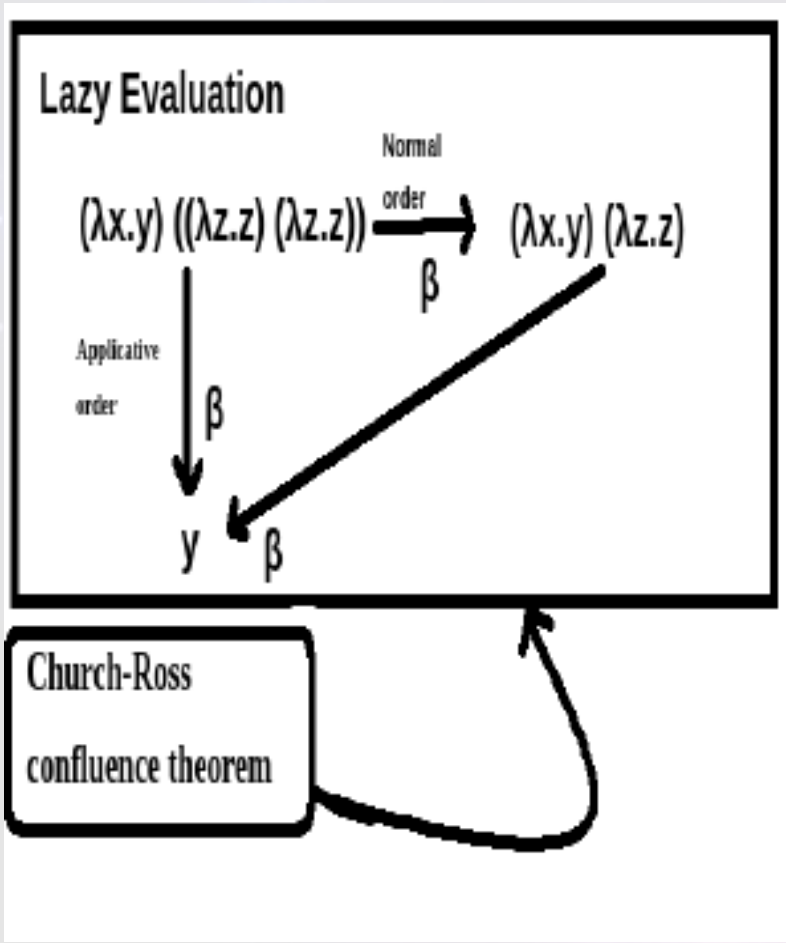
Università degli Studi di Torino
Scuola di Scienze Matematiche, Fisiche e Naturali
Dipartimento di Informatica
Laurea Triennale in Informatica - Linguaggi e Sistemi

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



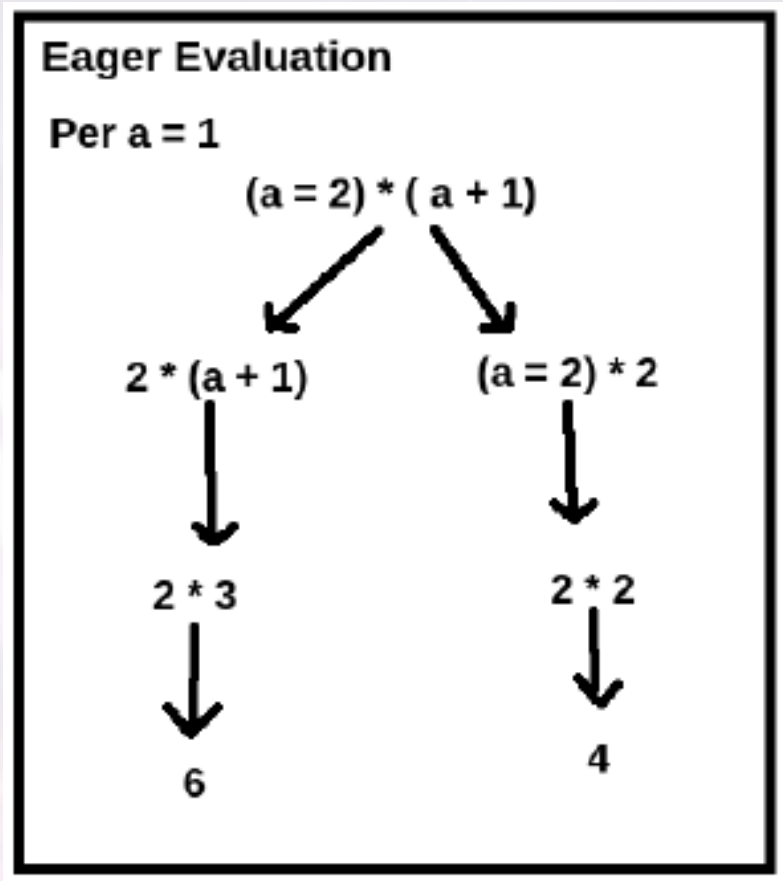


Figure 1: *Lazy Evaluation vs Eager Evaluation*

Determinismo

- *Dato un termine M che si riduce in uno o più passi di calcolo in N_1 e N_2, allora esiste un N tale che N_1 si riduce a N (in uno o più passi di calcolo) e N_2 si riduce a N (in uno o più passi di calcolo). Formalmente:*

Theorem (Church-Ross Confluence)

Se $M \to N_1$ e $M \to N_2$ allora $\exists N$ tale che $N_1 \to N$ e $N_2 \to N$

Dimostrazione:
Supponiamo che N_1 e N_2 sono due forme normali di M, allora per il teorema di confluenza sappiamo che N_1 si riduce a N (in 0 passi) e lo stesso N_2 e quindi per transitività abbiamo che:

$$(N_1 = N \wedge N_2 = N) \Rightarrow N_1 = N_2$$

Monadi per introdurre Non Determinismo(side effect)

- *Le Monadi in Haskell possono essere pensate come descrizioni computabili di calcoli;*
- *Monade $\Rightarrow$ composizione di classi di computazioni in modo da poter trasportare dati extra. Questo permette la realizzazione di operazioni come I/O, State, handler, ...;*
- *Una monade è caratterizzata da:*
  - *T la tipologia del costrutto $\Rightarrow$ equivale al costruttore di tipo M;*
  - *μ è una funzione che permette la concatenazione o la composizione di monadi della stessa tipologia $\Rightarrow$ equivale al bind (>>=);*
  - *η è una funzione che permette il passaggio da (a $\rightarrow$ T a) $\Rightarrow$ equivale al return.*

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
```

Monadi per introdurre Non Determinismo(side effect)

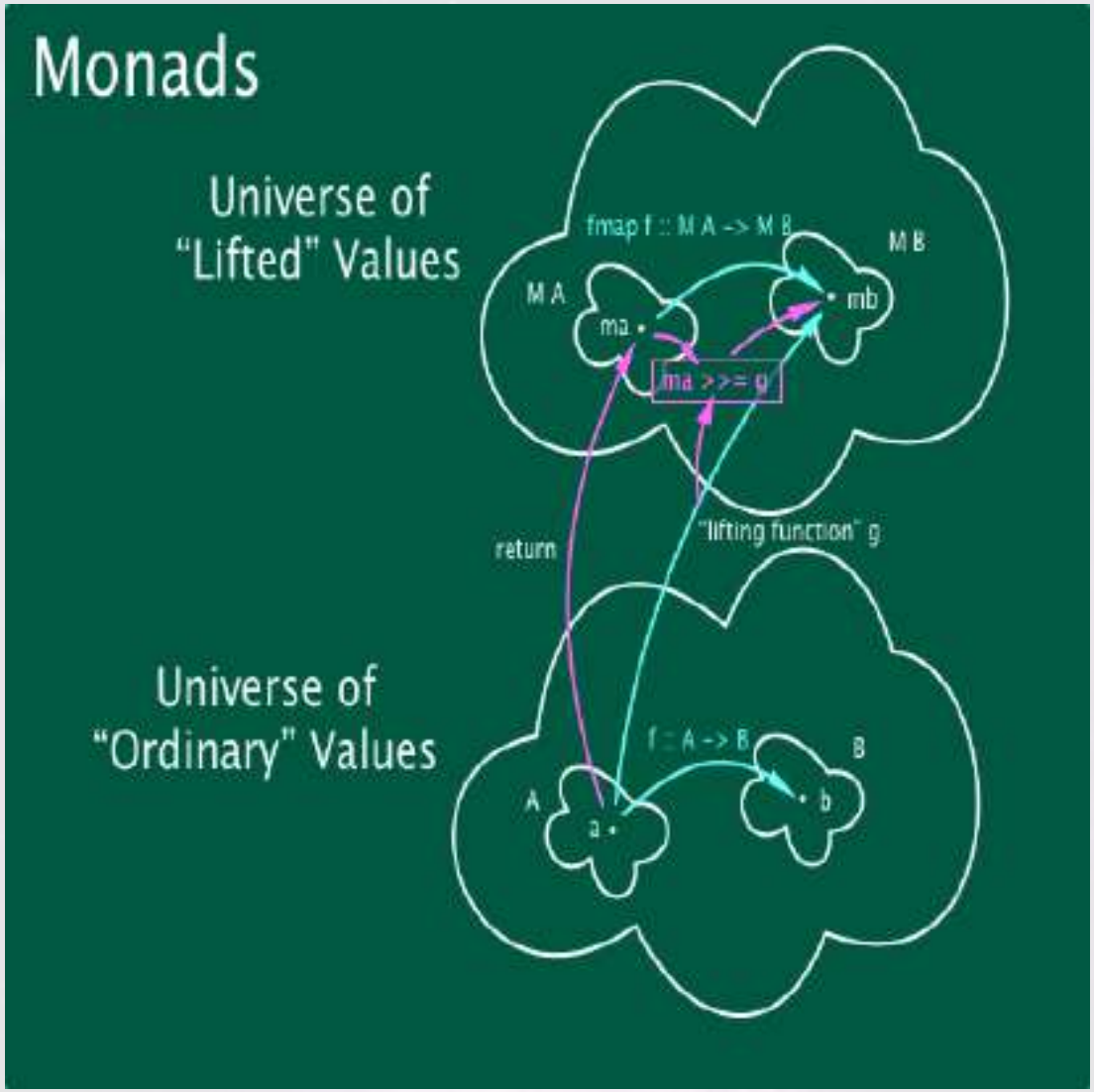


Figure 2: *Monade*

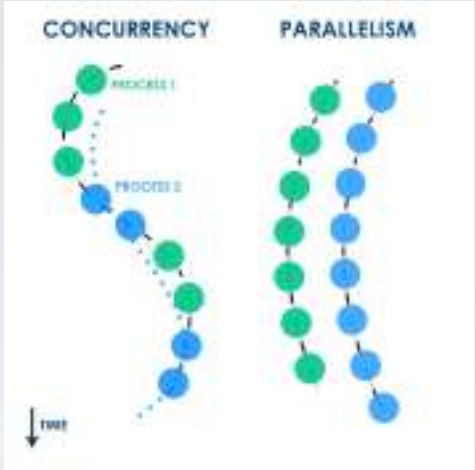


Figure 3: *Parallelismo ≠ Concorrenza*

- *Il Parallelismo e la Concorrenza sono concetti distinti:*
  - *Parallelism 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.*

Monade Eval

- *Monade Eval è il primo approccio per esprimere il Parallelismo in Haskell dal modulo Control.Parallel.Strategies. Si hanno 3 operatori base nella Monade Eval:*
  - *runEval → Computa l'operazione nell'Eval Monad e ritorna il risultato;*
  - *rpar → L'argomento potrebbe essere valutato in parallelo;*
  - *rseq → Forza la valutazione sequenziale e si mette in attesa del risultato.*
- *Per esempio :*

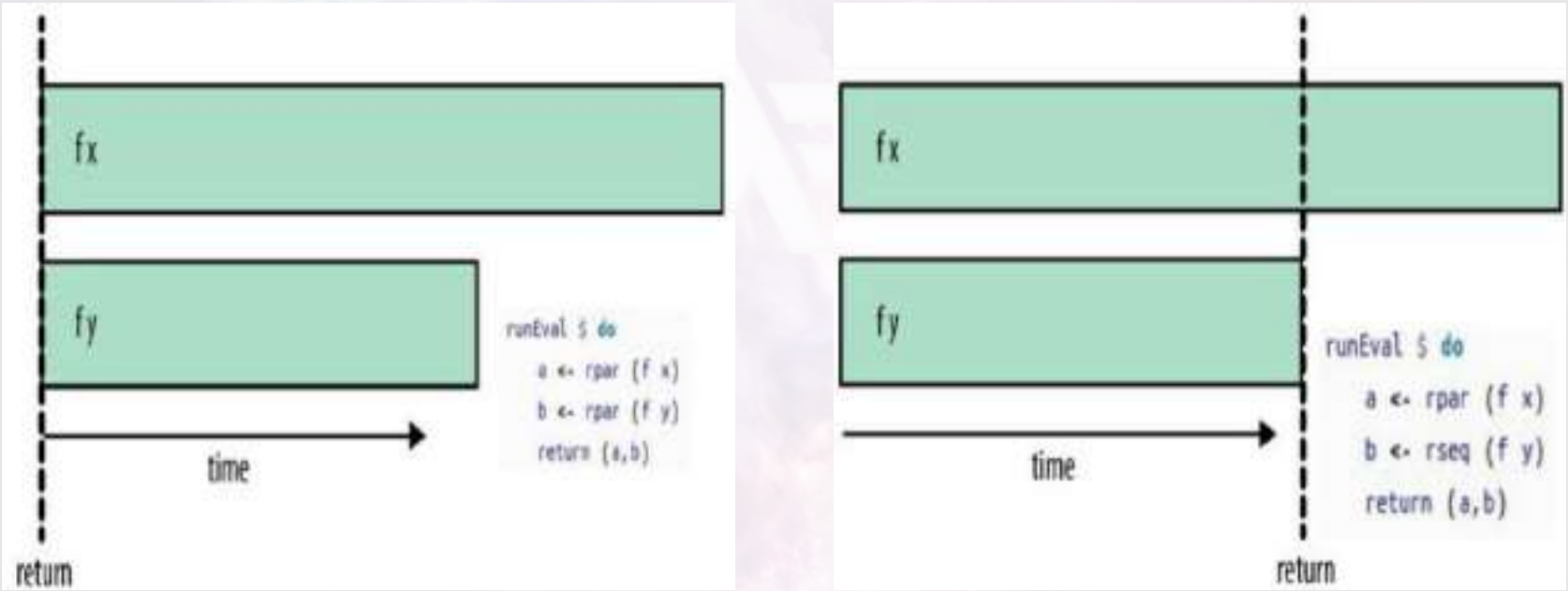


Figure 4: *rpar/rpar e rpar/rseq*

Monade Eval

- *La soluzione è:*

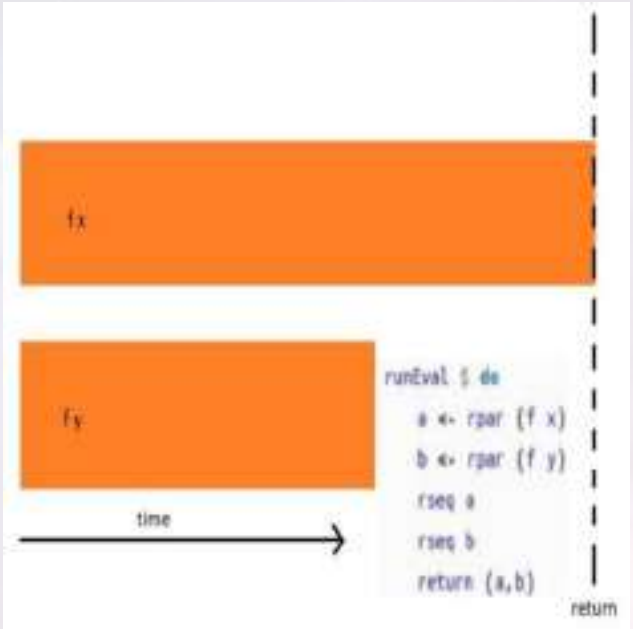


Figure 5: *rpar/rpar/rseq/rseq*

- *La Monade Eval è integrata in un programma di diagnostica chiamato Threadscope.*

Strategie

- *Esiste una relazione stretta tra Monade Eval e Strategie. Le Strategie possono essere pensate come un'astrazione di livello superiore per alleviare molti dei problemi associati alla regolazione della granularità e alla forzatura della valutazione manuale; Le Strategie hanno le seguenti caratteristiche:*
  - *Le Strategie esprimono parallelismo deterministico: il risultato del programma non è influenzato dalla valutazione in parallelo. I task parallel valutatati con le Strategie non hanno side effect;*
  - *Le Strategie consentono di separare il parallelismo dalla logica del programma, permettendo parallelismo modulare. L'idea di base è quella di costruire una struttura dati lazy che rappresenti il calcolo, e quindi scrivere una strategia che descriva come attraversare la struttura dei dati e valutarne i componenti in sequenza o in parallelo;*
  - *Le Strategie sono composizionali: grandi strategie possono essere costruite componendo strategie più piccole.*

Threadscope

- *Threadscope venne creato per aiutare i programmatori nel visualizzare l'esecuzione parallela dei programmi Haskell compilati con GHC. L'idea è che quando viene eseguito un programma parallelo in Haskell, il sistema di runtime di Haskell scrive data e tempistiche di eventi significativi in un log file. Il tool Threadscope allora leggerà il log file, e mostrerà graficamente all'utente cosa stava facendo la CPU in quell'intervallo di tempo. Il diagramma mostra al programmatore in quali istanti il programma viene eseguito in parallelo, e dove no.*

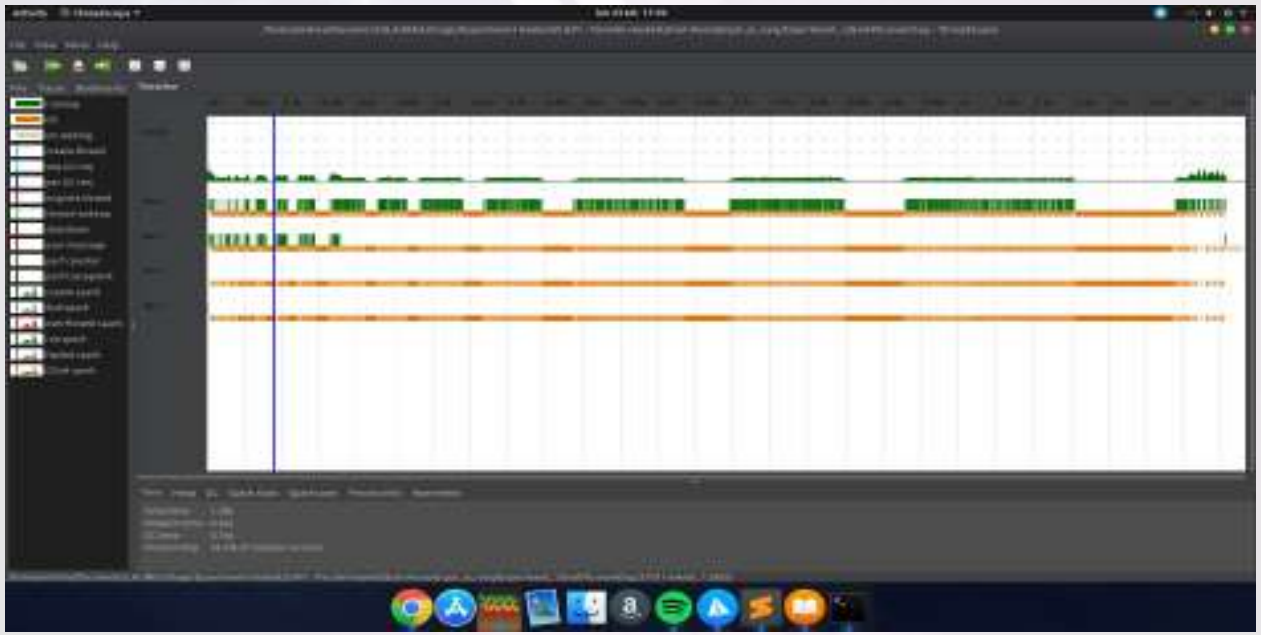


Figure 6: il tool Threadscope

Monade Par

- *Gli strumenti menzionati finora portano a tassi di fallimento elevati perché rpar richiede al programmatore di comprendere le proprietà operative del programma che risiedono, nella maggior parte dei casi, nell'implementazione;*
- *Un'altra pratica utilizzata per correggere le loro debolezze è la Monade Eval, che purtroppo è ancora limitata (e non va molto lontana);*
- *La Monade Par offre le seguenti operazioni di base:*
  - *runPar* $\rightarrow$ *computa l'operazione nella Monade Par e ritorna il risultato;*
  - *fork* $\rightarrow$ *crea un nuovo Thread;*
  - *new* $\rightarrow$ *create un nuovo IVar;*
  - *get* $\rightarrow$ *get the result written in the IVar;*
  - *put* $\rightarrow$ *writes the result in the IVar.*

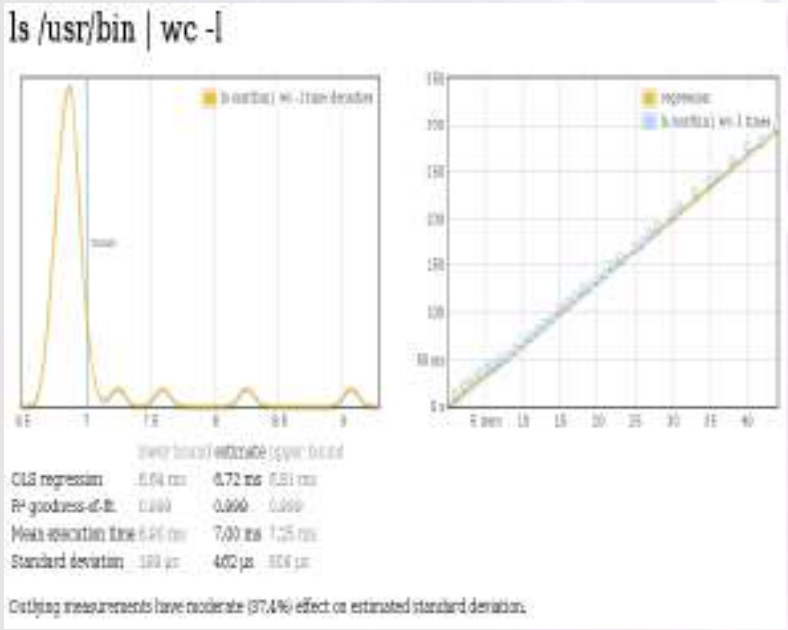
IVar ☠

L'IVar è una variabile immutabile che è possibile scrivere una sola volta e poi verrà bloccata (per garantire determinismo).

La Monade Par non è attualmente integrata in Threadscope.

Criterion

- *Criterion è un libreria che fornisce un modo semplice per misurare le prestazioni del software tramite definizione di benchmark;*
- *il flag "–option nomefile.html" indica al nostro programma di scrivere un report nel file HTML;*
- *Si ha un elenco di valori di benchmark dove ognuno corrisponde a una funzione da esaminare.*




```haskell
import Criterion.Main

-- The function we're benchmarking.
fib m | m < 0     = error "negative!"
      | otherwise = go m
  where
    go 0 = 0
    go 1 = 1
    go n = go (n-1) + go (n-2)

-- Our benchmark harness.
main = defaultMain [
  bgroup "fib" [ bench "1"  $ whnf fib 1
               , bench "5"  $ whnf fib 5
               , bench "9"  $ whnf fib 9
               , bench "11" $ whnf fib 11
               ]
  ]
```

Figure 7: *Test Criterion + Report*

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 e un singolo output rappresenta il tipo derivato per quel binding. Ad esempio, il seguente insieme di bindings può essere rappresentato dal dataflow graph sotto riportato:*

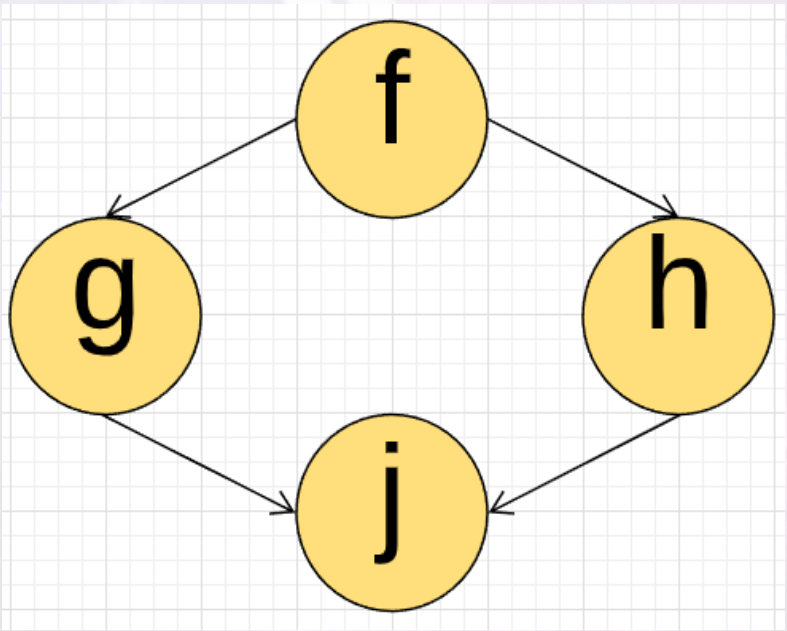


Figure 8: *Dataflow graph of bindings*

Parallel Type Inference Algorithm

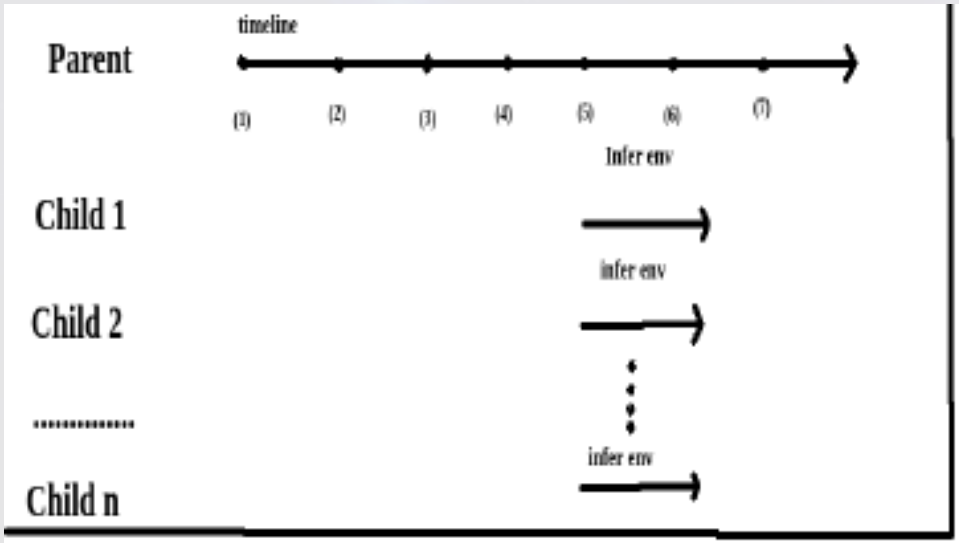
Definiamo la funzione infer come segue:

parInfer

```
--(1)
parInfer :: [(Var,Expr)] -> [(Var,Type)]
parInfer bindings  = runPar $ do
                let binders = map fst bindings  --(2)
                ivars <- replicateM (length binders) new --(3)
                let Env = Map.fromList (zip binders ivars) --(4)
                mapM_ (fork . infer env) bindings --(5)
                types <- mapM_ get ivars --(6)
                return (zip binders types) --(7)
```

Parallel Type Inference Algorithm

Un esecuzione concreta :



(1) call parInfer -> parInfer [(var1,exp1),(var2,exp1),...,(var_n,exp_n)]

(2) let binders = map fst [(var1,exp1),(var2,exp1),...,(var_n,exp_1)] = [var1,var2,...,var_n] (forall elem in list apply fst(elem))

(3) ivars = replicateM (length [(var1,exp1),(var2,exp1),...,(varn,exp1)]) new = replicateM n new = [Ivar1,Ivar2,.., Ivar_n]
(create a list of length n with n Ivar)

(4) env = Map.fromList (zip [var1,var2,...,var_n] [Ivar1,Ivar2,...,Ivar_n]) = Map.fromList [(var1,Ivar1),(var2,Ivar2),...,(var_n,Ivar_n)] =

	Key	Value
1	var1	Ivar1
...	...	...
n	var_n	Ivar_n

(5) Infer calls put for each binding in the list for write the type in the Ivar mapped by that variable in env.

(6) type = mapM_ get [Ivar1,...,Ivar_n] = [type1,type2,...,type_n] (wait the result computated by the childs).

(7) return (zip [var1,var2,...,var_n] [type1,type2,...,type_n]) =[(var1,type1),...,(var_n,type_n),] (return the result)

Parallel Type Inference Algorithm

- *Questa implementazione estrae il massimo parallelismo nella struttura di dipendenza di un determinato insieme di bindings, con pochissimo sforzo da parte del programmatore;*
- *Lo stesso trucco può essere fatto usando la valutazione lazy, e in effetti potremmo ottenere la stessa struttura parallela usando la valutazione lazy insieme alle Strategie, ma il vantaggio della versione Par monad è che la struttura è programmata esplicitamente e il comportamento a runtime non è legato a una particolare strategia di valutazione (o compilatore).*

Parallel Type Inference Algorithm : Threadscope

- *In Threascope:*

Figure 9: *ParInfer visto in Threadscope su una CPU quad-core con 8-threads.*

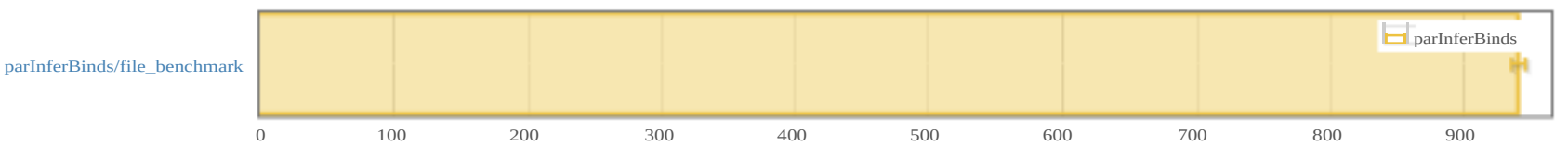
Figure 10: *ParInfer visto in Threadscope su una CPU quad-core limitato a 2 core*

Parallel Type Inference Algorithm : Prestazioni con Criterion

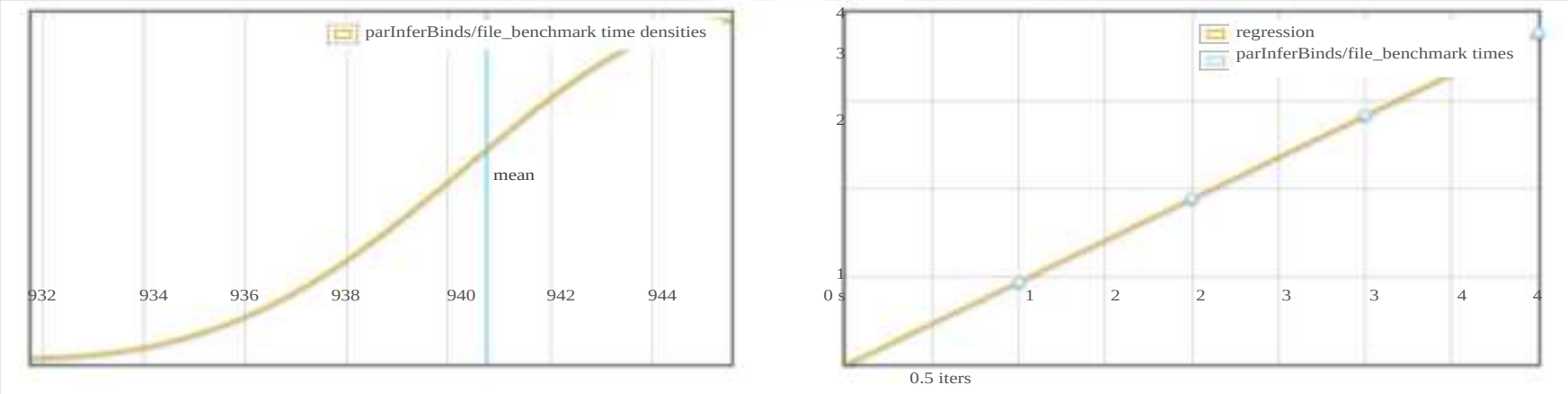
criterion performance measurements

overview

want to understand this report?



parInferBind/file_benchmark



	lower bound	estimate	upper bound
OLS regression	936 ms	945 ms	954 ms
R^2 goodness-of-fit	1.000	1.000	1.000
Mean execution time	933 ms	941 ms	944 ms
Standard deviation	768 µs	5.29 ms	6.62 ms

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

Fine

GRAZIE A TUTTI PER L'ATTENZIONE