3: Functional paradigm (III)

Programming Languages, Technologies and Paradigms

Escuela Técnica Superior de Ingeniería Informática
etsinf

Summary

Introduction to Functional Programming

PART I: Types in Functional Programming

1. Functional types. Algebraic types.
2. Predefined types.
3. Polymorphism: genericity, overloading and coercion. Inheritance in Haskell.

PART II: Models of computation in functional programming.

4. Operational model.

PART III: Advanced features

5. Anonymous functions and composition.
6. Iterators and compressors (foldl, foldr).

Summary

Anonymous functions

- Anonymous (or nameless) functions.
  - In Haskell we can define anonymous functions of the form \ x -> e

  Example: The function square x = x*x can be defined in this way as follows: square = (\ x -> x*x)
  - In general, \x1 x2… xn -> e is equivalent to

  $\backslash x_1$-> ($\backslash x_2$-> (··· -> ($\backslash x_n$ -> e) ···))

  Example:

  sumOfSquares = \x y -> x*x + y*y

  is equivalent to:

  sumOfSquares = \x -> (\y -> x*x + y*y)

Function composition

- Function composition
  - (.) :: (b -> c) -> (a -> b) -> a -> c

    Higher-order

    polymorphic

    (f . g) x = f (g x)
  - Function composition is defined in the **Haskell prelude** as follows:

    (f . g) = \ x -> f (g x)
  - Function composition is a frequent computation pattern. The solution of a problem consists of several steps each of which can be independently addressed by using independent functions that can be then *composed* to solve the problem.

Function composition

Example:

```
twice f x = (f . f ) x
```

point-wise

Equivalently,

```
twice f = f . f
```

point-free

in lambda notation:

```
twice = \ f x -> f (f x)
```

Summary

Introduction to Functional Programming

PART I: Types in Functional Programming

1. Functional types. Algebraic types.
2. Predefined types.
3. Polymorphism: genericity, overloading and coercion. Inheritance in Haskell.

PART II: Models of computation in functional programming.

4. Operational model.

PART III: Advanced features

5. Anonymous functions and composition.
6. Iterators and compressors (foldl, foldr).

Iterators and compressors

- Iterators
  - Iterators can be used to save memory and time when dealing with *iterable types* like lists or sequences.
  - iterate f x returns an infinite list of repeated applications of f to x: iterate f x is [x, f x, f (f x), ...]

```
iterate      :: (a -> a) -> a -> [a]
iterate f x   =  x : iterate f (f x)
```

Example: The Haskell prelude function from is defined by:

```
from = iterate (1+)
```

Iterators and compressors

- Compressors

Many functions over lists follow a recursive scheme

```
f :: [a] -> b
f [] = z
f (x:xs) = x ⊗ f xs
```

transforming a list x_1:(x_2:(x_3:(x_4:[]))) into $x_1 \otimes (x_2 \otimes (x_3 \otimes (x_4 \otimes z)))$

Example:

```
sum :: [Int] -> Int
sum [] = 0
sum (x:xs) = x + sum xs

product :: [Int] -> Int
product [] = 1
product (x:xs) = x * product xs
```

Iterators and compressors

- We can introduce a function “foldr” that implements this kind of transformation:

```
foldr :: (a -> b -> b) -> b -> [a] -> b
foldr op z [] = z
foldr op z (x:xs) = x `op` (foldr `op` z xs)
```

- The previously considered function f can be just define as follows f = foldr (⊗) z

  And similarly for specific functions like:
  - sum = foldr (+) 0
  - product = foldr (*) 1

Iterators and compressors

- Example: the function sum in a previous example (*length of a path*) can be given by using foldr or foldl

```
sum :: [Float] -> Float
sum  = foldr (+) 0.0
```

- Exercise: Define concat, and, or, and map by using foldr.

The Mapreduce scheme

- The combined use of optimized functions like map and fold inspired an efficient style of sequence processing, the *MapReduce* scheme, which has been successfully used in massive dataprocessing (> 1Tb), with thousands of processors and around 100.000 HDs.
- The functional scheme MapReduce was promoted by Google and has a number of relevant applications:
  - Information retrieval
  - cloud computing (computation services delivered by companies like Google, Yahoo!, etc., to external clients
  - …

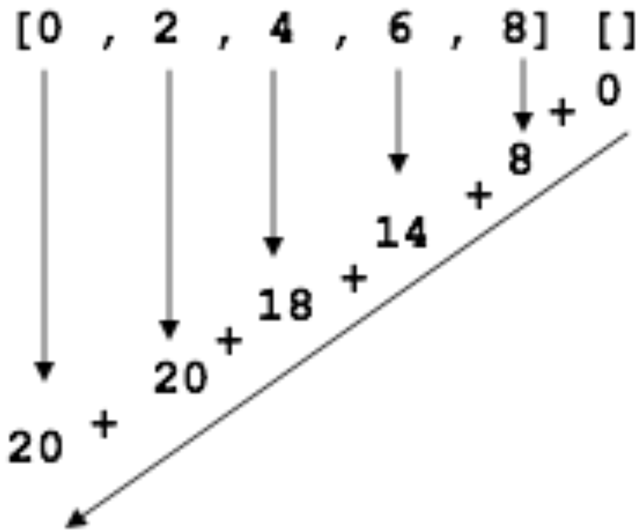
Use of Mapreduce

- In **Google:**
  - Building indices for Google Search
  - Classifying notices for Google News
  - Automatic translation
- In **Yahoo!:**
  - Yahoo! Search
  - Spam detection in Yahoo! Mail
- In **Facebook:**
  - Data mining
  - Optimization of publicity
  - Spam detection

Use of Mapreduce

The inspiration:

sumList = foldr (+) 0

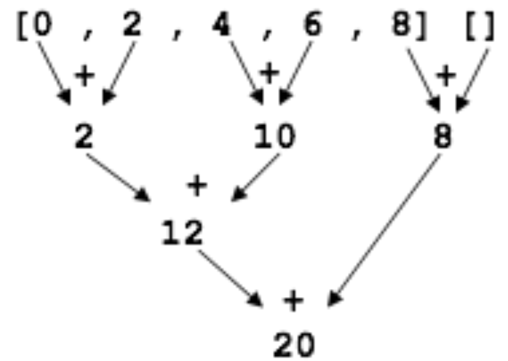


- The computation proceeds from left to right; the number of steps is equal to the length of the list
- *But (+) is associative and commutative!*
  -> ***we can (automatically) parallelize the process…***

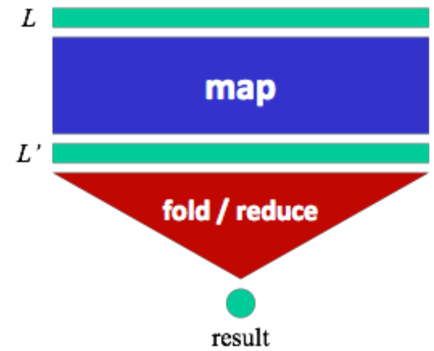
…and distribute the workload over hundreds/thousands processors!

Use of Mapreduce

- The cost becomes **O(log n)** if we proceed as follows:

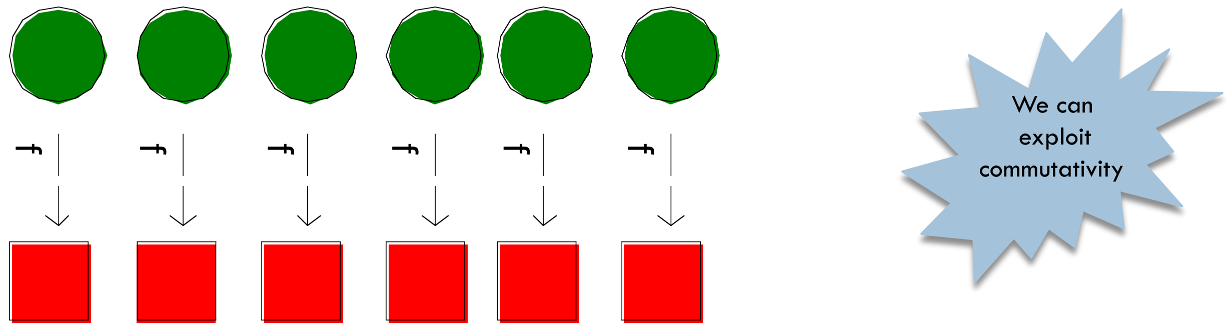


- We can do it by appropriately combining map and fold.

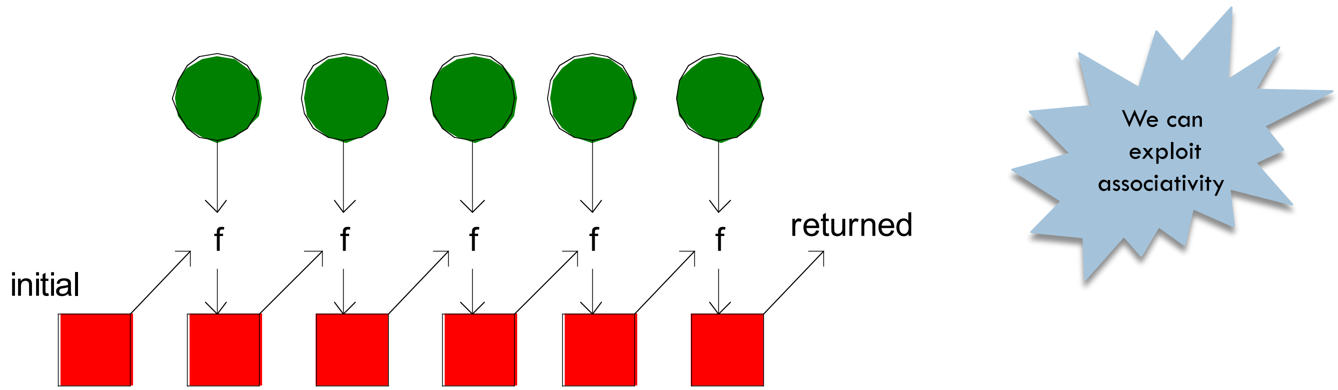


- We can generalize idea:
this is the "secret" of the MapReduce scheme

- map f: build a new list by applying f to the input list



- fold f z xs: applies f to the elements of a list and carries an *accumulator*.
  The function f *returns the new value of the accumulador, which is combined with the next element of the list*



Use of Mapreduce

- The MapReduce scheme is a useful abstraction that simplifies and optimizes heavy computations
- MapReduce has inspired the design of libraries for other languages:
  - There is now a C++ library MapReduce where map() is divided into 64 MB blocks (of the same size that the chunks of Google's File System).
  - There is a similar library for Java.
  - Advantages: we can focus on the problem, and leave the management details (organization, keys, access, etc.) to the library.

Bibliography

- BASIC

  - Bird, R. Introducción a la programación funcional con Haskell, Prentice-Hall, 2000. Traducción de Ricardo Peña.
  - Ruiz, B.C.; Gutiérrez, F.; Guerrero, P.; Gallardo, J.E. Razonando con Haskell, Thomson Editores, 2004.

- HASKELL

  - Lipovaca, M. Learn You a Haskell for Great Good!: A Beginner's Guide. http://learnyouahaskell.com/
  - O'Sullivan B., Goerzen, J, and Stewart D. Real world Haskell, O'Reilly, 2008.