Functional Programming
Using Haskell

Bruno Dias da Gião

December 8, 2022

Contents

How does a function work?

In order to ease understanding let us consider the following:

In order to ease understanding let us consider the following:

Definition

A sequence shall be “anything” that has a base case and every other element can be obtained from it using a rule.

In order to ease understanding let us consider the following:

Definition

A sequence shall be "anything" that has a base case and every other element can be obtained from it using a rule.
In mathematics we call this an inductively defined set.

In order to ease understanding let us consider the following:

Definition

A sequence shall be "anything" that has a base case and every other element can be obtained from it using a rule.
In mathematics we call this an inductively defined set.

For the sake of simplicity we shall only be preocupying ourself with defining functions over these sorts of sequences.

One case of such sequences are the Natural Numbers.

One case of such sequences are the Natural Numbers.

Definition (Natural Numbers)

1. $0 \in \mathbb{N}_0$
2. $n \in \mathbb{N}_0 \implies n + 1 \in \mathbb{N}_0$

One case of such sequences are the Natural Numbers.

Definition (Natural Numbers)

1. $0 \in \mathbb{N}_0$
2. $n \in \mathbb{N}_0 \implies n + 1 \in \mathbb{N}_0$

What this tells us is that Natural numbers can either be zero or the successor of a natural number!

Another form of sequence is the well known data structure - list:

Another form of sequence is the well known data structure - list:

Definition (Lists)

1. nil = [] belongs to Lists
2. if list belongs to Lists then "cons elem list = elem : list" belongs to Lists

Another form of sequence is the well known data structure - list:

Definition (Lists)

1. nil = [] belongs to Lists
2. if list belongs to Lists then "cons elem list = elem : list" belongs to Lists

Where nil is the function that creates an empty list and cons is the function that adds an element to the beginning of a list.
Haskell uses : to represent adding an element to the head of a list.

Are words, sentences and texts sequences?

Let us try to “define” these structures:

Let us try to "define" these structures:

Definition (String)

1. "" belongs to Strings
2. if string belongs to Strings then letter:string belongs to Strings

Let us try to "define" these structures:

Definition (String)

1. "" belongs to Strings
2. if string belongs to Strings then letter:string belongs to Strings

The keen reader may notice how this definition is quite literally the same as the definition of a List!

Let us try to "define" these structures:

Definition (String)

1. "" belongs to Strings
2. if string belongs to Strings then letter:string belongs to Strings

The keen reader may notice how this definition is quite literally the same as the definition of a List!

Theorem

A String is no more no less than a List of Characters!

Let us try to "define" these structures:

DEFINITION (STRING)

1. "" belongs to Strings
2. if string belongs to Strings then letter:string belongs to Strings

The keen reader may notice how this definition is quite literally the same as the definition of a List!

THEOREM

A String is no more no less than a List of Characters!

COROLLARY

Any function applied to generic Lists can be applied to Strings and vice versa.