

Report on OGO 2.2 Softwarespecification assignment 1a

Femke Jansen and Lasse Blaauwbroek OGO 2.2 group 6
Department of Computer Science
Technical University Eindhoven

February 7, 2012

Abstract

This document presents a formal specification and implementation of assignment 1a of OGO 2.2 Softwarespecification. Also included are the informal requirement of the stakeholder, their judgement about the formal specification and testcases for the implementation.

1 Informal requirements

Input A simple textfile consisting of real numbers separated with whitespace.

Output Given is a sequence of numbers from which some numbers needs to be summed. The indices of the numbers that need to be added are given by the Fibonacci sequence. The output is the result of the sum in flat text.

Example Input: -5 43 25 -83 67 -97 73 14 -58 25 84 29 81 -54.
The input has 14 elements, so the following part of the fibonacci sequence is used: $\{0, 1, 1, 2, 3, 5, 8, 13, 21, \dots\}$. Using the sequence as indices of the numbers on input, we get the following numbers that need to be summed: -5, 43, 43, 25, -83, -97, -58, -54.
The summation of these numbers is -186.

2 Formal specification

This section describes a formal specification adhering to the informal requirements described above.

We first have to define a function *fib* that returns the *n*'th Fibonacci number. Fibonacci numbers are recursively defined with the two previous Fibonacci numbers.

$$\begin{array}{|l} \hline fib : \mathbb{N} \rightarrow \mathbb{N} \\ \hline fib(0) = 0 \\ fib(1) = 1 \\ \forall n : \mathbb{N} \mid n \geq 2 \bullet fib(n) = fib(n-2) + fib(n-1) \end{array}$$

Next, we also need a function *sum* that returns the summation of a given sequence of numbers. This function is recursively defined as well, by computing the sum of the tail of the sequence until the sequence is empty, and adding all of the heads to this sum.

$$\begin{array}{|l} \hline sum : \mathbb{P}(\text{seq } \mathbb{R} \rightarrow \mathbb{R}) \\ \hline \forall (x, y) : sum \mid x = \langle \rangle \bullet y = 0 \\ \forall (x, y) : sum \mid x \neq \langle \rangle \bullet y = sum(tail\ x) + head\ x \end{array}$$

Last but not least, we need a method that selects the numbers with Fibonacci indices from a sequence and places those numbers in a new sequence. This is a bit tricky, because sequences begin numbering with one, and the Fibonacci sequence starts with zero. Therefore, we increment each Fibonacci number with one, and decrement each sequence index by one to get a correct mapping from Fibonacci numbers to indices.

$$\begin{array}{|l} \hline selectfibindices : \mathbb{P}(\text{seq } \mathbb{R} \rightarrow \text{seq } \mathbb{R}) \\ \hline \forall (x, y) : selectfibindices; z : \mathbb{N} \mid 1 \leq z \wedge z \leq \#y \bullet \\ fib(z-1) + 1 \leq \#x \wedge y(z) = x(fib(z-1) + 1) \end{array}$$

The real algorithm to take the summation of all numbers with a Fibonacci index is now very straightforward and given below.

$\begin{array}{ l} \hline \textit{Algorithm} \\ \hline input? : \text{seq } \mathbb{R} \\ output! : \mathbb{R} \\ \hline output! = sum(selectfibindices(input?)) \end{array}$

3 Judgement of the formal specification

4 Testcases for the implementation

5 Implementation

We chose to implement the algorithm in the Java programming language, because everyone in our group is familiar with it. The method implementing the algorithm is shown below and requires an import for *java.io.InputStream* and *java.io.Scanner*:

```
/**
 * This method is an algorithm that gives the sum of
 * the (real) numbers in a stream. But only the numbers
 * with indices equal to a fibonacci number are used.
 *
 * @param stream The stream to take the numbers from
 * @return The sum of the elements in the stream with
 *         indices equal to a fibonacci number
 */
public static double fibAlgo(InputStream stream) {
    Scanner scanner = new Scanner(stream);
    int secondOldFib = 0;
    int firstOldFib = 1;
    int fib = 0;
    int index = 0;
    double current;
    double sum = 0;
    while (scanner.hasNextDouble()) {
        current = scanner.nextDouble();
        if (index == fib) {
            sum += current;
            if (index == 1) {
                sum += current;
            }
        }
    }
}
```

```
        fib = secondOldFib + firstOldFib;
        secondOldFib = firstOldFib;
        firstOldFib = fib;
    }
    index++;
}
return sum;
}
```