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 8, 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: $0, 1, 1, 2, 3, 5, 8, 13, 21, ...$ 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 using the two previous Fibonacci numbers.

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

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.

$$
\begin{array}{|l}
\mathit{sum} : \mathbb{P}(\operatorname{seq}\mathbb{R} \to \mathbb{R}) \\
\hline
\mathit{sum}(\langle\rangle) = 0 \\
\forall (x) : \operatorname{seq}\mathbb{R} \mid \#x > 0 \bullet \mathit{sum}(x) = \mathit{sum}(\mathit{tail}\, x) + \mathit{head}\, x
\end{array}
$$

Last but not least, we need a method that filters certain numbers from a given sequence and places those numbers in a new sequence. This filtering method selects those numbers from the given sequence with an index equal to a Fibonacci number. The idea is that we can use the new sequence by summing it to calculate the output. The details are 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. The z variable in the specification below filters the right elements from the given sequence. It realizes this by ensuring that the z'th element in the new sequence is equal to the element in the first sequence with as index the z'th Fibonacci number.

$$
\begin{array}{|l}
\mathit{filterFibIndices} : \mathbb{P}(\operatorname{seq}\mathbb{R} \to \operatorname{seq}\mathbb{R}) \\
\hline
\forall (x, y) : \mathit{filterFibIndices};\ z : \mathbb{N} \mid \\
\qquad 1 \leq z \land z \leq \#y \land \mathit{fib}(z-1) + 1 \leq \#x \bullet \\
\qquad y(z) = x(\mathit{fib}(z-1) + 1)
\end{array}
$$

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

sumFibIndices

$input? : \text{seq}\,\mathbb{R}$

$output! : \mathbb{R}$

$output! = sum(filterFibIndices(input?))$

3 Judgement of the formal specification

4 Testcases

Null testcase This case checks whether an empty input creates the correct output.
Input: {}
Expected output: 0 (test succeeded).

Double one testcase This case checks whether the number on index one is summed twice.
Input: {0 1}
Expected output: 2 (test succeeded)

Negative numbers This case checks whether negative numbers are summed correctly.
Input: {-3 -2 -5 -8 -10}
Expected output: -20 (test succeeded)

Large mixed list This case check whether a longer list with mixed positive, negative and non-integer numbers are summed correctly.
Input {-800 2,85 -85,15 -7482,2654 87 0,002 -0,008 784 -235 12}
Expected output: -8596,7134 (test succeeded)

A Implementation

We chose to implement the algorithm in the Java programming language, because everyone in our group is familiar with it. The implementation is very different from the formal specification, and consist of one loop trough all input elements. A variable keeps track of the index of the current element, and variable keeps track of the current Fibonacci number. As soon as the index is equal to the current Fibonacci number, the element is added to a sum variable, and the next Fibonacci number is computed. All testcases are successfully tested (see previous sections). 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);
    // Fibonacci is repeatedly computed using older
    // fibonacci numbers (this is more efficient than
    // calculating each fibonacci number from scratch)
    int secondOldFib = 0;
    int firstOldFib = 1;
    int fib = 0;
    double current;
    // Index represents the index of the double in the
    // stream
    int index = 0;
    double sum = 0;
    while (scanner.hasNextDouble()) {
        current = scanner.nextDouble();
        // When the index equals to the fibonacci
```

```
            // number, we add it to the sum
            if (index == fib) {
                sum += current;
                // Index one is an exception because it
                // occurs twice in the fibonacci sequence
                if (index == 1) {
                    sum += current;
                }
                // Calculate the new fibonacci number and
                // store the old numbers
                fib = secondOldFib + firstOldFib;
                secondOldFib = firstOldFib;
                firstOldFib = fib;
            }
            index++;
        }
        return sum;
    }
```