Report on OGO 2.2 Softwarespecification assignment 1b

Tim van Dalen, Tony Nan and Ferry Timmers
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 1b 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 Provided is a simple (ASCII) file with a few numbers separated by a space of arbitrary size. There is no limit to the size of the file.

Output Take all unique numbers from the file in the form n^3 with n a natural number and select the lower median. The lower median of a array of numbers is the number x that separate the upper helft of the numbers from the lower half of the numbers. For all numbers in the upper half holds that they are greater then x and for all numbers in the lower half holds that they are smaller then x. Furthermore the number of elements in the upper half subtracted from the number of elements in the lower half is the number null or one.

2 Formal specification

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

We first have to define an enumeration *Cubes* that contains all sets of natural numbers for which each element is a cube of a natural number.

$$Cubes == \{x : \mathbb{P}\,\mathbb{N} \mid \forall\, y : x \mid \exists\, z : \mathbb{N} \mid y^3 = z\}$$

We define a function that calculates the lower median of a set of natural numbers. It defines two subsets that are the two halves of the input. The output is the number that is wedged between the two halves when they are in the natural order.

$$\begin{array}{|l}
\textit{Median} \\
\hline
s? : \mathbb{P}\,\mathbb{N} \\
m! : \mathbb{N} \\
\hline
\exists\, a, b : \mathbb{P}\, s? \mid [\ a \cup b \cup m! \\
\qquad\qquad a \cap b = \\
\qquad\qquad !m \notin (a \cup b) \\
\qquad\qquad \#a - \#b = 0 \lor \mid \#a - \#b \mid = 1 \\
\qquad\qquad \forall\, x : a, y : b \mid x < m! < y\ ] \\
\hline
\end{array}$$

Now we define the input and output of our specification.

$$\begin{array}{|l}
\textit{Input} : \operatorname{seq} \mathbb{N}
\end{array}$$

$$\begin{array}{|l}
\textit{Output} : \mathbb{N} \\
\hline
\exists\, X : \textit{Cubes} \mid [\ X \subseteq \operatorname{ran} \textit{Input} \\
\qquad\qquad \forall\, Y : \operatorname{ran} \textit{Input} \setminus X \mid Y \not\subseteq \textit{Cubes} \\
\qquad\qquad \textit{Output} = \textit{Median}(X)\ ]
\end{array}$$

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;
}
```