Advanced Functional Programming
Uppsala University – Autumn 2012
Assignment 1

Anders Hassis Jonatan Jansson

November 22, 2012

1 List comprehension

The idea of `dividers_of` is to use list comprehension to create a list (using the standard `lists:seq`) of all dividers of a given value n. We remove all values where the remainder is not 0. Note that 1 and n are excluded from the result as they are trivial.

In `primes_up_to` the idea is the same but instead of filtering on remainders we use the `dividers_of` and only keep the values with no dividers as they are prime by definition.

2 Fibonacci trees

The idea of `fibonacci_tree` is to use the *process* and *message* system in Erlang to calculate a sum of nodes in a *fibonacci tree*. This is done by creating a node as root. When a node gets a value of 0 or 1 it is considered a leaf node. Other values n causes the node to expand further by creating two child nodes getting the values $n - 1$ and $n - 2$. The task of each node is to calculate the number of nodes in the subtree in which it is root.

A non-leaf node is implemented to spawn 2 new processes and waits for a *sum*-message to arrive from its parent. The message is passed on to both children. The node waits for an answer from the children, calculates the size of its subtree using the answers, and sends the sum back to its parent.

A leaf node simply waits for a *sum*-message to arrive from its parent and sends 1 back as the number of nodes in a tree containing only a leaf is always 1.

To test the behaviour of `fibonacci_tree`, use the function `test_fibonacci()`. The function test the function for values 10 and 15.

3 Factorization of large integers

The idea of `factorize` is to find the prime factors of a number n using a state to remember previous calculations to improve speed. We have implemented the state as a *dictionary* from the built-in library `dict`. The reason for this is that a *dictionary* has fast random access time, both for reads and writes.

To *factorize* a number the state is first checked. If the number is already calculated, the result is return directly from the state. Otherwise a process tree is created to calculate the prime factors. Each node creates a list l of dividers of the given value. If l is empty the value is prime and is inserted to the state. The value is also returned to the parent as a list, together with the new state.

If the value is not prime, the first divider d in l is chosen. Note the the l is created in decreasing order. The reason for this is to slightly improve the balance of the tree as the lowest divider is always prime. The node then spawns two new nodes. One for d and one for the other divider n/d. The node waits for replies from both child nodes (note the similarity to the *fibonacci tree*).

When both children have replied, the resluting states are merged. The resulting lists are appended as the result will be a list of the prime factors for the current node. The list will be returned to the parent together with the merge of the two states, with the value n added.