Advanced Functional Programming – Autumn 2020
Assignment 1 - Stack Permutations

Li Ju

14th November 2020

Algorithm and implementation

Algorithm used for the problem can be referenced from the site provided in the problem description[1]. The algorithm can be described as following:

1. Starting from an input queue, an output queue and an empty stack.
2. If the output queue is empty, return true
3. Check if the element on the top of the stack or the first element of input queue equals to the first element of the output queue. If true: remove both equal elements. If false, dequeue the first element of the input queue and push it to the stack - if the input queue is empty, return false.
4. Go to step 2

Two points should be noticed in the implementation:
1). In Erlang, it is easier to match the first element of a list. However, given queues are supposed to pop the last element first. Therefore, a helper function named reverse/1 is implemented to reverse queues for better pattern matching.
2). In the implementation, two perm functions are used: perm/2 taking input queue and output queue, and perm/3, taking reversed input queue, reversed output queue and stack. But only perm/2 is exported. Each time perm(input, output) is called, it will call perm(reverse(input), reverse(output), []), recursively do the computing and return the value.

Unit Testing

For Unit testing, 8 test cases are conducted, including test for 3, 4, 5 and 10 elements for both true and false case. EUnit test generators are used.

Property-based Testing

For property-based testing, 2 properties of the function are chosen: 1). all input queues containing 231 permutation patterns cannot be stack-ordered[2], and 2). time cost for evaluation of 1,000,000-element queues should be less than 1 second.

For the first property, the test function named prop_unsortable() is created. The test function will generate an ordered list as the output_queue, then shuffle the output_queue till it contains at least 1 231 permutation patterns as the input_queue. The input_queue with 231 patterns will not be a stack permutation of the ordered queue. Therefore, the results of perm(input_queue, output_queue) should always be false, which will be checked by function prop_unsortable().

For the second property, the test function named prop_efficiency() is created. The function will generate random integers n which are greater than 0 but less than $1,000,000$. Then a sequence list $1, 2, 3 \ldots (n)$ will be generated as the input_queue and the input_queue will be shuffled to be the output_queue. Finally the test function will run perm(input_queue, output_queue) with function timer:rc/3 to check if the time cost is less than 1,000,000 microseconds.

References

[1] Suprotik Dey. "Stack Permutations (Check if an array is stack permutation of other)". Geeksforgeeks, 2019, https://www.geeksforgeeks.org/stack-permutations-check-if-an-array-is-stack-permutation-of-other/

[2] Both Neou, Romeo Rizzi, Stéphane Vialette. "Permutation Pattern matching in (213, 231)-avoiding permutations". Discrete Mathematics and Theoretical Computer Science, DMTCS, 2017, 18 (2), pp.14.1-22. ffhal-01219299v6