universität innsbruck

Advanced Functional Programming | WS 2025/2026 | LVA 703139

Exercise Sheet 12, 10 points | Deadline: Tuesday, January 20, 2026, 4pm

- Solve the tasks in files `Exercise12.hs` and `Exercise12_LPO_Encoder.hs` and upload only these files in OLAT.
- Mark the solved exercises in OLAT.
- Your modified `.hs`-files must compile without error messages.

Task 1 *Queues* **5 p.**

1. Consider the simple queue implementation. The amortized complexity is $O(1)$. This implies that n consecutive operations have cost $O(n)$.

   However, this statement is only true, if always the same queue is used and the queue is not copied. Write a Haskell program that performs $O(n)$ many queue operations (insertion, removal, and queue-copying), and requires $\Theta(n^2)$ time.

   Use `cabal repl Exercise12` and test using `:set +s` within ghci.
   Reason for using read-eval-print-loop: Full compilation might optimize and tune your code so that a quadratic behavior might not be visible after compilation. (2 points)

2. Study the improved implementation `Queue2`. Perform an evaluation of iterated insertion in the style of Slides 22 and 23 for `Queue2` to identify a pattern in the evaluation. Afterwards derive a lower bound on the worst case complexity of `remove2` after a sequence of n many insertions.

   Here, all arithmetic operations should be performed immediately, and the first argument of `foldl` should always be evaluated to WHNF before performing the next recursive step with `foldl`. (3 points)

Task 2 *Profiling and Optimization* **5 p.**

Consider the solution to the implementation of the LPO-encoding of exercise 11, which is given to you in `Exercise12_LPO_Encoder.hs`. Via profiling it was figured out that the lookup in the encoding is time-consuming.
Use any techniques that you learned in this course and optimize the implementation to make it more efficient. You may change `mainLPO` in `Exercise12.hs` as well as everything in file `Exercise12_LPO_Encoder.hs`.
Important: It is not allowed to make any changes that influence the resulting SMT-formula. In particular the output of the original version and the optimized version must be 100 % identical.
Perform profiling before and after your modification and briefly report on the results.
The measurement and checking is done with the following invocations.

```
# before optimization:
time cabal run Exercise12 -- lpo ariTRSs.txt +RTS -N4 -RTS > orig_output.txt

# after optimization
time cabal run Exercise12 -- lpo ariTRSs.txt +RTS -N4 -RTS > new_output.txt

diff orig_output.txt new_output.txt || echo "disqualified"

# measure real time output of "time" command from optimized implementation
```

On a test computer, the optimized version from the example solution could reduce the required time from previously 39.4 seconds down to 3.4 seconds.
The fastest student solution will be awarded with a copy of "Parallel and Concurrent Programming in Haskell" (unfortunately, I was informed about an expected delivery date of July 24).