universität innsbruck

Advanced Functional Programming — WS 2025/2026 — LVA 703139

Exercise Sheet 8, 10 points — Deadline: Tuesday, December 2, 2025, 4pm

- Solve the tasks in files `Exercise08*.hs` and upload only these files in OLAT.
- Mark the solved exercises in OLAT.
- Your modified `Exercise08*.hs` files must compile with `ghci` without error messages.

Task 1 *Monad Transformer Stacks* **2 p.**

Convince yourself that the order of monad transformers matters. Use two different monad transformer stacks to run the following code, so that the result is different. Provide both wrapper functions and explain the difference.

```haskell
testApp :: (MonadError String m, MonadWriter [Bool] m) => m Int
testApp = do
  tell [True]
  _ <- throwError "bar"
  return 5
```

Task 2 *Abstract SMT Encoder* **4 p.**

This task is about the development of an abstract SMT encoder.
This encoder should be implemented with the help of monad transformers.
The abstract encoder should have the following functionality:

- it keeps track of formulas that should be asserted;
- it can generate fresh variables on demand;
- it can be run, i.e., in the end a string is produced in SMT-LIB-2 format (https://smt-lib.org); this string might then be passed to an SMT-solver.

You already find datatypes to represent integer arithmetic formulas in file `SMT.hs`, and also the conversion to SMT-LIB-2 is available there.
Important: whenever you generate a fresh variable, then internally a `SmtVarDecl` has to be added.
Restriction: for this task, the usage of `RWS` is forbidden: you should use one or many more basic monads or monad transformers.

Example: Running the Haskell program

```haskell
smtEncoderTest :: IO ()
smtEncoderTest = do
  let output = runSmtEncoder (do
        b <- getNewSmtVariable SmtBool
        c <- getNewSmtVariable SmtBool
        x <- getNewSmtVariable SmtInt
        let f1 = Gt (IAVar x) (IAConst 5)
        let f2 = disj [BoolVar b, f1]
        assertFormula f2
        assertFormula (Equiv (BoolVar c) f2))
  putStrLn output
```

(which is located in `SMT_Encoder_Test.hs`) should print a string that is equivalent to

```
(set-logic QF_LIA)
(declare-fun x1 () Bool)
(declare-fun x2 () Bool)
(declare-fun x3 () Int)
(assert (or x1 (> x3 5)))
(assert (= x2 (or x1 (> x3 5))))
(check-sat)
```

where of course, you might have used a different numbering scheme for fresh variables.

Task 3 *LPO-Encoding* **4 p.**

The SMT-encoder from Task 2 should now be used to provide an encoding of the lexicographic path order (LPO). LPO is an extension of the embedding order, and the details are not relevant for this exercise. Just for your interest: if all rules $\ell \to r$ of a TRS satisfy $\ell >_{LPO} r$, then the TRS is terminating. Deciding the former property is NP-complete, so we encode it into Sat-Modulo-Theories and let an SMT-solver perform the search. Here are the required features of the encoding:

- As for the embedding relation, one needs memoization to avoid exponential runtime.
- LPO is parametrized by a precendence, where every symbol gets assigned a number between 1 and n, and n is the total number of symbols in the signature.
- One should perform an encoding similar to the Tseitin-transformation: whenever a non-atomic formula ϕ is obtained, a new Boolean variable b is introduced, and it is asserted that b is equivalent to ϕ. In this way, large formulas are not repeated in the encoding.

In file `Exercise08_LPO.hs` you find a partial implementation of an LPO-encoder where some functions are completely undefined, or at least need some adjustments. Moreover, at the end of the file you find two example TRSs with expected output.
Fill in the gaps and test your encoding.