Report Week 3

Assignment 1 (also in comments)

The *testRules*-function is used for testing the implemented functions for $(n+1)*3$ different formulas. The function *testRules2* tests equivalence and entails for two different formulas, whereas *testRules* $(n|n > 0)$ tests equivalence and entails for one (equal) formula and the rules:

```
satisfiable(x) -> ¬contradiction x
¬satisfiable x -> contradiction x & ¬tautology x
tautology x -> satisfiable x & ¬contradiction x
```

Testing equivalence and entailment uses the following tests for two forms x and y:

- $x \vDash \bot$
  - Here $\bot$ is used as Dsj [];
- $\top \vDash x$
  - Here $\bot$ is used as Cnj [];
- $(x \vDash y \wedge y \vDash x) \rightarrow (x = y)$
  - This tests equality: if x entails y and vice versa, then they are equal.
- $(x \neq y) \rightarrow (\neg(x \vDash y) \vee \neg(y \vDash x))$
  - This tests unequality: if x is uneqyual to y, then either x does not entail y or vice versa.

Assignment 3

In order to test whether or not the CNF-function works, we reasoned that two tests would be sufficient:

1. Testing the equivalence of CNF-processed form with the original form;
2. Grammar test, using the following rules, converted to a test using lists:
   (a) Rules

```
L :== p | ¬p
D :== L | L ∨ D
C :== D | D ∧ C
```

For the random tests, we developed a set of functions:

```
testCnf 0 = do
  x <- getRandomFSmpl
  return $ equivAndFstCnfGrammar (cnf x) x

testCnf n = do
  y <- testCnf 0
  z <- testCnf (n - 1)
  return $ y && z
```

testCnf(n) tests $n+1$ random grammars for equivalence and for having the correct grammar. $testCnf(0)$ calls the function *equivAndFstCnfGrammar* (which should be read as: for x and y, make sure that x is equivalent to y and that x is in conjunctive normal form). When an error occurs, it will check where the error came from. The *equivAndFstCnfGrammar,* the *grammar* test and the *equivalence* test is implemented as following.

```
cnfTest cnfF orig = (grC cnfF)
  || error ("Error in Grammar! 'CNF':\n\n" ++ (show cnfF)
  ++ "\n\nOriginal:\n\n" ++ (show orig))

equivTest cnfF orig = (equiv cnfF orig)
  || error ("Error in Equivalence! 'CNF':\n\n" ++ (show cnfF)
  ++ "\n\nOriginal:\n\n" ++ (show orig))

equivAndFstCnfGrammar x y = (cnfTest x y) && (equivTest x y)
```

Whereas the actual *equiv*-check was implemented in the first assignment and tested, and the *grammar*-check, which is based on lists, is defined as following:

```
grL (Prop p) = True
grL (Neg (Prop p)) = True
grL _ = False

grD (Dsj (a:as)) = all (\x -> x) [ grL x | x <- (a:as) ]
grD x = grL x

grC (Cnj (a:as)) = all (\x -> x) [ grD x | x <- (a:as) ]
grC x = grD x
```

Note, that we use $(a : as)$ everywhere instead of as, to ensure that the list contains at least one element. Also note the constraint that a list of conjuncts cannot contain lists of conjuncts and a list of disjuncts cannot contain lists of disjuncts. Although this is probably not a mandatory constraint, it does enforce us to create clean CNFs, by removing redundant parantheses.

Testing the cnf-function with 5000 different formulas:

```
> testCnf 5000
True
```

resulted in true, meaning that it is very likely that the cnf-function has been correctly implemented.