Revision

In this tutorial you will get some practice on exam-style questions and use Haskell to check your answers. You should place the three files, `CL7a.hs`, `CL9a.hs`, `CL10prop.hs`, and `CL10regex.hs` in a directory; run ghci and load `CL10prop.hs` for the propositional logic exercise, and `CL10regex.hs` for the regex and machines exercise. You can add your working code to the two `CL10*.hs` files.

Propositional logic

`CL7a.hs` provides lots of things. For this tutorial you need only the following

```
data Atom = A | B | C | D | W | X | Y | Z deriving (Eq, Show)
data Wff a = V a
           | T
           | F
           | Not (Wff a)
           | Wff a :|: Wff a
           | Wff a :&: Wff a
           | Wff a :->: Wff a
           | Wff a :<->: Wff a
type Env a = [(a, Bool)]
envs :: [a] -> [Env a]
eval :: Eq a => Env a -> Wff a -> Bool
```

1. The next question concerns the 256 possible truth valuations of eight propositional letters, $A\,B\,C\,D\,W\,X\,Y\,Z$.

   This question asks you to complete the definition of the function `countModels` in the CL10.hs file, so you can check your answers. This is the only function you need to write in this tutorial :-)

   For example, to check how many valuations satisfy $(A \to B) \leftrightarrow C$, you would write.

   ```
   countModels :: Wff Atom -> Int
   > countModels ((V A :->: V B) :<->: V C)
   128
   ```

2. For each of the following expressions say how many of the 256 valuations satisfy the expression. Then code the expression as a value of type `Wff Atom` and use Haskell to check your answer.

   Briefly explain your reasoning.

   (a) $(X \to A \wedge (\neg V \vee D \to B) \wedge W \to Z) \leftrightarrow C$
   Hint: you should not need truth tables, or any long-winded calculation, or machine assistance, to answer this question. You don't even need the answer to any questions you may have about the relative precedence of the various operators!

   (b) $(A\,?\,B : C)$
   (This expression represents the Boolean operator equivalent to the Haskell expression `if a then b else c` where all three arguments are of type `Bool`. You will have to find an equivalent `Wff Atom` in order to check your answer for this question.)

   (c) $(A \to B) \to C$

   (d) $A \to (B \to C) \wedge (D \vee W \vee W)$

(e) $(A \to B) \wedge (B \to C) \wedge (C \to A) \wedge (W \to X) \wedge (X \to C)$

(f) $(A \to D) \wedge (C \to D) \wedge (\neg D \to W) \wedge (W \to D) \wedge (X \to C)$

3. For each of the following entailments complete two Karnaugh maps, one to represent the assumption and one the conclusion, by **marking the valuations that make the expression false**.

   Examine thes Karnaugh maps to indicate whether the entailment is valid. Explain your reasoning.

   Use your Karnaugh maps to give a simple CNF for each assumption.

   (a) $\neg(\neg A \wedge \neg C) \wedge (B \to \neg C) \models \neg(B \to A)$

   (b) $(A \oplus B) \to (C \to D) \models (A \to C) \to (A \to (\neg B \to D))$

4. Use Karnaugh maps to convert each of the following expressions to CNF

   - $R \to (P \wedge Q)$
   - $(S \oplus P \ ? \ T \ : \ R)$
   - $(T \vee P \ ? \ Q : R)$
   - $\neg(A \leftrightarrow B) \to C$

Gentzen Rules

Satifaction The rules given here concern the satisfaction relation, $\Gamma \vDash \Delta$, between finite sets of predicates).

$$\frac{}{\Gamma, a \vDash \Delta, a} \ (I)$$

$$\frac{\Gamma, a, b \vDash \Delta}{\Gamma, a \wedge b \vDash \Delta} \ (\wedge L) \qquad \frac{\Gamma \vDash a, b, \Delta}{\Gamma \vDash a \vee b, \Delta} \ (\vee R)$$

$$\frac{\Gamma, a \vDash \Delta \quad \Gamma, b \vDash \Delta}{\Gamma, a \vee b \vDash \Delta} \ (\vee L) \qquad \frac{\Gamma \vDash a, \Delta \quad \Gamma \vDash b, \Delta}{\Gamma \vDash a \wedge b, \Delta} \ (\wedge R)$$

$$\frac{\Gamma \vDash a, \Delta \quad \Gamma, b \vDash \Delta}{\Gamma, a \to b \vDash \Delta} \ (\to L) \qquad \frac{\Gamma, a \vDash b, \Delta}{\Gamma \vDash a \to b, \Delta} \ (\to R)$$

$$\frac{\Gamma \vDash a, \Delta}{\Gamma, \neg a \vDash \Delta} \ (\neg L) \qquad \frac{\Gamma, a \vDash \Delta}{\Gamma \vDash \neg a, \Delta} \ (\neg R)$$

a and b are predicates, Γ, Δ are finite sets of predicates, and Γ, a refers to $\Gamma \cup \{a\}$.

5. Use reduction to determine whether the sequent
$(B \to A) \to (A \to C) \ \models \ B \to C$
is universally valid, and produce a counterexample if it is not.

6. Use reduction to determine whether the sequent
$(P \vee Q) \to (R \vee S),\ Q \to \neg R \ \models \ Q \to S$
is universally valid and produce a counter-example if it is not.

Entailment Question 7 refers to the following rules.

The rules given here define the entailment relation, $\Gamma \vdash \Delta$, between finite sets of propositional formulæ (wffs).

$$\frac{}{\Gamma, A \vdash \Delta, A}\ (I)$$

$$\frac{\Gamma, A, B \vdash \Delta}{\Gamma, A \wedge B \vdash \Delta}\ (\wedge L) \qquad \frac{\Gamma \vdash A, B, \Delta}{\Gamma \vdash A \vee B, \Delta}\ (\vee R)$$

$$\frac{\Gamma, A \vdash \Delta \quad \Gamma, B \vdash \Delta}{\Gamma, A \vee B \vdash \Delta}\ (\vee L) \qquad \frac{\Gamma \vdash A, \Delta \quad \Gamma \vdash B, \Delta}{\Gamma \vdash A \wedge B, \Delta}\ (\wedge R)$$

$$\frac{\Gamma \vdash A, \Delta \quad \Gamma, B \vdash \Delta}{\Gamma, A \to B \vdash \Delta}\ (\to L) \qquad \frac{\Gamma, A \vdash B, \Delta}{\Gamma \vdash A \to B, \Delta}\ (\to R)$$

$$\frac{\Gamma \vdash A, \Delta}{\Gamma, \neg A \vdash \Delta}\ (\neg L) \qquad \frac{\Gamma, A \vdash \Delta}{\Gamma \vdash \neg A, \Delta}\ (\neg R)$$

A and B are propositional expressions, Γ, Δ are sets of expressions, and Γ, A refers to $\Gamma \cup \{A\}$.

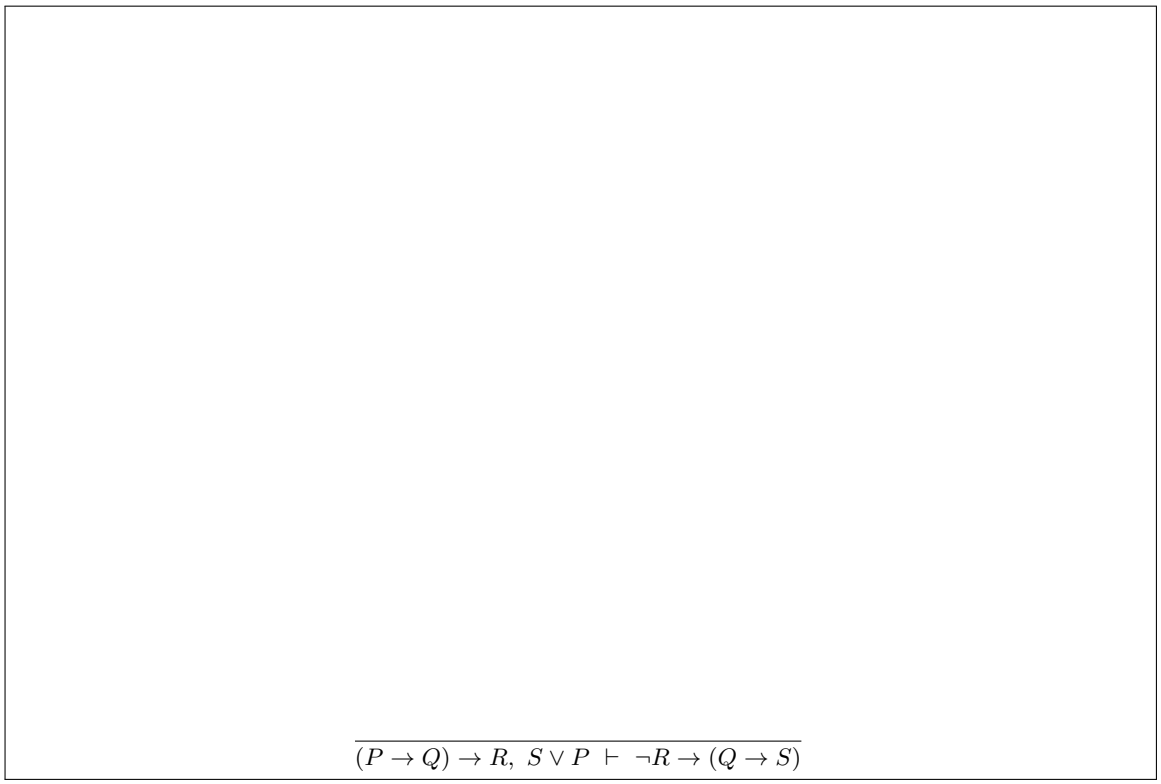
7. Use the Gentzen rules, provided on the previous page, to derive the following entailment,

$$(P \to Q) \to R,\ S \vee P \ \vdash \ \neg R \to (Q \to S) \qquad \text{(goal)}$$

(a) Which of the rules have a conclusion matching this goal?
For each such rule complete a line in the table below showing the name of the rule and the bindings for Γ, Δ, A, B

Rule	Γ	Δ	A	B

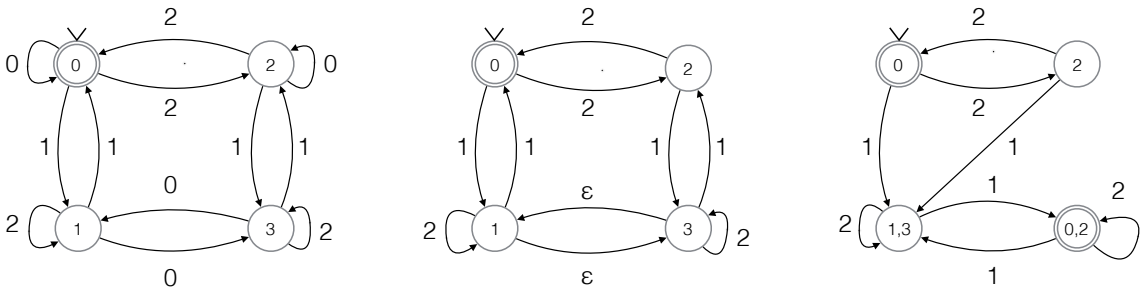
(b) Use the Gentzen rules to construct a formal proof with the goal as conclusion.
Label each step in your proof with the name of the rule being applied.


$\overline{(P \to Q) \to R,\ S \vee P \ \vdash\ \neg R \to (Q \to S)}$

regex and NFA

8. We begin this section with an exercise on NFA/DFA, to show how you use the code `Tutorial9Solution.hs` to check your answers to problems in this area.

   Consider the three machines below. The left-hand machine keeps track of the value of a ternary number, mod 4.[1] It is a DFA. The central machine is almost the same, but this is an NFA obtained by replacing each 0-labelled transition by an ε-transition (we remove the reflexive ε-transitions to get an equivalent machine). The right-hand machine is intended to be a DFA derived from this NFA, using the subset construction.



   We can check our work by coding the central machine as an NFA, and applying the `nfa2dfa` function from CL9.

```
eg4 = mkNFA qs as ts es ss fs where
  qs = [0..3]
  as = "12"
  ts = [ (0,'1',1), (0,'2',2), (1,'1',0), (1,'2',1)
       , (2,'2',0), (2,'1',3), (3,'1',2), (3,'2',3) ]
  es = [ (1,3), (3,1) ]
  ss = [0]
  fs = [0]
```

   We then call `nfa2dfa` to see the result of the subset construction (each `Set` of `xs` is shown as `fromList xs`, since evaluating this expression would recreate the set). Whitespace has been added to make the output below more immediately legible.

```
> nfa2dfa) eg4
NFA
  (fromList[fromList [0],fromList [0,2],fromList [1,3],fromList [2]])
  (fromList "12")
  [(fromList [0],'1',fromList [1,3]),  (fromList [0],'2',fromList [2])
  ,(fromList [0,2],'1',fromList [1,3]),(fromList [0,2],'2',fromList [0,2])
  ,(fromList [1,3],'1',fromList [0,2]),(fromList [1,3],'2',fromList [1,3])
  ,(fromList [2],'1',fromList [1,3]),  (fromList [2],'2',fromList [0])]
  [] (fromList [fromList [0]]) (fromList [fromList [0],fromList [0,2]])
```

   Here we can see the superstates and transitions generated by the subset construction. For a more succinct representation of the same DFA we can convert it to an `intFSM`.

```
> (intFSM.asFSM) it
FSM (fromList [0,1,2,3])
```

[1] Observe that this is a palindromic machine – reversing all the arrows and exchanging the start and finish states gives us back the same machine. What does this tell you about ternary numbers divisible by $4_{10} = 11_3$? Is the same true of n-ary numbers that are 0 mod $(n+1)$?

```
    (fromList "12")
    [(0,'1',2),(0,'2',3),(1,'1',2),(1,'2',1)
    ,(2,'1',1),(2,'2',2),(3,'1',2),(3,'2',0)]
    (fromList [0]) (fromList [0,1])
```

Can we find a regular expression for the language accepted by the NFA (or equivalently, by the right-hand DFA)? Here are three purported answers, which, if any, is/are correct? `(22)*(2*12*12)*`, `(22)*(2*12*12*)*`, `(2*12*128)*(22)*`. We can use the tools from CL9 to check.

```
w = S "1"      -- 1 wun
t = S "2"      -- 2 two
ts = Star t    -- 2*
r0 = Star (t:>:t) :>: Star(ts:>:w:>:ts:>:w:>:t)
r1 = Star (t:>:t) :>: Star(ts:>:w:>:ts:>:w:>:ts)
r2 = Star(ts:>:w:>:ts:>:w:>:ts) :>: Star (t:>:t)
```

We can check our coding of the regex using the function `pretty :: Regex -> String`.

```
> pretty r0
"(22)*(2*12*12)*"
> pretty r1
"(22)*(2*12*12*)*"
> pretty r2
"(2*12*12*)*(22)*"
```

We use the function `minimalDFA :: Regex -> FSM Int` to produce machine for each regex

```
> minimalDFA r0
FSM (fromList [1,2,3,4,5,6]) (fromList "12")
  [(1,'1',6),(1,'2',4),(2,'1',6),(2,'2',2)
  ,(3,'1',6),(3,'2',2),(4,'1',6),(4,'2',1)
  ,(5,'2',3),(6,'1',5),(6,'2',6)]
  (fromList [1]) (fromList [1,3])
> minimalDFA r1
FSM (fromList [0,1,2,3]) (fromList "12")
  [(0,'1',3),(0,'2',2),(1,'1',3),(1,'2',1)
  ,(2,'1',3),(2,'2',0),(3,'1',1),(3,'2',3)]
  (fromList [0]) (fromList [0,1])
> minimalDFA r2
FSM (fromList [0,1,2,3]) (fromList "12")
  [(0,'1',3),(0,'2',2),(1,'1',3),(1,'2',1)
  ,(2,'1',3),(2,'2',0),(3,'1',1),(3,'2',3)]
  (fromList [0]) (fromList [0,1])
```

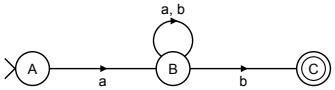
The upshot is that `r1` and `r2` correspond to the `FSM Int` given above – but note that we have to relabel the states – swapping states 2 and 3 – to see that these machines are isomorphic (of the same form).

Can you give an argument to show, from first principles, that `r1` and `r2` are equivalent?

To complete this exercise, you should now use the ternary-number-divisible-by-4 machine to generate two further NFA, by substituting ε, first for 1, then for 2. For each of these NFA, do the subset construction by hand and check your answer using Haskell, as we did above. Then produce a regex for each machine, and again check your answer by using Haskell to produce a minimal DFA for this regex.

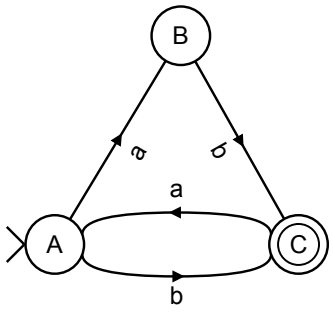
9. Give a regular expression (`re`) for the language accepted by each FSM. Use Haskell to check your answer.
   Mark the check boxes to show the strings it accepts, and whether it, together with any implicit black hole state, is deterministic.
   Draw an equivalent DFA if it is not; again, use Haskell to check your answer.

(a)



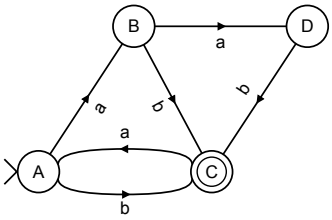
`abba` ☐
`abab` ☐
`abbb` ☐
`abba` ☐
DFA ☐
`re:`

(b)



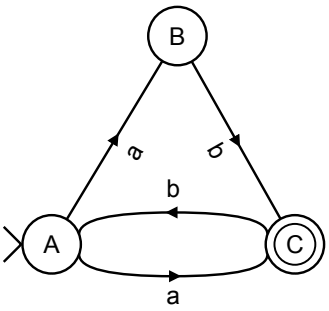
`abba` ☐
`abab` ☐
`abbb` ☐
`baab` ☐
DFA ☐
`re:`

(c)



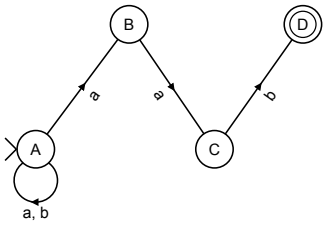
`abab` ☐
`abba` ☐
`babb` ☐
`baba` ☐
DFA ☐
`re:`

(d)



`aaa` ☐
`baab` ☐
`bbab` ☐
`bbbb` ☐
DFA ☐
`re:`

(e)



`aaab` ☐
`aab` ☐
`baab` ☐
`bbbb` ☐
DFA ☐
`re:`