PROGRAMMING IN HASKELL

Chapter 8 – Tautology Checker

Tautologies

- In logic, tautologies are logical propositions that are always true.
- We have the usual logical connectives:
  - and, or, not, if-then
- We have the usual truth tables for connectives.
- Some expressions are contingent, while others are tautologies:
  - if [A then [A and B]]
  - if [A and [if A then B]] then B

Declare Proposition Type

```
data Prop = Const Bool
          | Var Char
          | Not Prop
          | And Prop Prop
          | Imply Prop Prop
```

Note: we can use the Haskell ( ) to indicate grouping.

Proposition Examples

```
p1 :: Prop
p1 = And (Var 'A') (Not (Var 'A'))

p2 :: Prop
p2 = Imply (And (Var 'A') (Var 'B')) (Var 'A')

p3 :: Prop
p3 = Imply (Var 'A') (And (Var 'A') (Var 'B'))

p4 :: Prop
p4 = Imply (And (Var 'A') (Imply (Var 'A') (Var 'B'))) (Var 'B')
```

Substitutions

```
type Subst = Assoc Char Bool
type Assoc k v = [(k,v)]
```

```
find :: Eq k => k -> Assoc k v -> v
find k t = head [v | (k',v) <- t, k == k']
```

Note: Assoc is essentially a look-up table of key-value pairs. The find function gets the value of a key. Subst is a key-value pair of Char and Bool, e.g. [(`A`,False),(`B`,True)].

Evaluation Function

```
eval :: Subst -> Prop -> Bool
eval _ (Const b)    = b
eval s (Var x)      = find x s
eval s (Not p)      = not (eval s p)
eval s (And p q)    = eval s p && eval s q
eval s (Imply p q) = eval s p <= eval s q
```

Note: logical implication is implemented as <= ordering on logical variables.

Towards Substitutions

To determine if an expression is a tautology, we must evaluate the truth of the whole expression with respect to all possible combinations of truth values (think about the truth tables).

First, we gather all the variables of the expression.

Gather Variables

```
vars :: Prop -> [Char]
vars (Const _)    = []
vars (Var x)      = [x]
vars (Not p)      = vars p
vars (And p q)    = vars p ++ vars q
vars (Imply p q) = vars p ++ vars q
```

Gather Variables

```
vars :: Prop -> [Char]
vars (Const _)    = []
vars (Var x)      = [x]
vars (Not p)      = vars p
vars (And p q)    = vars p ++ vars q
vars (Imply p q) = vars p ++ vars q
```

```
vars p2 = [`A`, `B`, `A`]
```

remove duplicates later.

Gather Variables

```
vars :: Prop -> [Char]
vars (Const _)    = []
vars (Var x)      = [x]
vars (Not p)      = vars p
vars (And p q)    = vars p ++ vars q
vars (Imply p q) = vars p ++ vars q
```

```
vars p2 = [`A`, `B`, `A`]
```

remove duplicates later.

Generate Possible Values

What we know from the truth tables is if we have three propositional variables, we need 8 lines of the truth table to represent all possible combinations of truth values.

Have a recursive definition of the truth tables.

Generate Truth Tables

```
bools :: Int -> [[Bool]]
bools 0 = [[]]
bools n = map (False:) bss ++ map (True:) bss
        where bss = bools (n-1)
```

```
bools 3 =
[[False,False,False],[False,False,True],
[False,True,False],[False,True,True],
[True,False,False],[True,False,True],
[True,True,False],[True,True,True]]
```

Variables and Truth Tables

```
substs :: Prop -> [Subst]
substs p = map (zip vs) (bools (length vs))
           where vs = rmdups (vars p)

rmdups :: Eq a => [a] -> [a]
rmdups []     = []
rmdups (x:xs) = x : filter (/= x) (rmdups xs)
```

Variables and Truth Tables

substs p2 =

```
[[('A',False),('B',False)],[('A',False),('B',True)],
[('A',True),('B',False)],[('A',True),('B',True)]]
```

Tautology Checker

```
isTaut :: Prop -> Bool
isTaut p = and [eval s p | s <- substs p]
```

A proposition is a tautology if and only if it is true for all possible substitutions.

Result

```
isTaut p1 = False
isTaut p2 = True
isTaut p3 = False
isTaut p4 = True
```

Nice!