

G51PGP Programming Paradigms

Case Study 3

Tautology Checker

Abstract

The goal of this coursework is to write a Haskell script that uses truth tables to decide if logical propositions are true for all possible values of their variables, i.e. if they are *tautologies*.

Background

Suppose that propositions are built up from the variables A, B, \dots, Z together with the following constants and operators:

| Symbol | Meaning |
|-------------------|--------------|
| f | false |
| t | true |
| \neg | not |
| \wedge | and |
| \vee | or |
| \Rightarrow | implies |
| \Leftrightarrow | equivalences |

For example, the following are all propositions:

$$\begin{aligned} &A \wedge \neg A \\ &(A \wedge B) \Leftrightarrow (B \wedge A) \\ &A \Rightarrow (A \wedge B) \\ &(A \wedge (A \Rightarrow B)) \Rightarrow B \end{aligned}$$

A simple method for deciding if a proposition is a tautology is to calculate its truth table. For example, the following truth tables prove that the second and fourth propositions above are both tautologies:

| A | $A \wedge \neg A$ |
|-----|-------------------|
| f | f |
| t | f |

| A | B | $(A \wedge B) \Leftrightarrow (B \wedge A)$ |
|-----|-----|---|
| f | f | t |
| f | t | t |
| t | f | t |
| t | t | t |

| A | B | $A \Rightarrow (A \wedge B)$ |
|-----|-----|------------------------------|
| f | f | t |
| f | t | t |
| t | f | f |
| t | t | t |

| A | B | $(A \wedge (A \Rightarrow B)) \Rightarrow B$ |
|-----|-----|--|
| f | f | t |
| f | t | t |
| t | f | t |
| t | t | t |

Type Definition

Your script must contain the following type definition:

```
data Prop = Const Bool
          | Var Char
          | Not Prop
          | And Prop Prop
          | Or Prop Prop
          | Imply Prop Prop
          | Equiv Prop Prop
  deriving Show

type Subst = [(Char,Bool)]
```

That is, a proposition is represented by a value of type **Prop**, and a substitution (mapping variables to values) is represented by a value of type **Subst**.

Function Definitions

- **Exercise:** Define Haskell values

```
p1 :: Prop
p2 :: Prop
p3 :: Prop
p4 :: Prop
```

that represent the following four propositions:

$$\begin{aligned} & A \wedge \neg A \\ (A \wedge B) & \Leftrightarrow (B \wedge A) \\ A & \Rightarrow (A \wedge B) \\ (A \wedge (A & \Rightarrow B)) \Rightarrow B \end{aligned}$$

- **Exercise:** Define a function

```
vars :: Prop -> [Char]
```

that calculates the list of variables in a proposition. For example, `vars p2` should give `['A','B','B','A']`, which can also be written as `"ABBA"`.

- **Exercise:** Define a function

```
rmtdups :: Eq a => [a] -> [a]
```

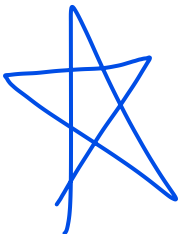
that removes the duplicate elements from a list. For example, `rmtdups "ABBA"` should give the list of characters `"AB"`.

- **Exercise:** Define a function

```
bools :: Int -> [[Bool]]
```

that calculates all possible lists of logical values of a specific length. For example, `bools 2` should give the following list:

```
[[False,False],
 [False,True ],
 [True ,False],
 [True ,True  ]]
```



- **Exercise:** Using `vars`, `rmdups` and `bools`, define a function

```
substs :: Prop -> [Subst]
```

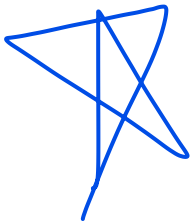
that calculates all possible substitutions for the variables of a proposition. For example, `substs p2` should give the following list:

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

- **Exercise:** Define a function

```
find :: Eq a => a -> [(a,b)] -> b
```

that finds the value associated with a key in a list of (key,value) pairs. For example, `find B [(A,True),(B,False)]` should give `False`.



- **Exercise:** Define a function

```
eval :: Subst -> Prop -> Bool
```

that evaluates a proposition to a logical value, given a substitution that defines the logical value of each variable in the proposition. For example, `eval [(A,True),(B,False)] p2` should give the value `True`.

- **Exercise:** Define a function

```
isTaut :: Prop -> Bool
```

that decides if a proposition is a tautology, by evaluating the proposition for each possible substitution of its variables. For example, `isTaut p1` should give the value `False`, while `isTaut p2` should give the value `True`.

— *The End* —