COL703: Logic for Computer Science
I semester 2018-19

Programming Assignment: Tableau for Propositional Logic

Using the datatype for propositional logic arguments as defined in the tautology checker, implement the tableau method in Standard ML or OCaml to

1. verify the validity of propositional logic arguments and
2. otherwise display all falsifying truth assignments in case of an invalid argument.

The signature of the package is given in Standard ML by the following:

```
signature PropLogicTableau =

sig
    exception Atom_exception
    datatype Prop =
         ATOM of string     |
         NOT of Prop        |
         AND of Prop * Prop |
         OR of Prop * Prop  |
         COND of Prop * Prop |
         BIC of Prop * Prop
    type Argument = Prop list * Prop
    val tableau: Prop list -> unit
end;
```

You may translate it into OCaml (if you prefer) by using the corresponding syntax given in `https://people.mpi-sws.org/~rossberg/sml-vs-ocaml.html`.

What you have to do

1. Decide whether to program in **Standard ML** or **Ocaml** and choose the appropriate instruction from each of the following steps.
2. Create a signature file

   Standard ML. <*entry-number*>`-sig.sml` containing the above signature.

   OCaml. <*entry-number*>`-sig.ml` containing the Ocaml translation of the above signature.
3. Implement the signature above as a structure and create a source file

   Standard ML. <*entry-number*>`-str.sml` which implements the above Standard ML signature.

   OCaml. <*entry-number*>`-str.mli` which implements the Ocaml signature.
4. Your implementation reads a file <*arg*>`.sml` or <*arg*>`.ml` which contains an argument written in the appropriate language and outputs a file <*arg*>`.dot` which contains the tableau for the argument in the `.dot` format.
5. Submit a <*entry-number*>`.tgz` file containing the signature and structure implmentation files.

Input Here is an argument that you might want to consider as a test case.

> *If I perceive then my perception is either delusive or veridical.*
>
> *If my perception is delusive then I perceive something and yet I do not perceive a material object.*
>
> *If I perceive something and yet I do not perceive a material object, then I perceive a sensation.*
>
> *If my perception is veridical, then I perceive something.*
>
> *If my perception is veridical and I perceive a material object, then my experience in veridical perception would always differ qualitatively from my experience in delusive perception.*
>
> *My experience in veridical perception does not always differ qualitatively from my experience in delusive perception.*
>
> *Therefore, if I perceive, then I perceive a sensation and I do not perceive a material object.*

whose atoms are defined (in SML syntax) below.

```
val perceive = ATOM "I perceive";
val delusive = ATOM "My perception is delusive";
val veridical = ATOM "My perception is veridical";
val something = ATOM "I perceive something";
val material = ATOM "I perceive a material object";
val sensation = ATOM "I perceive a sensation";
val differ = ATOM "My experience in veridical perception would always differ
qualitatively from my experience in delusive perception";
```

The following is an SML translation of the same sentences into the datatype `Prop` followed by a definition of the variable `arg` which represents the entire argument. **Do not change the name of the variable** `arg`.

```
  val h1 = COND (perceive, NOT (BIC (delusive, veridical)));
  val h2 = COND (delusive, AND (perceive, NOT (material)));
  val h3 = COND (AND (perceive, NOT (material)), sensation);
  val h4 = COND (veridical, perceive);
  val h5 = COND (AND (veridical, material), differ);
  val h6 = NOT (differ);
  val c = COND (perceive, AND (sensation, NOT (material)));
  val arg = ([h1, h2, h3, h4, h5, h6], c);
```

The call to the tableau on the argument `arg` will be `tableau((#2 arg)::(#1 arg))`.

Output

Each `.dot` file may be rendered using the `xdot` program
`https://www.graphviz.org/`
In particular, render them as done in example 9.2 in the course slides following the same coding conventions (black arrows, blue dotted arrows, red $\times$ for closed paths along with a red undirected arc for complementary pairs in a path. The source code for the example 9.2 is given below.

However the following `.dot` code has embedded latex commands for the math symbols to be displayed correctly.

```
digraph{
      ranksep = 0.35;
```

```
     node [shape=plaintext];
     0 [texlbl="\underline{0. {\LARGE \color{green} $\neg ((p \wedge q)\vee(\neg p \wedge r))\rightarr
     1 [texlbl="\underline{1. {\LARGE \color{green} $((p \wedge q) \vee (\neg p \wedge r)) \wedge (\ne
     2 [texlbl="\underline{2. {\LARGE \color{green} $(p \wedge q) \vee (\neg p \wedge r))$}}"];
     3 [texlbl="\underline{3. {\LARGE \color{green} $\neg ((\neg p \vee q) \wedge (p \vee r))$}}"];
     4 [texlbl="\underline{4. {\LARGE \color{green} $\neg (\neg p \vee q)$}}"];
     5 [texlbl="\underline{5. {\LARGE \color{green} $\neg (p \vee r)$}}"];
     6 [texlbl="\underline{6. {\LARGE \color{green} $\neg \neg p$}}"];
     7 [texlbl="\underline{7. {\LARGE \color{green} $\neg q$}}"];
     8 [texlbl="\underline{8. {\LARGE \color{green} $p$}}"];
     9 [texlbl="\underline{9. {\LARGE \color{green} $p \wedge q$}}"];
     10 [texlbl="\underline{10. {\LARGE \color{green} $\neg p \wedge r$}}"];
     11 [texlbl="\underline{11. {\LARGE \color{green} $p$}}"];
     12 [texlbl="12. {\LARGE
       $\displaystyle{\frac{\color{green}{q}}
    {\color{red}{\times}}}
       $}"];
     13 [texlbl="13. {\LARGE
       $\displaystyle{\frac{\color{green}{\neg p}}
      {\color{red}{\times}}}
$}"];
     14 [texlbl="\underline{14. {\LARGE \color{green} $\neg p$}}"];
     15 [texlbl="\underline{15. {\LARGE \color{green} $\neg r$}}"];
     16 [texlbl="\underline{16. {\LARGE \color{green} $p \wedge q$}}"];
     17 [texlbl="\underline{17. {\LARGE \color{green} $\neg p \wedge r$}}"];
     18 [texlbl="18. {\LARGE
       $\displaystyle{\frac{\color{green}{p}}
    {\color{red}{\times}}}
        $}"];
     19 [texlbl="\underline{19. {\LARGE \color{green} $\neg p$}}"];
     20 [texlbl="20. {\LARGE
        $\displaystyle{\frac{\color{green}{r}}
                           {\color{red}{\times}}}
        $}"];

     subgraph dir
     {
           0 -> 1;
           1 -> 2;
           2 -> 3;
           3 -> 4;
           3 -> 5;
           4 -> 6;
           6 -> 7;
           7 -> 8;
           8 -> 9;
           8 -> 10;
            10 -> 13;
           9 -> 11;
        11 -> 12;
           5 -> 14;
           14 -> 15;
           15 -> 16;
           15 -> 17;
```

```
            16 -> 18;
     17 -> 19;
     19 -> 20;
      }

      subgraph ancestor
      {
        edge [dir=back, color=blue, style=dashed]
    1->3;
    2->9;
    2->10;
    2->16;
    2->17->20;
    4->7;
    6->8;
    5->15;
      }

      subgraph undir
      {
        edge [dir=none, color=red]
       7 -> 12;
       8 -> 13;
       14 -> 18;
    15 -> 20;
     }
}
```

Note:

1. You are *not* allowed to change any of the names given in the signature. You are not even allowed to change upper-case letters to lower-case letters or vice-versa.
2. You may define any new functions you like in the structure besides those mentioned in the signature.
3. Your program should work with the given signature.
4. You need to think of the *most efficient way* of implementing the various functions given in the signature so that the function results satisfy their definitions and properties.
5. Since the evaluator is going to look at your source code before evaluating it, you must explain your algorithms in the form of ML comments, so that the evaluator can understand what you have implemented.
6. Do *not* add any more decorations or functions or user-interfaces in order to impress the evaluator of the program. Nobody is going to be impressed by it.
7. There is a serious penalty for copying. If it is felt that there is too much similarity in the code between any two persons, then both are going to be penalized equally. So please set permissions on your directories, so that others cannot copy your programs.