Assignment 2

COMP 2411, Session 1, 2004

Starred questions are harder.

Q1 (2 marks): Using Natural deduction, show that $\neg(\varphi \rightarrow \psi)$ is a logical consequence of $\{\varphi \wedge \neg\psi\}$. You have to use Natural deduction, not truth tables or semantic tableaux.

Q2 (2 marks): Consider the Hilbert-style proof system studied in lectures where Axiom 3 is replaced by:

$$\neg\varphi \rightarrow \varphi \rightarrow \psi$$

(with still modus ponens as the only inference rule). Call $\mathcal{H}$ the resulting proof system. Show that every logical consequence that can be proved in $\mathcal{H}$ can also be proved in intuitionistic logic. (Actually, $\mathcal{H}$ is equivalent to intuitionistic logic.)

Q3 ($\star$) (2 marks): Using Natural deduction, show that $(\varphi \rightarrow \psi) \vee (\psi \rightarrow \varphi)$ is valid. You have to use Natural deduction, not truth tables or semantic tableaux.

For questions 4 and 5, place your complete solution for *both* problems (a modified version of the provided `assignment2.pl`) in a *single* file and submit this file using the following command:

```
$ give ass2p2 assignment2.pl
```

The file for the assignment can be downloaded from:

```
http://www/~cs2411/assignments/assignment2.pl
```

Your assignment solution will be tested using the version of SWI-Prolog installed on the lab computers on the submission day. All marks for questions 4 and 5 will be from machine marking.

Q4 (2 marks): You will probably spend more time reading this question than answering it...

Consider the following alternative syntax for propositional formulas, given by the smallest set $\mathcal{F}$ such that:

- all literals belong to $\mathcal{F}$;
- for all (possible empty) finite subsets D of $\mathcal{F}$, $\bigvee D$ and $\bigwedge D$ belong to $\mathcal{F}$.

For instance p, $\neg p$, $\bigvee\{p, \neg q, r\}$, $\bigwedge\{p, \bigvee\{p, \neg r\}, \bigvee \emptyset\}$ are all members of $\mathcal{F}$. Note that this particular syntax imposes that negation be only applied to propositional atoms.

Semantically, $\bigvee D$ represents the disjunction of the members of D and $\bigwedge D$ represents the conjunction of the members of D. Hence:

- a member of $\mathcal{F}$ of the form $\bigvee D$ is true iff some member of D is true;
- a member of $\mathcal{F}$ of the form $\bigwedge D$ is true iff all members of D are true.

For instance, $\bigvee\{p, \neg q, r\}$ is true iff p is true or q is false or r is true; $\bigwedge\{p, \neg q, r\}$ is true iff p is true and q is false and r is true. Note that in particular, $\bigvee \emptyset$ is unsatisfiable (it represents False) and $\bigwedge \emptyset$ is valid (it represents True).

We inductively define for all $\varphi \in \mathcal{F}$ a member of $\sim \varphi$ of $\mathcal{F}$ as follows:

- if φ is a propositional atom then $\sim \varphi = \neg\varphi$;
- if φ is of the form $\neg\psi$ then $\sim \varphi = \psi$ (note that ψ is necessarily a propositional atom);
- if φ is of the form $\bigvee D$ then $\sim \varphi = \bigwedge\{\sim \psi \mid \psi \in D\}$;
- if φ is of the form $\bigwedge D$ then $\sim \varphi = \bigvee\{\sim \psi \mid \psi \in D\}$.

For instance, $\sim \bigwedge\{p, \bigvee\{p, \neg r\}, \bigvee \emptyset\} = \bigvee\{\neg p, \bigwedge\{\neg p, r\}, \bigwedge \emptyset\}$.

We use the usual binary boolean operators as abbreviations. A member of $\mathcal{F}$ is an abbreviation for itself. Let $\varphi, \psi \in \mathcal{F}$ have abbreviations $\varphi^\star, \psi^\star$, respectively. Then:

- $\varphi^\star \vee \psi^\star$ and $\psi^\star \vee \varphi^\star$ are abbreviations for $\bigvee\{\varphi, \psi\}$ (*e.g.*, $p \vee p$ is an abbreviation for $\bigvee\{p\}$);
- $\varphi^\star \wedge \psi^\star$ and $\psi^\star \wedge \varphi^\star$ are abbreviations for $\bigwedge\{\varphi, \psi\}$ (*e.g.*, $p \wedge p$ is an abbreviation for $\bigwedge\{p\}$);
- $\varphi^\star \rightarrow \psi^\star$ is an abbreviation for the formula $\sim \varphi \vee \psi$ is an abbreviation of;
- $\varphi^\star \leftrightarrow \psi^\star$ is an abbreviation for the formula $(\varphi \rightarrow \psi) \wedge (\psi \rightarrow \varphi)$ is an abbreviation of.

The file `assignment2.pl` contains a (big) fragment of a (small) Prolog program that contains a partially defined procedure `standardize/2`. The goal `standardize(+AbbreviatedFormula, -Formula)` is meant to succeed iff `AbbreviatedFormula` (provided as input) is an abbreviation of `Formula` (obtained as output). The file `assignment2.pl` also contains a totally defined procedure `standardize_test(+AbbreviatedFormula, -Formula)` to test the previous procedure and print out `Formula` over many lines with appropriate indenting to better reflect its structure (the procedures for writing `Formula` are also totally defined).

The program already has some rewriting rules for `AbbreviatedFormula`. For instance you can already ask the following queries and get the following answers:

```
?- standardize_test(\/ [p, q, r, s]).
\/ [p, q, r, s]

Yes
?- standardize_test(\/ [p1 v q1 v r1, p2 v q2 v r2, p3 v q3 v r3, p4 v q4 v r4]).
\/ [\/ [p1,
        \/ [q1, r1]],
    \/ [p2,
        \/ [q2, r2]],
    \/ [p3,
        \/ [q3, r3]],
    \/ [p4,
        \/ [q4, r4]]]

Yes
```

Complete the implementation of `standardize(+AbbreviatedFormula, -Formula)` to make the program work with `AbbreviatedFormula` representing an arbitrary abbreviation of a member of $\mathcal{F}$. Your program could produce for instance:

```
?-  standardize_test(\/ [p1, /\ [p2, ~p3]]).
\/ [p1,
    /\ [p2, ~p3 ]]

Yes
?- standardize_test(p -> r).
\/ [r, ~p ]

Yes
?- standardize_test((~p -> ~q) -> q -> p).
\/ [/\ [q, ~p ],
    \/ [p, ~q ]]

Yes
?- standardize_test(~ ~p <-> (p <-> q v r)).
/\ [\/ [p,
        \/ [/\ [p,
                /\ [~q, ~r ]],
            /\ [~p,
                \/ [q, r]]]],
    \/ [/\ [\/ [p,
                /\ [~q, ~r ]],
            \/ [~p,
                \/ [q, r]]],
        ~p]]

Yes
```

Observe that in this program, ~ represents (without risk of confusion) both ¬ and ∼.

Your program will be tested using the `standardize_test` predicate.

Q5 (⋆) (2 marks): The previous syntax significantly simplifies the construction of semantic tableaux. The end of `assignment2.pl` contains a procedure `tableau_test/1` that tests the systematic construction of semantic tableaux for formulas converted to that syntax. It also contains a procedure `write_tableau`. Here are two possible queries and outputs that illustrate the former procedure.

```
:- tableau_test((p -> q) -> (r v p ) -> (r v q)).

\/ [/\ [p, ~q], \/ [/\ [~p, ~r], \/ [q, r]]]
    /\ [p, ~q]
        p, ~q   Open
    \/ [/\ [~p, ~r], \/ [q, r]]
        /\ [~p, ~r]
            ~p, ~r   Open
        \/ [q, r]
            q   Open
            r   Open

Yes

:- tableau_test(~(p v q) ^ ~ ~(p v q) v /\ []).

\/ [/\ [], /\ [/\ [~p, ~q], \/ [p, q]]]
    /\ []   Open
    /\ [/\ [~p, ~q], \/ [p, q]]
        /\ [~p, ~q], \/ [p, q]
            ~p, ~q, \/ [p, q]
                p, ~p, ~q   Closed
                q, ~p, ~q   Closed

Yes
```

Implement the procedures `create_tableau/2` and `write_tableau/1` involved in the definition of `tableau_test`. Note that formulas are displayed over one line only, thanks to a procedure that is also implemented in `assignment2.pl`. To solve this question, you will probably use and modify the procedure `extend_systematic_tableau` in the file `tableau.pl` that can be downloaded from the course's web page.

Your program will be tested using the provided `tableau_test` predicate.