CS206 Homework #1

Max marks: 75 **Due Feb 7, 2006**

- *Be brief, complete and stick to what has been asked.*
- ***Do not copy solutions from others.***

1. *[5 + 10 + 10 marks]* In this question, we will reason about a proof system for propositional logic assuming that the only propositional connective is $\rightarrow$, and that the only propositional constant is $\perp$. For example, if x, y, z are propositional variables, then $(x \rightarrow (y \rightarrow \perp)) \rightarrow (\perp \rightarrow z)$ is a propositional logic formula using only the allowed connective and constant.

   (a) Let ϕ_1 and ϕ_2 be propositional logic formulae using $\rightarrow$ as the only connective and $\perp$ as the only constant. Give *semantically equivalent* formulae for $\phi_1 \wedge \phi_2$ and $\neg\phi_1$, such that $\rightarrow$ is the only connective and $\perp$ is the only constant in the resulting formulae. You must give justification for your claim of semantic equivalence.

   (b) Your solution to the previous subquestion should convince you that any propositional logic formula can be converted to a semantically equivalent one using only $\rightarrow$ and $\perp$. A student now claims that it is possible to prove sequents in this version of propositional logic (with $\rightarrow$ as the only connective and $\perp$ as the only constant) using rules $\rightarrow_i$, $\rightarrow_e$, $\perp_e$ of the natural deduction system studied in class, in addition to the following special rule, called $(\rightarrow \perp)_e$ rule:

   $$(\rightarrow \perp)_e : \ \frac{(\phi \rightarrow \perp) \rightarrow \psi,\ \phi \rightarrow \zeta,\ \psi \rightarrow \zeta}{\zeta}$$

   Using only the above four rules, prove the following sequent:
   $(\phi \rightarrow \perp) \rightarrow \psi,\ \phi \rightarrow \zeta \ \vdash\ (\psi \rightarrow \perp) \rightarrow \zeta$

   (c) Do you think a proof system with only the above four rules, i.e. $\rightarrow_i$, $\rightarrow_e$, $\perp_e$ and $(\rightarrow \perp)_e$, is complete for the version of propositional logic that uses $\rightarrow$ as the only binary connective and $\perp$ as the only constant? In other words, given two formulas ϕ and ψ, each involving only $\rightarrow$ and $\perp$, such that $\phi \models \psi$, is it always possible to prove the sequent $\phi \vdash \psi$ using only the above four rules? Answers without justification will fetch zero marks.

2. *[10 + 10 + 10 marks]* Use natural deduction to prove the following sequents. You must use only the basic rules of natural deduction (no derived rules, including LEM, are allowed). Your proof using the basic rules must not exceed the number of steps mentioned alongside each sequent. The number of steps includes the statement of the premises. You must also annotate each step of your proof with the basic rule applied at that step.

   (a) $\phi \vee \psi,\ \neg\phi \vee \psi \ \vdash\ \psi$ *[within 15 basic steps].*
   This rule is also popularly known among logicians as the "Resolution Rule".

   (b) $\phi \vee \psi,\ \neg(\phi \wedge \psi) \ \vdash\ (\phi \wedge \neg\psi) \vee (\psi \wedge \neg\phi)$ *[within 19 basic steps]*

   (c) $\neg(\phi \wedge \neg\psi),\ \neg(\psi \vee \neg\phi) \ \vdash\ \phi \wedge \psi$ *[within 16 basic steps]*

3. *[20 marks]* In this question, we wish to investigate how to build a naive *automated propositional prover*. Given a set of formulae $\{\psi_1, \ldots, \psi_n, \phi\}$, our prover's task is to either show that $\psi_1, \ldots, \psi_n \vdash \psi$ or demonstrate that $\psi_1, \ldots, \psi_n \not\models \psi$

   Let $P = \{p_1, \ldots p_k\}$ be the set of propositions in $\{\psi_1, \ldots \psi_n, \phi\}$. Let $O = \{\vee, \wedge, \neg, \rightarrow, (,)\}$ be the set of symbols representing operators and connectives in propositional logic. We will assume that P is *disjoint* from O, and does not contain any numeral, i.e. symbol in $\{0, \ldots 9\}$, nor any symbol in { `#`, `[`, `]`} (which we will soon need for special purposes).

   Every propositional logic formula over P is a concatenation of symbols, or *string*, from $P \cup O$, constructed according to the syntax studied in class. A possible *proof*, as studied in class, is a sequence of propositional logic formulae along with some annotations. For purposes of this question, we will assume that each formula ϕ in the proof is annotated as follows:
   `## num0 # opt_[` ϕ `opt_] # rule_num # num1 # ...  # numk ##`
   In the above annotated formula, `num0` is a non-negative integer used to refer to the current formula ϕ, `opt_[` is an optional `[` denoting the start of scope of an assumption, and `opt_]` is an optional `]` denoting the end of scope of the last assumption whose scope is not yet closed. We assume that the small set of natural deduction rules (and-introduction, or-elimination, etc.) studied in class are indexed, so that we can refer to a rule by its rule number. Rule numbers are assumed to start from 1. In the annotated formula, `rule_num` is a number indicating which rule is used to infer ϕ, and `num1, ...  numk` are numbers used to refer to other formulae from which the current formula ϕ is obtained by application of rule `rule_num` . If ϕ is a premise or an assumption, we use the special rule number 0, i.e. `rule_num` $= 0$, without any `num1, ...  numk`. Note that every annotated formula begins and ends with `##`, and the various parts of the annotation and the formula itself are separated from each other by `#`. Such a format makes it easy to write a parser that can separate the different parts of an annotated formula.

   Using the above notation, a potential proof is a string over $P \cup O \cup \{0, \ldots 9\} \cup$ { `[,], #`}. Of course, not every string over $P \cup O \cup \{0, \ldots 9\} \cup$ { `[,], #`} constitutes a valid natural deduction proof. There are additional checks to be made (and you must figure these out) before a string w can be considered a valid natural deduction proof of $\psi_1, \ldots \psi_n \vdash \phi$.

   As examples, the following is a valid natural deduction proof of $a \vdash b \rightarrow a$, where the set of propositions is $P = \{a, b\}$, rule 1 is the copy rule (i.e. any formula that is not within a closed box can be copied), and rule 5 is the implication introduction rule:
   `## 10 # a # 0 ## 2 # [ b # 0 ## 5 # a ] # 1 # 10 ## 6 # b -> a # 5 # 2 # 5 ##`
   However, the following is not a valid proof in natural deduction.
   `## 10 # a # 0 ## 2 # b -> a # 5 # 10 ##`

   In building our naive automated prover, we are allowed to use a library of functions, with the following descriptions.

   (a) Functions `InitGenerateString()` and `GenerateNextString()`: None of these functions take any argument. Successive invokations of `GenerateNextString()` generate the strings over $P \cup O \cup \{0, \ldots 9\} \cup$ { `[,], #`} in lexicographic order. Each invokation of `GenerateNextString()` returns a string that is lexicographically next to the string generated by the last invokation of `GenerateNextString()`. The first invokation of `GenerateNextString()` after invoking `InitGenerateString()` returns the lexicographically first string over $P \cup O \cup \{0, \ldots 9\} \cup$ { `[,], #`}.

   (b) Function `CheckSyntax(w)`: Takes a string, w, over $P \cup O \cup \{0, \ldots 9\} \cup$ { `[,], #`} as input, and outputs True if w is the concatenation of one or more annotated propositional logic formulae, and False otherwise.

(c) Function `PropSATSolve(r, f1, ... fr)`: Takes a positive integer r, and r formulae $f_1, \ldots f_r$ as inputs and returns an empty set if the conjunction of the f_i's is unsatisfiable. Otherwise, it returns a set of pairs $\{(p, t_p) \mid p \text{ is a proposition in } f, t_p \in \{\mathsf{True}, \mathsf{False}\}\}$ giving a satisfying assignment of the conjunction of the f_i's.

(d) Function `GeneratePremises(r, f1, ... fr)`: Takes a positive integer r, and r formulae $f_1, \ldots f_r$ as inputs and returns the string `## 1 # f1 # 0 ## 2 # f2 # 0 ... ## r # fr # 0 ##`. Note that this string is a valid (though useless) proof in natural deduction containing only the premises (recall that `rule_num = 0` means that the formula is a premise or assumption).

(e) Function `Concatenate(w1, w2)`: Takes two strings w_1 and w_2 over $P \cup O \cup \{0, \ldots 9\} \cup \{$ `[`,`]`, `#`$\}$ and returns the string $w_1 w_2$ obtained by concatenating w_2 to w_1.

(f) Function `IsEmptySet(S)`: Takes a set S as input, and returns True if the set is empty, else it returns False.

(g) Function `Parse(w)`: Takes a string, w, over $P \cup O \cup \{0, \ldots 9\} \cup \{$ `[`,`]`, `#`$\}$ as input and parses it. It returns 0 if `CheckSyntax(w)` returns False. Otherwise, it returns the number of annotated formulae in w. In the latter case, `Parse(w)` also has the side effect of filling in seven global arrays, `RefNum`, `OpenAssume`, `Form`, `CloseAssume`, `Rule`, `AntecNum` and `Antec` as follows. If `## num0 # opt_[` f `opt_] # rule_num # num1 # ... # numk ##` is the i^{th} annotated formula in w, then `num0` is stored in `RefNum[i]`, `f` in `Form[i]`, `rule_num` in `Rule[i]`, `k` in `AntecNum[i]`, and an array of k integers in `Antec[i]`. For $j \in \{1, \ldots k\}$, the number `numj` is stored as the j^{th} integer in array `Antec[i]`, i.e. `Antec[i][j] = numj`. All the arrays are assumed to be indexed from 1 upwards. In addition, if the optional `[` is present before f, `OpenAssume[i]` is set to True, else it is set to False. Similarly, if the optional `]` is present after f, `CloseAssume[i]` is set to True, else it is set to False.

(h) Function `CheckRule(i)`: Takes a non-negative index i, and returns True if the formula in `Form[i]` is indeed obtained by applying `Rule[i]` to the `AntecNum[i]` formulae with reference numbers `Antec[i][1] ... Antec[i][AntecNum[i]]`. Otherwise, it returns False.

Using the above library of functions, describe in C-style pseudocode how you would design an automated propositional prover that takes as input a set of propositional logic formulae $\{\psi_1, \ldots \psi_n, \phi\}$ and constructs a proof of $\psi_1, \ldots \psi_n \vdash \phi$ using natural deduction, if such a proof exists. Otherwise, it returns an assignment of True/False values to the propositions in $\{\psi_1, \ldots \psi_n, \phi\}$ that demonstrate the impossibility of a proof of $\psi_1, \ldots \psi_n \vdash \phi$

Briefly justify why your pseudocode gives the correct answer for any set of formulae $\{\psi_1, \ldots \psi_n, \phi\}$, with at least one premise formula ψ_1 and one inferred formula ϕ.