COMP 3400 Computational Logic and Automated Reasoning
Winter 2017 Assignment 5 - Part 1

Instructions:

1. **For your solution use the template file that was posted on the course news, and follow the instructions in it.**

   In particular: (a) Include at the top of the first page: full name, student number, and email address. (b) Assignments have to be created with Latex, and submitted in pdf format. (c) Every problem solution MUST include the problem statement. The source file for this assignment is provided.

   Latex has to be used as such, not as you would use a text editor, such as Notepad. In particular, formulas have to be written using Latex's mathematical features, and then compiled.

2. Assignments are individual, no groups.

3. Submit by email to the instructor, with "Assignment "Number", CompLog" in the subject. **Include your last name in the file name!** For example, in the subject: "Assig. 5 CompLog". The file name: "bertossi-5.pdf".

   Only a single pdf file will be accepted as submission. No tar or zip files (or anything like that), please. Keep your Latex source files in case you are requested to show them.

4. Explain your solution very carefully, but still be succinct with your answers. No unnecessary verbose arguments, please. Go to the point.

   Make explicit all your assumptions.

5. **Not following the instructions above or the solution template file will make you lose points.**

1. [50%] (a) Use Prolog with predicates and variables (not propositional Prolog) to decide for an **arbitrary finite string** φ over the alphabet $\{\wedge,), \neg, (, \vee, \rightarrow, \leftrightarrow, p, q, r, s, t, ...\}$ (the latter stand for propositional variables) is a well-formed propositional formula. Use a unary predicate for this, say $Decide(\cdot)$.

For example, the program should return *yes* when posed a query about $((p \wedge q) \vee \neg r)$, say something like: `:- Decide("(p ^ q) v -r)")?`

But *no* when asked about $\neg(\vee p \wedge q)$, say `:- Decide("-(v p ^ q)")?`

The quotes above mean that those formulas have to be properly represented, see the enumeration below for alternatives (among others).

Hint: Recall the inductive definition of propositional formula given in Chapter 3.

There are at least three different ways to represent formulas (choose one and stick to it for the rest of this problem):

i. Strings (and formulas) can be represented as lists, e.g. $((p1 \vee \neg p3) \wedge p4)$ becomes the list $[(, (, p1, \vee, \neg, p3,), \wedge, p4,)]$.

   You may use the list manipulation predicates presented at the end of Chapter 8 or use the "list library".

ii. You can represent formulas as Boolean algebra terms in prefix notations, e.g. $((p1 \vee \neg p3) \wedge p4)$ becomes: $\wedge(\vee(p1, \neg(p3)), p4)$.

iii. You can define operators in Prolog, e.g. something like (depending on the Prolog implementation, see the manual) this for the negation and the conjunction, resp.:

```
:- op(510,fx,[~]).
:- op(520,xfy,[/\]).
```

   (The first with one argument, the second with two.)

Run your program to decide a couple of well-formed formulas and a couple of non-legal formulas. Clearly indicate the formulas (written in Latex, as usual) you run the program on. As usual, explain your whole methodology, inputs, outputs, and append the execution traces.

(b) As an extension of part (a), use Prolog to define a binary predicate that returns the length of a formula. Also, compute the lengths of the well-formed formulas you used in part (a). Same instructions as for part (a).

Hint: Again, you have to use the inductive definition of propositional formula.

(c) As an extension of (a), use Prolog to define a binary predicate that returns in an output argument the list of propositional variables in the formula, e.g. it should be:
$PV([(, (, p1, \vee, \neg, p3,), \wedge, p4,)], [p1, p3, p4])$. Do this with the formulas used in (a) and (b).

Hint: Again, you have to use the inductive definition of propositional formula.

2. [50%] In class you saw that the natural numbers can be defined from scratch with Prolog, by using the constructors 0 and $\sigma(\cdot)$. Actually, all you need is an arbitrary constant, say a, and a unary operation symbol (functor), say $g(\cdot)$, and you can generate an isomorphic structure to that generated with 0 and $\sigma(\cdot)$.

The same can be done with lists, they can be defined from scratch from logic (and Prolog). So, use a constant, say *nil*, for the empty list, and a binary operation $cons(\cdot,\cdot)$ (again, the chosen names are not important). Prolog provides lists and their management as built-ins, but everything can be done from scratch.

(a) Define lists, i.e. a general predicate $list(\cdot)$ that is true with (finite) lists over the alphabet $A = \{a, b, c, d, e, f, g, h\}$. For example, the list $bdeh$, with first element b and last h, should be represented by the term $cons(b, cons(d, cons(e, cons(h, nil))))$, and $list(cons(b, cons(d, cons(e, cons(h, nil)))))$ should be true. Use Prolog to verify that $cons(b, cons(d, cons(e, cons(h, nil))))$ is a list, but not $cons(b, cons(d, e), cons(h, nil))))$.

(b) Define a binary predicate $sublist(\cdot,\cdot)$, such that $sublist(L_1, L_2)$ becomes true when L_1 is a sublist of L_2. Use Prolog to verify (with the representation in (a)) that ab is a sublist of $efabde$. But fd is not a sublist of the latter.

(c) Use the representation in (a) to define the length of a list. It should be $length(efabde, 6)$ true. Use Prolog to verify this. (You can use Prolog's built-in numbers and arithmetic.)

The whole assignment has to be submitted as a single PDF file, including the traces of your runs.

Deadline: April 2, at 23:55