MSc Computing Science
Imperial College London

Prolog Assessed Exercise
November 2017

You are required to:
Submit an extension of the program in the Prolog file '**nat.pl**' according to the specifications (a)-(c) below.

It is very important that you use predicates exactly as specified (spelling, lower and upper case letters and the number and position of the arguments). Your answers will be auto-tested.

You will lose substantial marks if your programs cannot be auto-tested, because you have submitted wrong files or you have used wrong predicate names. Please copy and paste the given predicate names from the specifications below or from the supplied 'nat.pl' file into your program to avoid misspelling.

Of course, you can and should define your own auxiliary (helper) predicates in order to reduce the size of predicate definitions.

Please write comments to explain your program.
Marks will be deducted for lack of clarity of code and bad formatting.

You can use `findall/3`, `setof/3` and Prolog conditional and negation operators, if needed, together with any comparison primitives you may need (e.g. <, >), and list predicates `member/2`, `length/2`, `append/3`, but no other built-in predicates, unless otherwise permitted in a question. Do not use the cut, `!`.

The provided file 'nat.pl' contains a program for natural language processing. This assessed work extends that program. In the file you are given a predicate `sentence/1` that constructs and recognizes sentences based on the following simple grammar:

> A *sentence* is made up of a *noun phrase* followed by a *verb phrase*.
> A *noun phrase* is made up of an *article* followed by *noun*.
> A *verb phrase* is made up of either a *verb*, or a *verb* followed by a *noun phrase*.

The auxiliary predicates include `noun_phrase/1` and `verb_phrase/1`.
Sentences are represented as lists of words, *e.g.* "`[the, cow, chews, the, grass]`".

You are also given a file 'lexicon.pl' which contains the following:

> Article: *the*
> Nouns: *grass, cow, boy, girl, song*
> Verbs: *eats, chews, kicks, sings*

The nouns *cow, boy, girl* are specified as animate.

To test your program in Sicstus, simply load (or compile) both 'nat.pl' and 'lexicon.pl'. **Do not load 'lexicon.pl' from the program 'nat.pl', nor add any facts or rules about the lexicon (`noun/1`, `article/1`, `verb/1` and `animate/1`) to the file 'nat.pl'**. However, feel free to modify 'lexicon.pl' as you wish. Note that your 'nat.pl' program should work with any valid lexicon.

a) Assume that a conjunction of sentences is a list such as "`[the, boy, chews, the, grass, and, the, cow, kicks, the, boy]`" where each sentence follows the rules of grammar above, and each two adjacent sentences are separated by the word "`and`". A conjunction of sentences can have one or more sentences.

   Write a program `conj(Text)` that recognizes conjunction of sentences, *i.e.* answers `'yes'` if `Text` is a conjunction of sentences and answers `'no'` otherwise. You can assume that in all calls to `conj/1`, `Text` is given to you (*i.e.* is ground).

   Test your program with the following queries:

```
| ?- conj([the, cow, chews, the, grass]).
yes

| ?- conj([the, boy, eats, and, the, girl, sings, and, the, cow, kicks]).
yes

| ?- conj([the, boy, eats, and, sings]).
no
```

b) Write a program for `encode(T, ET)`, where `T` is some text (list of words, not necessarily a sentence or a conjunction of sentences) and `ET` is an encoding of `T` such that every noun `N` in `T` is replaced by a constant `C` constructed as follows.

   `C` is 3 characters long and is made up as follows:

   - The first character of `C` is `'a'` if the noun is animate, otherwise it is `'d'`.
   - The second character of `C` is `'l'` (for *long*) if `N` is longer than 3 letters and `'s'` (for *short*) otherwise.
   - The third character of `C` is the first letter of `N`.

   So, for example, `'grass'` is replaced by `'dlg'`, `'cow'` is replaced by `'asc'`, and `'song'` is replaced by `'dls'`.

   You can assume that in all calls to `encode/2`, `T` is given to you (*i.e.* is ground).

   Hint: Prolog has a built-in predicate `atom_chars/2` such that `atom_chars(W, L)` succeeds when `L` is the list of characters in the constant `W` in the order in which they appear in `W`. So `atom_chars(cow, L)` succeeds with `'L = [c,o,w]'` and `atom_chars(X, [c, o, w])` succeeds with `'X = cow'`.

Test your program with the following queries:

```
| ?- encode([the, girl, chews, grass, and, the, boy, kicks, cow], X).
X = [the,alg,chews,dlg,and,the,asb,kicks,asc] ; no

| ?- encode([sings, kicks, eats], X).
X = [sings,kicks,eats] ; no
```

c) Write a program for `same_actor(Text)` that answers `'yes'` when all the sentences in the conjunction of sentences `Text` refer to the same actor and answers `'no'` otherwise. An actor in a sentence is the noun immediately preceding the verb in the sentence. You can assume that in all calls to `same_actor/1`, `Text` is given to you (*i.e.* is ground) and is a conjunction of sentences.

Test your program with the following queries:

```
| ?- same_actor([the, boy, chews, the, grass]).
yes

| ?- same_actor([the, boy, chews, the, grass, and, the, boy, kicks, the, cow]).
yes

| ?- same_actor([the, boy, chews, the, grass, and, the, girl, kicks, the, boy]).
no
```

The three parts carry 20%, 50% and 30% of the marks respectively.

Assessment Notes:

- Programs should work.
- Programs should not loop, but apart from that efficiency is not important.
- Your program should not contain any specific lexicon (*i.e.* do not define/redefine the predicates `noun/1`, `article/1` and `verb/1` in 'nat.pl').
- Your programs should work if we change the lexicon.
- You should use the predicates exactly as specified in the questions (spelling, lower and upper case letters and the number and position of the arguments).
- Clarity – use comments to explain your predicates.
- Format your program clearly, so it is easily readable.
- Use good Prolog style – clear short rules.
- Your solutions will be tested using Sicstus Prolog under Linux, so make sure that you use Sicstus (not SWI or any other Prolog) and that your code runs on the Lab Machines.
- 80% of the marks are allocated by auto-testing.
- 20% of the marks are allocated to programming style, including among others:
  - Clear and helpful commenting.
  - No long predicate definitions. In case a clause gets too long, try to split it into auxiliary predicates.
  - Readable code: indentation, sensible naming, *etc*.