MSc Assessed Prolog Lab Exercise 2

Issued: 1st December 2015

Due: 15th December 2015

Start with the file natLang.pl provided, extend the program according to all the specifications given below, and submit it.

The starter file natLang.pl defines a predicate *sentence/1* and auxiliary predicates, such that *sentence(S)* succeeds if *S* is a correct grammatical sentence, according to the following grammar:

Sentence consists of a *noun phrase* followed by a *verb phrase*.

Noun phrase is an *article* followed by a *noun*.

Verb phrase is 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 a list of words, e.g. *[the, boy, eats, an, apple]*.

The lexicon includes the following:

Nouns: grass, cow, girl, boy, apple, song,

Articles: the, a, an,

Verbs: eats, sings, chews, kicks.

Some example queries and answers:

?- sentence([a, cow, eats, the, grass]).	gives the answer yes.
?- sentence([the, girl, eats, an, apple]).	gives get the answer yes.
?- sentence([the, boy, sings]).	gives the answer yes.
?- sentence([the, girl, sings, eats]).	gives the answer no.
?- sentence([girl, sings, a, song]).	gives the answer no.
?- sentence(S).	gives all instances of S that are grammatically correct. You will get many ridiculous sentences ([a, grass, eats, a, cow]), because the grammar is very simple – don't worry!

Q1. Assume that a conjunction of sentences is a list such as

[the, boy, chews, an, apple, and, the, girl, kicks, the, boy] where each sentence follows the rules of grammar above and each two adjacent sentences are separated by the word "*and"*.

a) Write a program for *count_sentences*(*Text, Count),* where *Text* is a conjunction of sentences, and *Count* is the number of sentences in *Text*. In all calls you can assume that *Text* is ground.

Test your program with the following queries:

count_sentences([the, boy, chews, an, apple], X). answer X=1

count_sentences([the, boy, chews, an, apple, and, the, girl, kicks, the, boy], X). answer X=2

count_sentences([the, girl, chews, an, apple, and, the, girl, sings, a, song, and, the, girl, kicks, the, cow], X). answer X=3

b) Write a program for *actions*(*Actor, Text, As)* such that *As* is the list of actions that the *Actor* does according to the *Text*. You do not need to check that *Text* is a conjunction of sentences. You can assume that it is in all calls to *actions*.

Test your program with the following queries:

actions(boy, [the, boy, chews, an, apple, and, the, girl, kicks, the, boy], X).

answer X=[chews].

actions(apple, [the, boy, chews, an, apple, and, the, girl, kicks, the, boy], X).

answer X=[].

actions(girl, [the, girl, chews, an, apple, and, the, girl, sings, a, song, and, the, girl, kicks, the, cow], X).

answer X=[chews, sings, kicks]. The order of the elements in the list is not important.

Q2. Modify the program in the starter file natLang.pl for *noun_phrase/1* to a program *noun_phrase_better/1* so that it requires the article to match the noun in noun phrases, i.e. if the noun starts with a vowel then the article is either 'the' or 'an', not 'a', and if the noun starts with a consonant then the article is either 'the' or 'a', not 'an'. So, for example, [a, apple] and [an, cow] should not be recognized or generated as a noun phrases.

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(apple, L)* succeeds with *L=[a,p,p,l,e]*.

Test your program with the following queries:

```
| ?- noun_phrase_better(X).

| ?- noun_phrase_better([a, apple]).

| ?- noun_phrase_better([an, apple]).

| ?- noun_phrase_better([an, cow]).

| ?- noun_phrase_better([a, X]).
```

Q3.

a) Define a predicate *cadvs/1* such that *cadvs(L)* means *L* is conjunction of adverbs according to the specification below.

Extend the grammar so that your program can also deal with verb phrases that consist of a conjunction of adverbs, followed by a verb, followed by a noun phrase. An example of a sentence conforming to this extended grammar is “a cow slowly and deliberately chews the grass”. When there are more than two adverbs, all but the final two are separated by commas, and only the final two are separated by the word “and”. So, for example [slowly, ‘,’, deliberately, and, merrily] is recognized as a conjunction of adverbs, but not [slowly, and, deliberately, and, merrily].

Also make sure that your program does not allow repetitions of adverbs in the same phrase. So, for example, a sequence of words such as [a, cow, slowly, and, slowly, chews, the, grass] is not accepted or generated as a sentence.

Hint: Use an accumulator to keep track of the adverbs used or seen so far, so you do not repeat them.

b) Add the lexicon:

adverb([slowly]).

adverb([deliberately]).

adverb([merrily]).

adverb([sweetly]).

Test your program with queries including the following:

cadvs([slowly,',', merrily, and, sweetly]).	Answer: yes
cadvs([slowly, and, merrily, and, sweetly]).	Answer: no
cadvs([slowly,',', merrily, and, slowly]).	Answer: no
cadvs([slowly,',', merrily, ',', sweetly]).	Answer: no
cadvs(X).	

c) Define a predicate *verb_phrase_better/1* such that *verb_phrase_better(V)* means *V* is a verb phrase, where a verb phrase is now:

a verb, or

a verb followed by a *better* noun phrase, according to the specification in (Q2), above, or

a conjunctions of adverbs followed by a verb, or

a conjunctions of adverbs followed by a verb, followed by a *better* noun phrase, according to the specification in (Q2), above.

Test your program with queries including the following:

verb_phrase_better([slowly,',', merrily, and, sweetly, sings]).	yes
verb_phrase_better([slowly,',', merrily, and, sweetly, sings, a, song]).	yes
verb_phrase_better([sings, a, song]).	yes
verb_phrase_better([sings, an, song]).	no
verb_phrase_better([slowly,',', merrily, and, slowly, sings, a, song]).	no

d) Define a predicate *sentence_better/1* where a better sentence is a better noun phrase (Q2) followed by a better verb phrase (Q3c).

Test your program with queries including the following:

sentence_better([a, boy, slowly,',', merrily, and, sweetly, eats, a, apple]).	no
sentence_better([a, boy, slowly,',', merrily, and, sweetly, eats, an, apple]).	yes
sentence_better([a, apple, slowly,',', merrily, and, sweetly, eats, an, apple]).	no

You can use any additional auxiliary predicates.

Parts 1- 3 have 30%, 20%, 50% of the marks, respectively.

Assessment Notes:

- Programs should work.
- Programs should not loop, but apart from that efficiency is not important.
- Clarity – use comments to explain your predicates.
- Format your program clearly, so it is easily readable.
- Use good Prolog style – clear short rules.
- You probably do not need cuts. If you do use cuts you should make sure legitimate answers are not removed.
- Please make sure your programs include the predicates and the lexicon mentioned above, as these are important for auto-testing.

- When you submit your program should have exactly the lexicon you have been given, no more and no less.
- Your programs should work if we change the lexicon.