Prolog Tick 2006

Submission Instructions

Please submit your work electronically to Student Admin as a single Prolog (.pl) file and a single trace file (.txt) attached to an email with the subject line "Prolog Tick". The trace file should be produced as shown in the lectures using the `trace` command.
You will also need to provide a paper printout for the submitted portfolio of practical work which must be submitted to Student Admin by 25 May 2007 as described here: http://www.cl.cam.ac.uk/UoCCL/teaching/exams/headofdeptnotices.pdf).

None of the predicates you write should produce extra, unwanted answers. None of the predicates need a cut.

Prolog Tick

This program reads in a sentence to check that its structure matches the grammar and its words are in the dictionary. In this tick, you will extend the program to understand more sentence structures, print out the resulting parse tree and dynamically add new words to the dictionary.

```
parse(List,S):- phrase(sentence(S), List).

sentence(s(NP,VP))        --> noun_phrase(NP),
verb_phrase(VP).
noun_phrase(np(D,N))      --> determiner(D),
noun_phrase2(N).

noun_phrase2(np2(N))      --> noun(N).

noun_phrase2(np2(A,N))    --> adjective(A),
noun_phrase2(N).

verb_phrase(vp(VP,PP))    --> verb_phrase2(VP),
prepositional_phrase(PP).
verb_phrase(vp(VP))       --> verb_phrase2(VP).

verb_phrase2(vp2(V,N))    --> verb(V), noun_phrase(N).
verb_phrase2(vp2(V))      --> verb(V).

prepositional_phrase(pp(PR,NP)) --> preposition(PR),
noun_phrase(NP).
```

```
% The lexical rules that make up the lexicon or
dictionary
verb(verb(X))    --> [X], { verb(X) }.
adjective(adjective(X))  --> [X], { adjective(X) }.
adverb(adverb(X))  --> [X], { adverb(X) }.
determiner(determiner(X))  --> [X], { determiner(X) }.
noun(noun(X)) --> [X], { noun(X) }.
pronoun(pronoun(X)) --> [X], { pronoun(X) }.
preposition(preposition(X))   --> [X], { preposition(X)
}.

% The lexicon
verb(read).
verb(write).
verb(sing).

adjective(quick).
adjective(tall).
adjective(small).

adverb(quickly).
adverb(slowly).

determiner(a).
determiner(the).

noun(singers).
noun(writers).
noun(poets).
noun(poem).
noun(song).
noun(story).

preposition(on).
preposition(by).

pronoun(he).
pronoun(she).
pronoun(they).
pronoun(we).
pronoun(you).
pronoun(it).
```

1. Interpreting ‘Yes’ and ‘No’ (15% marks)

Run English sentences as queries for `parse` and explain carefully what the ‘Yes’ and ‘No’ answers mean with appropriate test data.

2. Adding new rules (25% marks)

Modify the rules so that the following queries **succeed**

a. `parse([they ,sing, the, song],S).`

b. `parse([we, sing, quickly],S).`

whilst the following **fail**:

c. `parse([the,she, sing],S).`

d. `parse([the,green,she, sing],S).`

3. Printing out the parse tree (25% marks)

Looking at the definition for the `parse` predicate, we see that it uses the predicate `sentence` and the variable `B` to form the root of a tree.

```
parse(A,B) :- phrase(sentence(B),A).
```

If we use `phrase` on the command line, we can see the tree that is being built.
For example:

```
?- phrase(sentence(B), [the, singers, sing, the, song]).
B = s(np(det(the), np2(noun(singers))), vp(vp2(v(sing),
np(det(the), np2(noun(song))))))
```

Write a predicate `print_tree(S)` that produces the tree indented by five spaces and new line as shown below

```
s
     np
          det
               the
          np2
               noun
                    singers
     vp
          vp2
               v
                    sing
          np
               det
                    the
               np2
                    noun
                         song
```

Use the library predicates `=.. (univ)` and `atomic`, which work as shown below, and the predicate `nl` which writes out a new line.

```
?- node(left, right) =.. C.
C = [node, left, right];
No
?- atomic(value).
Yes
?-
```

4. Writing a simple shell to add new words (40% marks)

Write a simple shell using the top level predicate `parser` that prompts the user to define the type of the missing word *choir* and then performs the parse as before. The dialog should look like this:

```
?- parser[the, choir, sing, the, song],S).
What kind of word is choir: noun.
B = s(np(det(the), np2(noun(choir))), vp(vp2(v(sing),
np(det(the), np2(noun(song))))))
Yes
```

Use the library predicates `read, write` and `assert` introduced in the lectures.

Provide English sentences as test queries for **each** of the **seven** parts of speech that you have provided that succeed and another **three** valid English sentences which do not parse correctly.

End.