Homework 9 (extra credit) — Prolog — due Monday 29 April

Total number of points available on this homework is 100. Full credit is equivalent to 0 points.

9.1 Grammar specification language (100pts)

In class we looked at one way to represent a context-free grammar in Prolog. For instance, for the grammar of Exercise 8.2:

```
Grammar -->  Symbols_List
Symbols_List --> Symbol  Symbols_List
               | Symbol
Symbol --> Identifier  Right_Arrow  Productions_List
Productions_List --> Production
                     Productions_Separator Productions_List
                   | Production
Production --> Identifiers_List
Identifiers_List --> Identifier  Identifiers_List |
```

we used the following rules:

```
nGrammar(L,grammar(X)) :-
   nSymbols_List(L,X).

nSymbols_List(L,sl_many(X,Y)) :-
   append(S,SL,L),
   nSymbol(S,X),
   nSymbols_List(SL,Y).
nSymbols_List(L,sl_one(X)) :-
   nSymbol(L,X).

nSymbol([I,right_Arrow|PL],symbol(X,Y)) :-
   tIdentifier(I,X),
   nProductions_List(PL,Y).

nProductions_List(L,pl_many(X,Y)) :-
   append(P,[productions_Separator|PL],L),
   nProduction(P,X),
   nProductions_List(PL,Y).
nProductions_List(L,pl_one(X)) :- nProduction(L,X).
```

```
nProduction(L,production(X)) :-
   nIdentifiers_List(L,X).

nIdentifiers_List([I|IL],il_many(X,Y)) :-
   tIdentifier(I,X),
   nIdentifiers_List(IL,Y).
nIdentifiers_List([],il_none).

tIdentifier(identifier(X),i(X)).
```

Given these rules, we can issue a query such as:

```
?- nGrammar([identifier('Grammar'),
        right_Arrow,
        identifier('Symbols_List'),

        identifier('Symbols_List'),
        right_Arrow,
        identifier('Symbol'),
        identifier('Symbols_Separator'),
        identifier('Symbols_List'),
        productions_Separator,
        identifier('Symbol'),

        identifier('Symbol'),
        right_Arrow,
        identifier('Identifier'),
        identifier('Right_Arrow'),
        identifier('Productions_List'),

        identifier('Productions_List'),
        right_Arrow,
        identifier('Production'),
        identifier('Productions_Separator'),
        identifier('Productions_List'),
        productions_Separator,
        identifier('Production'),

        identifier('Production'),
        right_Arrow,
        identifier('Identifiers_List'),

        identifier('Identifiers_List'),
```

```
        right_Arrow,
        identifier('Identifier'),
        identifier('Identifiers_List'),
        productions_Separator
    ], G).
```

Note that in addition to recognizing the language, these rules build a parse tree (returned as `G` in the example query).

If we don't care about building a parse tree and just want a yes/no answer, we can use simpler rules:

```
nGrammar(L) :- nSymbols_List(L).
nSymbols_List(L) :- nSymbol(L).
nSymbols_List(L) :- append(S,SL,L),
                    nSymbol(S),
                    nSymbols_List(SL).
nSymbol([identifier(_),right_Arrow|PL]) :- nProductions_List(PL).
nProductions_List(L) :- nProduction(L).
nProductions_List(L) :- append(P,[productions_Separator|PL],L),
                        nProduction(P),
                        nProductions_List(PL).
nProduction(L) :- nIdentifiers_List(L).
nIdentifiers_List([]).
nIdentifiers_List([I|IL]) :- tIdentifier(I),
                             nIdentifiers_List(IL).
tIdentifier(identifier(_)).
```

The above Prolog programs use a representation in which each grammar nonterminal is mapped to a Prolog predicate.

The task in this exercise is to write a different Prolog program, in which grammar productions are represented as data, not as code.

For instance, the production

```
Symbol --> Identifier  Right_Arrow  Productions_List
```

might be represented as a Prolog fact:

```
production(symbol,[identifier,right_arrow,productions_list]).
```

Write such production facts for a few grammars. Write a predicate `accepts(L)`, which means that a list of Prolog atoms corresponding to terminals of the language, understood as a string of the alphabet of the language, is in the language defined by the grammar.

How to turn in

Turn in your code by running *~dmykola/handin your-file* on a regular UNM CS machine. You should use whatever filename is appropriate in place of your-file. You can put multiple files on the command line, or even directories. Directories will have their entire contents handed in, so please be sure to clean out any cruft.

Remember to submit extensive tests of your programs!

Homework must be accompanied by the following statement: *"I pledge my honor that in the preparation of this assignment I have complied with the University of New Mexico Board of Regents' Policy Manual, including Section 4.8, Academic Dishonesty."* The manual is available at `http://www.unm.edu/~brpm/index.html`.