

# Project 2

**Due:** Both the paper and the electronic submission are due on **Thursday, July 26, 6 pm, for the day section**, and **Friday, July 27, 6 pm for the evening section**, in 324 drop box.

**Lateness reminder:** penalty of 15% of the mark if one day late, 30% if two days late; not accepted later.

**Note:** The names of your predicates should be the ones required in this assignment. Otherwise you will lose marks. The name of the submitted file should be **p2.pl**. Otherwise you will lose marks. You may not use any of the following: ; ("or"), -> ("if-then").

## 1 Part A

1. Implement the following predicates:

- `repl(L1,L2)` which holds if the list `L2` is the list `L1` where every number on the surface level is replaced with its square. The elements which are not numbers are kept unchanged. Precondition: `L1` is always instantiated.
- `nested_repl(L1,L2)` which holds if the list `L2` is the list `L1` where every number is replaced with its square at all levels. Atoms (non-numbers) are kept unchanged. Precondition: `L1` is always instantiated.

Examples:

```
?- repl([1,a,2,b],P).
P = [1, a, 4, b] ;
No
?- nested_repl([1,[a,[3]],2,b],P).
P = [1, [a, [9]], 4, b] ;
No
```

2. Each row in Pascal's triangle is computed from the previous row. There are 1's on the sides. An interior value is computed as the sum of the element in the corresponding position in the previous row and the element in position - 1 in the previous row.

```
1
1 1
1 2 1
1 3 3 1
1 4 6 4 1
1 5 10 10 5 1
...
```

Implement a predicate `next(L1,L2)` which holds if `L1` is a row in Pascal's triangle, and `L2` is the next row. Assume that `L1` is always instantiated and it is a valid row in Pascal's triangle. Examples:

```

?- pascal([1],X).
X = [1, 1] ;
No
?- pascal([1,4,6,4,1],X).
X = [1, 5, 10, 10, 5, 1]
Yes

```

## 2 Part B

In this part of the project you will write a parser for a small subset of the English language. The parser you will implement will be increasingly sophisticated. In order to preserve the earlier versions, don't change the name of the top predicate, but rather increase its arity (number of arguments) at each step. As you move to a new step, make a copy of the previous predicates and add the new feature required. The names of the helper predicates (used to implement the grammar) can also be the same if they have different arity at each step.

The language consists of short questions having to do with family relations. Here are some examples:

```

who is the sister of anna ?
who is the husband of afrodita ?
who is anna's husband ?
who is the brother of anna's husband ?
who is the brother-in-law of the sister of anna ?

```

The language is fully described by the following BNF grammar:

```

<question> ::= who is <np> ?
<np> ::= <person> |
        the <reln> of <np> |
        <person> s <reln>
<reln> ::= spouse | wife | husband | sister | brother | sibling | sister_in_law | brother_in_law
<person> ::= ...

```

Here, <person> stand for the name of a person. We consider a valid name any atom in Prolog (use the atom predicate).

Rather than representing the sentences as strings, we represent them as lists of words. We treat 's as a single word. We skip some special characters such as simple quote and dash when they occur in the sentences. Examples of input sentences:

```

[who, is, the, brother, of, anna, s, husband, ?]
[who, is, the, brother, in, law, of, the, sister, of, anna, ?]

```

Here is an example of running the three versions of your parser:

```

?- parse([who,is,the,brother,of,anna,?]).
Yes
?- parse(Tree,[who,is,the,brother,of,anna,?]).
Tree = question(who, is, np(the, reln(brother), of, np(person(anna))))
Yes
?- parse(Tree,Person,[who,is,the,brother,of,anna,?]).
Tree = question(who, is, np(the, reln(brother), of, np(person(anna))))

```

```

Person = john
Yes
?- parse(Tree,Person, [who,is,the,brother,of,anna,?]).
Tree = question(who, is, np(the, reln(brother), of, np(person(anna))))
Person = john ;
Tree = question(who, is, np(the, reln(brother), of, np(person(anna))))
Person = ed ;
No

```

1. Consider a database of the following form:

```

female(mary).
female(beth).
female(anna).
female(ellen).

male(john).
male(jim).
male(ed).
male(al).

married(mary, john).
married(jim,anna).
married(beth,ed).
married(al,ellen).

sibling_rel(john,anna).
sibling_rel(john,ed).
sibling_rel(anna,ed).
sibling_rel(mary,al).

```

Here, married(X,Y) means X is married to Y. The relation holds in both senses, that is Y is also married to X, but the information is stated only once. Similarly, sibling\_rel(X,Y) means X is Y's sibling and Y is X's sibling.

Implement the following predicates for family relations:

- spouse(X,Y) holds if X is married to Y. Example:

```

?- spouse(mary, john).
Yes
?- spouse(john, mary).
Yes

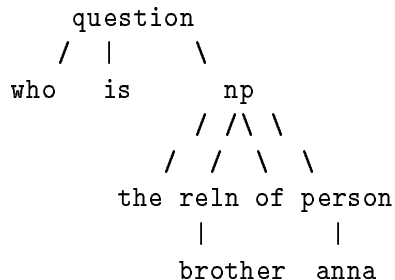
```

- husband(X,Y) holds if X is the husband of Y.
- wife(X,Y) holds if X is the wife of Y.
- sibling(X,Y) holds if X is Y's sibling.
- sister(X,Y) holds if X is the sister of Y.

- `brother(X,Y)` holds if X is the brother of Y.
- `sister_in_law(X,Y)` holds if X is the sister-in-law of Y. Note that X can be Y's sister-in-law in one of the following ways: by being married to Y's sibling, or by being the sister of Y's spouse.
- `brother_in_law(X,Y)` holds if X is the brother-in-law of Y.

2. Write a predicate `parse(S)` which holds if S is a sentence accepted by the above BNF grammar, and fails otherwise. S is always instantiated. It is very easy to implement a top-down parser in Prolog. All you have to do is to specify the BNF grammar in a manner Prolog can understand, and the built-in Prolog mechanism will apply the rules for you. One of the most elegant ways to implement the parser is to use the DCG (Definite Clause Grammar) notation Prolog supports. See the webpage for additional notes on parsing in Prolog.

3. Write a predicate `parse(Tree,S)` which holds if S is a sentence accepted by the grammar, and Tree is a parse tree for the sentence S. S is always instantiated. The parse tree has the words of the sentence as leaves, and the internal nodes have names corresponding to the non-terminals in the BNF grammar. For example, the parse tree for the sentence [who, is, the, brother, of, anna,?] is:



and gets represented in Prolog as:

```
question(who, is, np(the, reln(brother), of, np(person(anna)))).
```

Note that internal nodes such as `np` and `reln` are represented as Prolog functors (constant terms with different arity, depending of the number of branches in the parse tree).

4. Write a predicate `parse(Tree,Person,S)` which holds if S is a sentence accepted by the grammar, Tree is a parse tree for the sentence S, and Person is the answer to the question (the name of the person who satisfies the family relation). S is always instantiated. The parse tree is the same as before. This predicate is not only parsing your sentence, but it also does semantic interpretation and provides the answer to the question.

For this part you need to use the Prolog operator `=..` (also called “universal predicate”) which composes a predicate from a list with its name and its arguments. For example, try the query:

```
?- Pred =.. [brother,Pers,anna], call(Pred).
```

### 3 What to hand in?

#### 3.1 On paper

Please use an unsealed envelope, having on top the cover page provided at the end of this file. Put inside:

- A printout of your code.

- A printout of one or more script files containing tests run for each predicate, including helper predicates.

The tests for each predicate you implemented should include several relevant runs, degenerated cases for the input (such as empty list). You should also test for multiple solutions and for different legal combinations of constants and variables.

For part B, you don't have to test with a different database, but you should make your predicates general so that they will work with any database of the given form we may use for marking purposes.

You can save your test cases in a file, then cut and paste when you use script. You are allowed to try alternative methods Prolog may provide for producing script files.

### 3.2 Electronically

In addition to your paper submission, you must submit your code electronically.

- Put all definitions of the predicates above, as well as any helper predicates, in a file called `p2.pl`
- Submit the file `p2.pl` using the following command:

```
submit -N project2 csc324h p2.pl
```

See `man submit` if you need more information. Note that if you already submitted a file and you wish to replace it with a different version, you can submit it again. But you must use the `-f` option to overwrite the old version.

## 4 How the project will be marked?

It is worth emphasizing that your project will be **marked not only for correctness, but for logic programming style, comments, and your testing strategy as well.**

**Style:** the code should use logic programming style not imperative style. Helper predicates should be used when appropriate. Significant names should be used for predicates and their arguments. The code should be as efficient and elegant as possible.

**Comments:** should clearly state what are the preconditions of each predicate (expected number of arguments and their type), and what are the postconditions (results and expected behaviour). Additional comments inside the predicates should be used to explain steps that are not obvious. The comments should consist of full English sentences, including punctuation.

**Testing:** all predicates should be tested. The test cases for each predicate should include relevant cases (no less and no more than the necessary cases), including degenerate cases such as empty lists. You should also test for multiple solutions (using `;`). Please explain in a few sentences what is your testing strategy for each question (or annotate the test cases).

## Cover Page for Project 2

Family name: \_\_\_\_\_ Student #: \_\_\_\_\_

First name: \_\_\_\_\_ CDF id: \_\_\_\_\_

Section:        Day section        Evening section

I declare that this assignment is my own work, and it is in accordance with the University of Toronto Code of Behaviour on Academic Matters and the Code of Student Conduct. I declare that I didn't discuss this assignment with anybody else.

Signature \_\_\_\_\_

The following is for our use in grading:

Part A \_\_\_\_\_ / 18

Part B \_\_\_\_\_ / 46

Style \_\_\_\_\_ / 8

Comments \_\_\_\_\_ / 8

Testing \_\_\_\_\_ / 20

\_\_\_\_\_

Total \_\_\_\_\_ / 100

Other (penalties)

Final mark \_\_\_\_\_ / 100