ECE/CPSC 3520
Spring 2019
Software Design Exercise #1
Grammar-Based Image Analysis

Canvas submission only

Assigned 1/31/2019; Due 2/28/2019 11:59 PM

Contents

1 Preface

1.1 Objectives

The objective of SDE 1 is to implement a `Prolog` version of what is known as the picture description language (PDF). The PDL is a grammar-base technique to extract object or region outlines from an image using geometric primitives derived from the image. This will be implemented in Prolog and involves using the Prolog LGN preprocessor. We will specify 14 predicates which **must** be developed as part of this assignment. I recommend you attempt development in the order in which the predicates are introduced.

This effort is straightforward, and about 4 weeks are allocated for completion. The motivation is to:

- Learn the paradigm of declarative programming;
- Implement a declarative (Prolog) version of an interesting algorithm;
- Deliver working software based upon specifications; and
- Learn `Prolog`.

1.2 Resources

As discussed in class, it would be foolish to attempt this SDE without carefully exploring:

1. The text, especially:
   (a) The many Prolog examples in Chapter 3;
   (b) The Prolog LGN use in Chapter 5 (ignore scanner development; and
   (c) Chapter 7, Sections 7.7. and 7.8 (Attribute Grammars).
2. The `Prolog` lectures;
3. The background provided in this document;
4. **Class discussions and examples**;

5. The PDL references on the Assignment page (only for perspective); and

6. The `swipl` website and reference manual.

1.3 Standard Remark

Please note:

1. **This assignment (which counts as a quiz) tests *your* effort (not mine or those of your classmates or friends)**. I will not debug or design your code, nor will I install software for you. You (**and only you**) need to do these things.

2. This is an individual effort.

3. It is never too early to get started on this effort.

4. We will continue to discuss this in class.

2 Preliminary Considerations

2.1 Data Structures and Representation in `Prolog`

2.1.1 The PDL Primitives (Terminals)

One of the most common questions which arise at this point is:

> *'How do I get the image-based PDL data 'into' Prolog?'*

We will encode PDL descriptions using string lists in Prolog. For example, `"abcd"` is a Prolog string and `["a","b","c","d"]` is a string list. The following figure and the lecture discussion(s) should help. Here are some notes:

1. The entire extracted region outline, using the directed primitives shown, is a **Prolog string list**. For example, a simple square (of side dimension equal to the length of primitive 'u'), is represented by the string list `["u","r","d","l"]`. We want to be able to represent, and recognize, represent any side dimension greater than zero for all our regions.

2. Notice we have standard starting points for every grammar class representation. This is the base or 'tail' of the first 'u' (every class in this assignment contains at least on 'u'). In the figure, this is shown with a black dot. In order to accommodate a representation starting anywhere in the closed region contour, we can either create a very complex set of grammar productions (each starting a some point on a region of arbitrary size), or somehow consider all starting points in the closed region. The latter is our solution.

3. Note if the contour is not closed, all class tests must fail (e.g. `["u","r","d"]` fails. Also, for those who want to make this assignment even more complicated, we don't allow stupid descriptions due to overwriting the path, such as `["u","u","d","d","u"]`.

The simplest way to get comfortable with the PDL (not the Prolog implementation) is to draw the figures corresponding to the PDL string.

2.1.2 Sample Figure Classes

In Figure 1, notice a number of stings and corresponding string lists are shown. Each corresponds to a class, such as square, rectangle or triangle. Our effort will focus on developing parsers for each of these classes in Prolog. Of course, for a parser, we need a grammar. This is part of your effort. An attribute grammar is necessary.

3 Prototypes and Examples of Predicates to be Developed

You will design, implement, test and deliver the predicates described in the following sections. Be aware that some predicates are created using the LGN, as indicated. Also note:

1. Recall the '/n' in the predicate representation indicates the arity of the predicate is n. It is not part of the name but rather the Prolog convention for showing predicate name and arity.

2. Recall the +,- or ? symbol indicates the role of the argument.

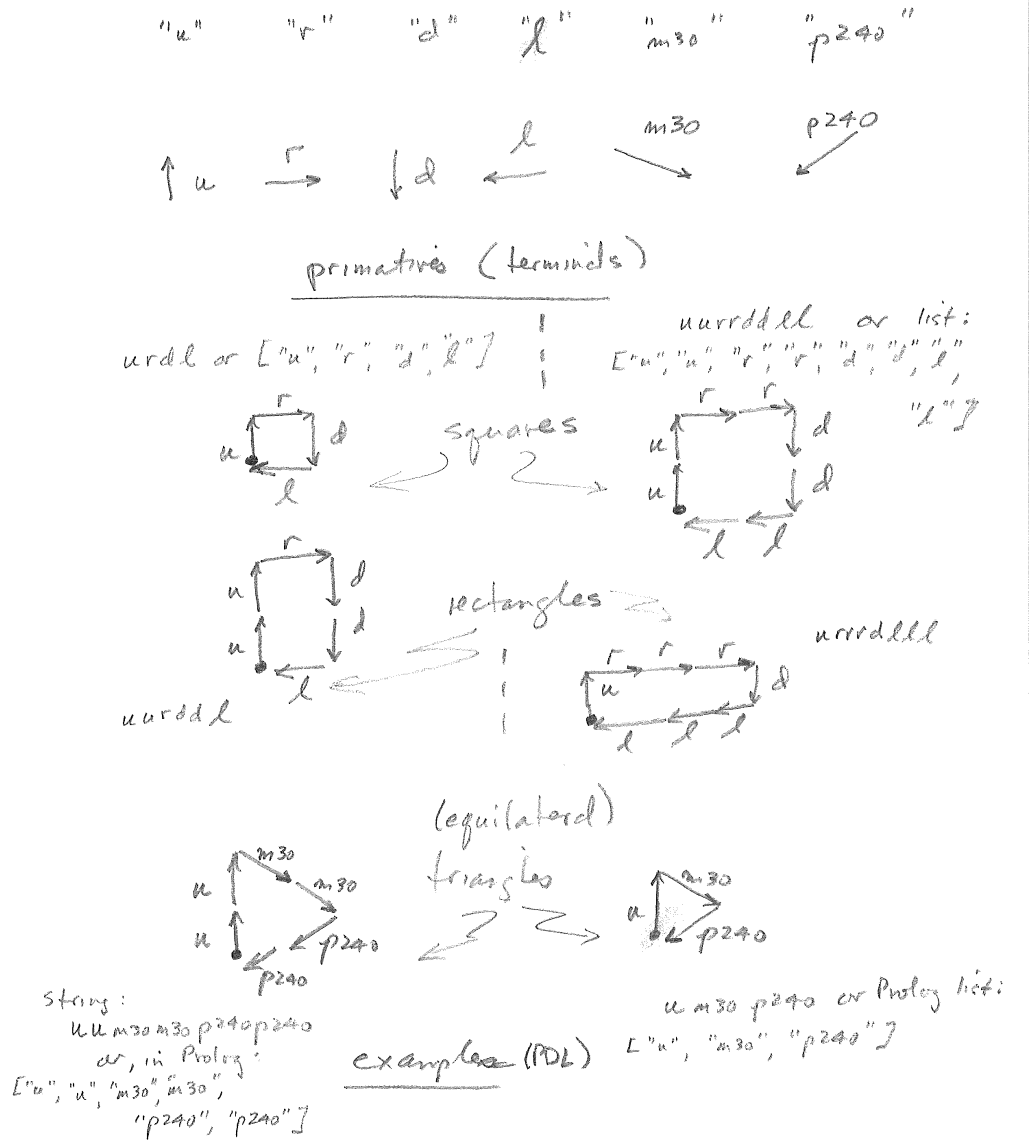

Figure 1: Sample Use of the PDL

3. **Pay special attention to the predicate naming and argument specifications**. Case matters. Since your submission will be graded by a Prolog script, if you deviate from this your tests will probably fail. **The graders will not fix or change anything in your submission.**

4. Pay attention to the file naming conventions specified.

5. Your implementation of **all** predicates is to be included in a single file named `which_shape.pro` in your archive.

6. You may design and use other predicates to support any of these predicates, but they are not (directly) graded. Of course, they must also be included in your single SDE1 Prolog file in your archive.

Note: in the specifications shown below, a (+) designation indicates that you obtain the predicate using the LGN.

3.1 uA/3 (+)

```
(**
Prototypes:

uA(-Length,+String,+Leftover)

Arguments:
Note: Parses string for sequence of "u", and, if succeeds,
      indicates length (number of "u").
Really big note: You create predicate uA using the LGN. *)
```

Sample Use.

```
?- uA(L,["u","u","u"],[]).
L = 3 .

?- uA(L,["u","u","d"],[]).
false.
```

3.2 rA/3, dA/3 and lA/3 (+)

```
(**
Prototype: Same structure as uA, but unifies with strings
```

```
          of "r","d" and "a" respectively.

Notes:  Created by LGN.
*)
```

Sample Use.

```
?- rA(W,["r"],[]).
W = 1 .

?- rA(W,["r","r","r","r","r"],[]).
W = 5 .

?- rA(W,["r","r","r","d","r"],[]).
false.
```

3.3 sq/2 (+)

This predicate is included just to point out that without contextual constraints (e.g., all sides of the same length), the desired class is different. See the examples.

```
(**
Prototype:

sq(+In,+Leftover)

Succeeds if In is a string list representing u^n r^m d^l l^p with string Leftover leftover.

Remark: Not really a square, just u^n r^m d^l l^p.
```

Sample Use.

```
?- sq(["u","r","d","l"],[]).
true .

?- sq(["u","u","d","l"],[]).
false.
```

3.4 sqA/2 (+)

```
(**
Prototype: sqA(+In,+Leftover)

like sq, but the length of each side must be equal.
```

Sample Use.

```
?- sqA(["u","r","d","l"],[]).
true .

?- sq(["u","u","r","d","l"],[]). /* Note sqA; note difference */
true .

?- sqA(["u","u","r","d","l"],[]).
false.

?- sqA(["u","u","u","r","r","d","d","d","l","l"],[]).
false.
```

3.5 rctA/2 (+)

```
(**
Prototype: rctA(+In,+Leftover)

Like sqA, but only parallel sides must have equal length.
```

Sample Use.

```
?- rctA(["u","u","r","d","d","l"],[]).
true .

?- rctA(["u","u","r","d","l","l"],[]).
false.

?- rctA(["u","r","r","d","l","l"],[]).
true .

?- rctA(["u","u","u","r","r","d","d","d","l","l"],[]).
true .

?- sqA(["u","u","u","r","r","d","d","d","l","l"],[]).
false.
```

3.6 grect/3

This is a good time to throw in an easy problem. This predicate generates a rectangle in the PDL.

```
(**
Prototype: grect(+A,+B,-C)

    A is length of "u" and "d" sides;
    B is length of "r" and "l" sides
    C is the resulting PDL description as a top-level Prolog string list
```

Sample Use.

```
?- grect(5,2,What).
What = ["u", "u", "u", "u", "u", "r", "r", "d", "d"|...].

?- grect(5,2,What), writeq(What).
["u","u","u","u","u","r","r","d","d","d","d","d","l","l"]
What = ["u", "u", "u", "u", "u", "r", "r", "d", "d"|...].

?- grect(3,4,R).
R = ["u", "u", "u", "r", "r", "r", "r", "d", "d"|...].

?- grect(3,4,R), writeq(R).
["u","u","u","r","r","r","r","d","d","d","l","l","l","l"]
R = ["u", "u", "u", "r", "r", "r", "r", "d", "d"|...].
```

3.7 m30A/3 and p240A/3 (+)

```
(**
Prototype: Just like uA,rA, etc. but unify with a contiguous
substring of "m30A" or "p240A" and return length. See the examples.
```

Sample Use.

```
?- p240A(L,["p240","p240"],[]).
L = 2 .

?- p240A(L,["p240","m30"],[]).
false.
```

3.8 eqtriA/2 (+)

```
(**
Prototype: eqtriA(+In,+Leftover)

Note: Equilateral triangle recognizer, starting at the first "u".
```

Sample Use.

```
?- eqtriA(["u","u","m30","p240"],[]).
false.

?- eqtriA(["u","m30","p240"],[]).
true.

?- eqtriA(["m30","p240","u"],[]).
false.
?- eqtriA(["u","u","u","m30","m30","m30","p240","p240","p240"],[]).
true.

?- eqtriA(["u","u","m30","m30","m30","p240","p240","p240","u"],[]).
false.

/* Notes: Last example above (draw the figure) illustrates the next challenge (cyclic shifts). T
```

4 Prototypes, Signatures and Examples of Higher-Level Predicates to be Developed

Here, the real work starts. These predicates have the higher-level functionality involved for recognition of the shape class **regardless of where the PDL string description starts**. Recall in Section 2.1.1 we assumed it always started with the 'lowest' "u".

4.1 one_shift/2

```
(**
Prototype: one_shift(+A,-R)

R is list A cyclically shifted to the left by one position.

Notes:
```

Sample Use.

```
?- one_shift([1,2,3],W).
W = [2, 3, 1].

?- one_shift([1,2,3,4,5],W).
W = [2, 3, 4, 5, 1].
```

```
?- one_shift([1,2,3,4,5],W),one_shift(W,R).
W = [2, 3, 4, 5, 1],
R = [3, 4, 5, 1, 2].

?- one_shift([1,2,3,4,5],W),one_shift(W,R),one_shift(R,Q).
W = [2, 3, 4, 5, 1],
R = [3, 4, 5, 1, 2],
Q = [4, 5, 1, 2, 3].
```

4.2 all_shifts/4

```
(**
Prototype: all_shifts(+A,-R,+L,+S)

R is all cyclic shifts of A > 0. L is the length of A.
S starts at 1.

Note: R does not contain A.
```

Sample Use.

```
?- length([1,2,3,4],L), all_shifts([1,2,3,4],R,L,1).
L = 4,
R = [[2, 3, 4, 1], [3, 4, 1, 2], [4, 1, 2, 3]].
```

4.3 start_shifts/2

```
(**
Prototype: start_shifts(+L,-AS)

L is the input list, AS is all cyclic shifts of L. (See example).
Note: This predicate uses all_shifts. AS still does not contain A.
Look at the example immediately preceding this one.
```

Sample Use.

```
?- start_shifts([1,2,3,4],What).
What = [[2, 3, 4, 1], [3, 4, 1, 2], [4, 1, 2, 3]].
```

4.4 all_cases/2

```
(**
```

```
Prototype: all_cases(+A,-R).

R is all shifts of A (computed via previous predicates) appended to A. This is all possible shif
```

Sample Use.

```
?- all_cases([1,2,3,4],What).
What = [[1, 2, 3, 4], [2, 3, 4, 1], [3, 4, 1, 2], [4, 1, 2, 3]].

?- all_cases(["u","r","d","l"],What).
What = [["u", "r", "d", "l"], ["r", "d", "l", "u"],
["d", "l", "u", "r"], ["l", "u", "r", "d"]].

?- all_cases(["u","u","u","r","r","d","d","d","l","l"],What).
What = [["u", "u", "u", "r", "r", "d", "d", "d"|...], ["u", "u", "r",
 "r", "d", "d", "d"|...], ["u", "r", "r", "d", "d", "d"|...],
["r", "r", "d", "d", "d"|...], ["r", "d", "d", "d"|...],
["d", "d", "d"|...], ["d", "d"|...], ["d"|...], [...|...]|...].

/* The display is getting cluttered and truncated. Let's write it.
    Display of all cases (all cyclic shifts) */

?- all_cases(["u","u","u","r","r","d","d","d","l","l"],What), write(What).
[[u,u,u,r,r,d,d,d,l,l],[u,u,r,r,d,d,d,l,l,u],
[u,r,r,d,d,d,l,l,u,u],[r,r,d,d,d,l,l,u,u,u],
[r,d,d,d,l,l,u,u,u,r],[d,d,d,l,l,u,u,u,r,r],
[d,d,l,l,u,u,u,r,r,d],[d,l,l,u,u,u,r,r,d,d],
[l,l,u,u,u,r,r,d,d,d],[l,u,u,u,r,r,d,d,d,l]]
What = [["u", "u", "u", "r", "r", "d", "d", "d"|...],
["u", "u", "r", "r", "d", "d", "d"|...], ["u", "r", "r", "d", "d",
 "d"|...], ["r", "r", "d", "d", "d"|...], ["r", "d", "d", "d"|...],
 ["d", "d", "d"|...], ["d", "d"|...], ["d"|...], [...|...]|...].

/* Better to print the resulting list of lists as string lists (writeq) */

?- all_cases(["u","u","u","r","r","d","d","d","l","l"],What), writeq(What).
[["u","u","u","r","r","d","d","d","l","l"]
,["u","u","r","r","d","d","d","l","l","u"]
,["u","r","r","d","d","d","l","l","u","u"]
,["r","r","d","d","d","l","l","u","u","u"]
,["r","d","d","d","l","l","u","u","u","r"]
,["d","d","d","l","l","u","u","u","r","r"]
,["d","d","l","l","u","u","u","r","r","d"]
,["d","l","l","u","u","u","r","r","d","d"]
,["l","l","u","u","u","r","r","d","d","d"]
```

```
,["l","u","u","u","r","r","d","d","d","l"]]

What = [["u", "u", "u", "r", "r", "d", "d", "d"|...],
["u", "u", "r", "r", "d", "d", "d"|...], ["u", "r", "r", "d", "d",
 "d"|...], ["r", "r", "d", "d", "d"|...],
["r", "d", "d", "d"|...], ["d", "d", "d"|...], ["d", "d"|...],
 ["d"|...], [...|...]|...].
```

4.5 try_all_sqA/1

Now we consider the ultimate solution to our recognition/parsing problem: recognition of specific figures over all shifts. We do this for each prototype class (type of figure, i.e., square, rectangle, triangle). We rely separately (outside this predicate) on the generation of all shifts. See all_cases/2.

```
(**
Prototype: try_all_sqA(+Cases)

Given all_shifts (including none),
   succeeds if some cyclic shift of Cases satisfies sqA.
Your output display should look as shown.
```

Sample Use.

```
?- all_cases(["u","r","r","d","d","l","l","u"],What),try_all_sqA(What).

cyclic shift: ["u","u","r","r","d","d","l","l"] is a square
What = [["u", "r", "r", "d", "d", "l", "l", "u"],
["r", "r", "d", "d", "l", "l", "u"|...],
["r", "d", "d", "l", "l", "u"|...], ["d", "d", "l", "l", "u"|...],
["d", "l", "l", "u"|...], ["l", "l", "u"|...], ["l", "u"|...],
 ["u"|...]].

?- all_cases(["u","u","u","r","r","d","d","d","l","l"],What),try_all_sqA(What).
false.
```

4.6 try_all_rctA/1

```
(**
Prototypes: try_all_rctA(+Cases), try_all_eqtriA(+Cases)

Same as try_all_sqA(+Cases), but looking for rctA and eqtriA in Cases.
```

Sample Use.

```
?- all_cases(["u","u","u","r","r","d","d","d","l","l"],What), try_all_rctA(What).

cyclic shift: ["u","u","u","r","r","d","d","d","l","l"] is a rectangle

What = [["u", "u", "u", "r", "r", "d", "d", "d"|...], ["u", "u", "r",
 "r", "d", "d", "d"|...], ["u", "r", "r", "d", "d", "d"|...], ["r", "r",
 "d", "d", "d"|...], ["r", "d", "d", "d"|...], ["d", "d", "d"|...],
 ["d", "d"|...], ["d"|...], [...|...]|...].

?- all_cases(["d","d","d","l","l","u","u","u","r","r"],Cases),try_all_rctA(Cases).

cyclic shift: ["u","u","u","r","r","d","d","d","l","l"] is a rectangle

Cases = [["d", "d", "d", "l", "l", "u", "u", "u"|...], ["d", "d", "l",
 "l", "u", "u", "u"|...], ["d", "l", "l", "u", "u", "u"|...], ["l", "l",
 "u", "u", "u"|...], ["l", "u", "u", "u"|...], ["u", "u", "u"|...],
 ["u", "u"|...], ["u"|...], [...|...]|...].

?- eqtriA(["u","u","u","m30","m30","m30","p240","p240","p240"],[]).
true.

?- eqtriA(["u","u","m30","m30","m30","p240","p240","p240","u"],[]).
false.

?- all_cases(["u","u","u","m30","m30","m30","p240","p240","p240"],All),writeq(All).

[["u","u","u","m30","m30","m30","p240","p240","p240"]
,["u","u","m30","m30","m30","p240","p240","p240","u"]
,["u","m30","m30","m30","p240","p240","p240","u","u"]
,["m30","m30","m30","p240","p240","p240","u","u","u"]
,["m30","m30","p240","p240","p240","u","u","u","m30"]
,["m30","p240","p240","p240","u","u","u","m30","m30"]
,["p240","p240","p240","u","u","u","m30","m30","m30"]
,["p240","p240","u","u","u","m30","m30","m30","p240"]
,["p240","u","u","u","m30","m30","m30","p240","p240"]]

All = [["u", "u", "u", "m30", "m30", "m30", "p240", "p240"|...], ["u",
 "u", "m30", "m30", "m30", "p240", "p240"|...], ["u", "m30", "m30",
 "m30", "p240", "p240"|...], ["m30", "m30", "m30", "p240", "p240"|...],
 ["m30", "m30", "p240", "p240"|...], ["m30", "p240", "p240"|...],
 ["p240", "p240"|...], ["p240"|...], [...|...]].

 ?- all_cases(["p240","p240","u","u","u","m30","m30","m30","p240"],Cases).
```

```
Cases = [["p240", "p240", "u", "u", "u", "m30", "m30", "m30"|...],
 ["p240", "u", "u", "u", "m30", "m30", "m30"|...], ["u", "u", "u",
 "m30", "m30", "m30"|...], ["u", "u", "m30", "m30", "m30"|...], ["u",
 "m30", "m30", "m30"|...], ["m30", "m30", "m30"|...], ["m30",
 "m30"|...], ["m30"|...], [...|...]].

?- all_cases(["p240","p240","u","u","u","m30","m30","m30","p240"],Cases),writeq(Cases).
[["p240","p240","u","u","u","m30","m30","m30","p240"]
,["p240","u","u","u","m30","m30","m30","p240","p240"]
,["u","u","u","m30","m30","m30","p240","p240","p240"]
,["u","u","m30","m30","m30","p240","p240","p240","u"]
,["u","m30","m30","m30","p240","p240","p240","u","u"]
,["m30","m30","m30","p240","p240","p240","u","u","u"]
,["m30","m30","p240","p240","p240","u","u","u","m30"]
,["m30","p240","p240","p240","u","u","u","m30","m30"]
,["p240","p240","p240","u","u","u","m30","m30","m30"]]

Cases = [["p240", "p240", "u", "u", "u", "m30", "m30", "m30"|...],
 ["p240", "u", "u", "u", "m30", "m30", "m30"|...], ["u", "u", "u",
 "m30", "m30", "m30"|...], ["u", "u", "m30", "m30", "m30"|...], ["u",
 "m30", "m30", "m30"|...], ["m30", "m30", "m30"|...], ["m30",
 "m30"|...], ["m30"|...], [...|...]].

?- all_cases(["u","m30","m30","m30","p240","p240","p240","u","u"],Cases),writeq(Cases).

[["u","m30","m30","m30","p240","p240","p240","u","u"]
,["m30","m30","m30","p240","p240","p240","u","u","u"]
,["m30","m30","p240","p240","p240","u","u","u","m30"]
,["m30","p240","p240","p240","u","u","u","m30","m30"]
,["p240","p240","p240","u","u","u","m30","m30","m30"]
,["p240","p240","u","u","u","m30","m30","m30","p240"]
,["p240","u","u","u","m30","m30","m30","p240","p240"]
,["u","u","u","m30","m30","m30","p240","p240","p240"]
,["u","u","m30","m30","m30","p240","p240","p240","u"]]

Cases = [["u", "m30", "m30", "m30", "p240", "p240", "p240", "u"|...],
 ["m30", "m30", "m30", "p240", "p240", "p240", "u"|...], ["m30", "m30",
 "p240", "p240", "p240", "u"|...], ["m30", "p240", "p240", "p240",
 "u"|...], ["p240", "p240", "p240", "u"|...], ["p240", "p240", "u"|...],
 ["p240", "u"|...], ["u"|...], [...|...]].

?- all_cases(["u","u","u","r","r","d","d","d","l","l"],What),try_all_eqtriA(What).
false.

?- all_cases(["p240","p240","u","u","u","m30","m30","m30","p240"],What),try_all_eqtriA(What).
```

```
cyclic shift: ["u","u","u","m30","m30","m30","p240","p240","p240"] is an equilateral triangle

What = [["p240", "p240", "u", "u", "u", "m30", "m30", "m30"|...],
 ["p240", "u", "u", "u", "m30", "m30", "m30"|...], ["u", "u", "u",
 "m30", "m30", "m30"|...], ["u", "u", "m30", "m30", "m30"|...], ["u",
 "m30", "m30", "m30"|...], ["m30", "m30", "m30"|...], ["m30",
 "m30"|...], ["m30"|...], [...|...]].

?- all_cases(["p240","p240","u","u","m30","m30","m30","p240"],What),try_all_eqtriA(What).
false.
```

5 How We Will Grade Your Solution

A (Prolog) script will be used to evaluate your Prolog predicates with varying inputs and parameters. **The grade is based upon a correctly working solution.**

6 Prolog Use Limitations and Constructs Not Allowed

1. The entire solution must be in SWI-Prolog (Version 7.2 or newer.)
2. No use of the `if..then` construct anywhere in your Prolog source file. (Don't even bother looking for it). It has the syntax:

   ```
   condition -> then_clause ; else_clause
   ```

   and is useful for people who want an if-then capability, but don't understand the goal-satisfaction mechanism in Prolog.
3. No use of predicates `assert` or `retract`.

   If you are in doubt about anything allowable, ask and I'll provide a 'private-letter ruling'.

The objective is to obtain proficiency in declarative programming and Prolog, not to try to find built-in predicates which simplify or trivialize the effort, or to implement an imperative solution using a declarative programming paradigm.

7 Pragmatics

Use this document as a checklist to be sure you have responded to all parts of the SDE, and have included the 3 required files (*.txt, *.pro and *.log) in you archive. Furthermore:

- The simulation uses **only** (SWI-Prolog) code you wrote.
- The final, single zipped archive which must be submitted (to Canvas) by the deadline must be named **<yourname>-sde1.zip**, where <yourname> is your (CU) assigned user name.

  The contents of this archive are:

  1. A `readme.txt` file listing the contents of the archive and a very brief description of each file. Include 'the pledge' here. Here's the pledge:

     Pledge:
     On my honor I have neither given nor received aid on this exam
     SIGN _____

  2. Your single SWI-Prolog source file `which_shape.pro` containing the required predicates.
  3. A log file named `sde1.log` showing 3 uses of each required predicate. Generate and use test cases other than those shown herein.

We will attempt to consult your Prolog source file and then query Prolog with relevant goals. Most of the grade will be based upon this evaluation.

8 Final Remark: Deadlines Matter

Since multiple submissions to Canvas are allowed[1], if you have not completed all predicates, you should submit a freestanding archive of your current success before the deadline. This will allow the possibility of partial credit. **Do not attach any late submissions to email and send to either me or the graders. There will be a penalty for this.**

[1]But we will only download and grade the latest (or most recent) one, and it must be submitted by the deadline.