Defining recursive rules in PROLOG to answer questions about a knowledge base.

a. *RoboCup* (www.robocup.org) is an international joint project to promote AI, robotics,and related research. One of the grand challenges in this project is to have teams of robots play soccer.The robots participating in this challenge may need to reason to choose the most successful behavior.In this part of the assignment, you are asked to write a set of control rules that will allow a robot to reason whether they can score from their current situation. The knowledge base will include statements defining the current situation using the following predicates:

- robot(R) - means that R is a robot.
- pathClear(L1, L2) - means that there is a clear path (*ie.* no opponent player in the way) from L1 to L2. Here, L1 and L2 can either be a robot or the goal, which is denoted by the constant goal.
- hasBall(R) - means that robot R has the ball in the current situation.

Note that pathClear(R, L2) means that if R has the ball, then they can kick it to L2 without it being intercepted by the opponent. The same is true of pathClear(L2, R), since pathClear is a symmetric predicate.

You will now write rules for the following predicates:

- canPass(R1, R2, M) - means that if the robot R1 has the ball, they can get the ball to R2 in at most M passes. That is, either R1 can pass the ball directly to R1 (and $M \geq 1$), or the ball can be passed through a sequence of robots so that it arrives to R2 in at most M passes.
- canScore(R, M) - means that the robot R can score using at most M kicks, where a kick is either a pass or a shot at the goal. That is, R can score if they have the ball and can shoot directly at the goal (and $M \geq 1$), or because the ball can be passed to R in at most M - 1 steps.

Note, that for simplicity, we are assuming that the opponents are not moving (*ie.* if a path remains clear before a pass, then it also remains clear after the pass. For example, consider the situation defined by the following knowledge base.

```
robot(r1). robot(r2). robot(r3).
robot(r4). robot(r5). robot(r6).
hasBall(r3).
pathClear(r1, goal).
pathClear(r2, r1).
pathClear(r3, r2).
pathClear(r3, goal).
pathClear(r3, r1).
pathClear(r3, r4).
pathClear(r4, goal).
pathClear(r1, r5).
pathClear(r5, r6).
```

Here, canScore(r3, 1) holds since r3 has the ball, and the path is clear between r3 and the goal. It also holds that canPass(r3, r1, 1), since the path since the path is clear between r3 and r1. In turn, this means that canScore(r1, 2) holds, since r3 can get r1 the ball in 1 step, and the path is clear between the r1 and the goal. Since r3 can get the ball to r2 with one pass, who can then pass it to r1, it is also true that canScore(r2, 3).

In addition, note the following:

- Your rules should be put in the 1_robocup.pl section of the supplied file

- Since canScore(r2, 3) holds, it is also the case that canScore(r2, 4), canScore(r2, 5), etc. hold. However, canScore(r2, 2) does not hold, since it takes 1 pass to get r2 the ball, and r2 can't then get it to the goal without at least 2 more kicks.
- canPass(R1, R2, M) does not require that R1 has the ball, only that if they ever get the ball, they can get it to R2 in at most M passes.
- A robot cannot pass the ball to the goal, so canPass(R, goal, M) should always fail for any robot R or M.
- The KB for the situation above is already given in the supplied file. We will test your rules in situations (ie. KBs) other than the one given above. We may use any robot names, but the constant goal will always mean the goal.
- We will not test your canPass or canScore rules with queries in which M is a variable. That is, we will not perform tests of the form canPass(r1, r2, M) or canScore(r1, M).
- Any of the other parameters may be variables. In particular, we may have tests of the form canPass(r1, r2, 5), canPass(R, r1, 5), canPass(r1, R, 5), and canPass(R1, R2, 5). Similarly, we may test queries of the form canScore(r1, 4) or canScore(R, 4).
- If multiple answers are requested using ";" or "More" it is ok if your rules return duplicate answers. However, they should never return incorrect answers.
- You can assume only one robot has the ball in the knowledge base.
- If a robot r1 can pass to at least one other robot, then canPass(r1, r1, M) should succeed for any $M \geq 2$. This is because of the symmetrical nature of pathClear.
- You are allowed to define and use helper predicates.
- You may edit the KB given if you find it helpful for testing. However, your interaction log (see part (c)) should be based on the given KB.

b. Create 10 queries that test your rules on the situation given by the KB from part (a). Enter these queries, along with an English language explanation of each, in the file queries.txt. In addition, put a log of your tests with these queries at the bottom of the queries.txt file.
Make sure the answers you get are for KB given in part (a). You will lose marks if our tests with your queries produce different answers on this KB than you list in this file.

c. Recall the Robocup question from Assignment 1.[1] There, you were asked to find if it was possible to score in at most a given number of passes/shots. However, your predicates in Assignment 1 only indicated whether it was possible, and they did not provide information about what path of passes and shots were necessary. In this question, you will use lists to compute such information, as well as to ensure that there are no unnecessary passes (ie. the ball never reaches the same robot more than once). That is, you will create two new versions of the canPass and canScore:

- canPass(R1, R2, M, Path) - this predicate is true if robot R1 has the ball, they can get the ball to R2 in *at most* M passes, and Path is the list of robots along the way from R to R2.
- canScore(R, M, Path) - this predicate is true if R can score using *at most* M kicks, where a kick is either a pass or a shot at the net. **This means that** R **has a clear path to kick directly at the net, and the robot with the ball can pass it to** R **in at most** M - 1 **passes**. Path is the list of robots along the way from the robot that has the ball to net.

To better understand what is required, recall the situation given in assignment 1:

```
robot(r1). robot(r2). robot(r3). robot(r4). robot(r5). robot(r6).
hasBall(r3).
pathClear(r1, net). pathClear(r2, r1). pathClear(r3, r2).
pathClear(r3, net). pathClear(r3, r1). pathClear(r3, r4).
pathClear(r4, net). pathClear(r1, r5). pathClear(r5, r6).
```

Here, canScore(r3, 1, [r3, net]) holds since r3 has the ball, and the path is clear between r3 and the net. It also holds that canPass(r3, r1, 1, [r3, r1]), since the path since the path is clear between r3 and r1. Since the path is clear from r1 to the net, this also means that canScore(r1, 2, [r3, r1, net]) holds, as does canScore(r1, 3, [r3, r1, net]). ~~In addition, since r3 can get the ball r2 with one pass, who can then pass it to r1, it is also true that canScore(r2, 3, [r3, r2, r1, net])~~.

Additional notes:

- For canPass, the first element of Path should be R1 and the last should be R2. For canScore, the first element of Path should be the robot with the ball, and the last one should be net.
- For both canPass and canScore, there should not be any robots that appear more than once along Path. For example, Path cannot equal something like [r3, r1, r3] for canPass or [r3, r1, r3, net] for canScore.
- Recall that pathClear is a symmetric relation, a robot cannot pass to the net (or vice versa), and a robot cannot pass to itself.
- You do **not** need to submit the knowledge base you use for testing. Space has been made for it in the required file, but we will not be marking it. That space is simply there to make it easier for you to test your program. It is recommended that you test with other KBs to ensure your program's correctness.
- Your program should be able to handle queries like canScore(R, 3, Path) which can be read as find me a robot who can score in at most 3 kicks. By asking for "More", you should be able to get all robots who can score in at most 3 kicks. Your program should similarly be able to handle queries like canPass(R1, R2, 3, Path) which can be used to find all pairs of robots that can pass between each other in at most 3 passes. However, the maximum value M will not be set to a variable in all test queries.

Write your program in the robocup section of the file 2_robocup.pl. You may also define and use any helper predicates that you wish, but they must be in this section of the file.

[1]This question was updated on September 29 due to an important clarification. See the **bold** and ~~strikethrough~~ text.