Solving a Dungeons & Dragons riddle using prolog



# Solving a Dungeons & Dragons riddle using prolog

**Bringing back the magic of Christmas using the magic of prolog**

As part of a holiday D&D one-shot session where Santa Claus's toy factory had been sabotaged, our dungeon master presented to us, a group of Christmas elves, a riddle to solve.

9 cards, labeled with the names of Santa's reindeer were presented to us. The instructions indicated that we had to find the order reindeer were in, according to this riddle:

> Vixen should be behind Rudolph, Prancer and Dasher, whilst Vixen should be in front of Dancer and Comet. Dancer should be behind Donder, Blitzen and Rudolph. Comet should be behind Cupid, Prancer and Rudolph. Donder should be behind Comet, Vixen, Dasher, Prancer and Cupid. Cupid should be in front of Comet, Blitzen, Vixen, Dancer and Rudolph. Prancer should be in front of Blitzen, Donder and Cupid. Blitzen should be behind Cupid but in front of Dancer, Vixen and Donder. Rudolph should be behind Prancer but in front of Dasher, Dancer and Donder. Finally, Dasher should be behind Prancer but in front of Blitzen, Dancer and Vixen.

These sentences seem a lot like prolog facts to me! Let's translate them to prolog facts, and then use said facts to solve the riddle.

## Solution with the least knowledge about the problem

First, we declare facts for all relationships between reindeer as described in the riddle.

We'll use `is_behind(Second, First)` to describe when a `Second` reindeer is behind the `First` reindeer, and the opposite for when a reindeer is in front of the other one (`Second` is in front of `First` becomes `is_behind(First, Second)`).

```
% Vixen should be behind Rudolph, Prancer and Dasher,
is_behind(vixen, rudolph).
is_behind(vixen, prancer).
is_behind(vixen, dasher).

% whilst Vixen should be in front of Dancer and Comet.
is_behind(dancer, vixen).
is_behind(comet, vixen).

% Dancer should be behind Donder, Blitzen and Rudolph.
is_behind(dancer, donder).
is_behind(dancer, blitzen).
is_behind(dancer, rudolph).

% Comet should be behind Cupid, Prancer and Rudolph.
is_behind(comet, cupid).
is_behind(comet, prancer).
is_behind(comet, rudolph).

% Donder should be behind Comet, Vixen, Dasher, Prancer and Cupid.
is_behind(donder, comet).
is_behind(donder, vixen).
is_behind(donder, dasher).
is_behind(donder, prancer).
is_behind(donder, cupid).

% Cupid should be in front of Comet, Blitzen, Vixen, Dancer and Rudolph.
is_behind(comet, cupid).
is_behind(blitzen, cupid).
is_behind(vixen, cupid).
is_behind(dancer, cupid).
is_behind(rudolph, cupid).

% Prancer should be in front of Blitzen, Donder and Cupid.
is_behind(blitzen, prancer).
is_behind(donder, prancer).
is_behind(cupid, prancer).

% Blitzen should be behind Cupid but in front of Dancer, Vixen and Donder.
is_behind(blitzen, cupid).
is_behind(dancer, blitzen).
is_behind(vixen, blitzen).
is_behind(donder, blitzen).

% Rudolph should be behind Prancer but in front of Dasher, Dancer and Donder.
is_behind(rudolph, prancer).
is_behind(dasher, rudolph).
is_behind(dancer, rudolph).
```

```
is_behind(donder, rudolph).

% Finally, Dasher should be behind Prancer but in front of Blitzen, Dancer and Vixen.
is_behind(dasher, prancer).
is_behind(blitzen, dasher).
is_behind(dancer, dasher).
is_behind(vixen, dasher).
```

Using a rule, we declare that if `Last` is behind `Middle` and `Middle` is behind `First`, then `Last` is behind `First`. In other words, by transitivity, if `x > y` and `y > z`, then `x > z`. We'll need it when testing if a sequence respects the reindeer order described by the facts.

```
follows(Last, First) :- is_behind(Last, First).
follows(Last, First) :- is_behind(Middle, First), follows(Last, Middle).
```

A sequence respects the order if reindeer follow each other according to the facts. Let's specify that in a rule.

```
respects_order([First|[Second]]) :- follows(Second, First).
respects_order([First|[Second|Rest]]) :-
    follows(Second, First), respects_order([Second|Rest]).
```

Finally, let's declare a rule that will find a valid solution. We first list all unique known reindeer. Then, we compute possible reindeer sequence permutations. Lastly, we find the first permutation that's a valid solution — the one that respects the reindeer order.

```
solution(Permutation) :-
    findall(Reindeer, (is_behind(Reindeer, _); is_behind(_, Reindeer)), List),
    list_to_set(List, UniqueReindeer),
    permutation(UniqueReindeer, Permutation),
    once(respects_order(Permutation)).
```

We can now query `solution` to find the reindeer sequence...

```
?- solution(Sequence)
    Sequence = [
        prancer,
        cupid,
        rudolph,
        dasher,
        blitzen,
        vixen,
        comet,
        donder,
```

```
        dancer
    ]
```

And there you have it, the answer to our riddle! We only needed a few elements:

- Facts about the reindeer
- Rules to confirm that a sequence is valid
- A rule to test possible sequences

Although this works great, we are not taking full advantage of the power of prolog, nor the knowledge we have about what a valid solution consists of.

## Other solution, knowing there are 9 reindeer

We can combine the knowledge that there are exactly 9 unique reindeer with this Prolog wiki solution for a similar logic puzzle to create a more succinct solution. Furthermore, whereas in the previous solution we had to manually make the `is_behind` rule transitive by declaring the `follows` rule, we'll depend on the transitivity of comparison operators ("less than" and "greater than") in this solution.

We'll generate permutations by associating every reindeer to a unique number. Then, we'll test the permutations against the riddle.

First, we list the reindeer with a list of possible positions for them (1 through 9). We use the permutation predicate to have prolog generate reindeer sequences:

```
permutation(
    [Vixen, Rudolph, Prancer, Dasher, Dancer, Comet, Donder, Blitzen, Cupid],
    [1, 2, 3, 4, 5, 6, 7, 8, 9])
```

Then, we declare the riddle sentences as comparisons on free variables. Second > First means that Second is behind First. First < Second means that First is in front of Second.

```
% Vixen should be behind Rudolph, Prancer and Dasher,
Vixen > Rudolph, Vixen > Prancer, Vixen > Dasher,

% whilst Vixen should be in front of Dancer and Comet.
Vixen < Dancer, Vixen < Comet,

% Dancer should be behind Donder, Blitzen and Rudolph.
Dancer > Donder, Dancer > Blitzen, Dancer > Comet,

% Comet should be behind Cupid, Prancer and Rudolph.
Comet > Cupid, Comet > Prancer, Comet > Rudolph,

% Donder should be behind Comet, Vixen, Dasher, Prancer and Cupid.
```

```
Donder > Comet, Donder > Vixen, Donder > Dasher, Donder > Prancer, Donder > Cupid,

% Cupid should be in front of Comet, Blitzen, Vixen, Dancer and Rudolph.
Cupid < Comet, Cupid < Blitzen, Cupid < Vixen, Cupid < Dancer, Cupid < Rudolph,

% Prancer should be in front of Blitzen, Donder and Cupid.
Prancer < Blitzen, Prancer < Donder, Prancer < Cupid,

% Blitzen should be behind Cupid but in front of Dancer, Vixen and Donder.
Blitzen > Cupid, Blitzen < Dancer, Blitzen < Vixen, Blitzen < Donder,

% Rudolph should be behind Prancer but in front of Dasher, Dancer and Donder.
Rudolph > Prancer, Rudolph < Dasher, Rudolph < Dancer, Rudolph < Donder,

% Finally, Dasher should be behind Prancer but in front of Blitzen, Dancer and Vixen.
Dasher > Prancer, Dasher < Blitzen, Dasher < Dancer, Dasher < Vixen.
```

Finally, we put it together under a solution rule...

```
solution([Vixen, Rudolph, Prancer, Dasher, Dancer, Comet, Donder, Blitzen, Cupid]) :-
    permutation([Vixen, Rudolph, Prancer, Dasher, Dancer, Comet, Donder, Blitzen, Cupi
    % Vixen should be behind Rudolph, Prancer and Dasher,
    Vixen > Rudolph, Vixen > Prancer, Vixen > Dasher,
    % ... more comparison rules
```

... and we query it!

```
?- solution([Vixen, Rudolph, Prancer, Dasher, Dancer, Comet, Donder, Blitzen, Cupid])
    Blitzen = 5,
    Comet = 7,
    Cupid = 2,
    Dancer = 9,
    Dasher = 4,
    Donder = 8,
    Prancer = 1,
    Rudolph = 3,
    Vixen = 6
```

Sorting them by hand, we get the answer to the riddle:

```
Prancer = 1,
Cupid = 2,
Rudolph = 3,
Dasher = 4,
Blitzen = 5,
Vixen = 6
Comet = 7,
```

```
  Donder = 8,
  Dancer = 9,
```

In the first solution, we defined transitivity of the `is_behind` rule using the `follows` rule. In this solution, we equate reindeer to numbers and rely on the transitivity of comparison operators to arrive at the solution! We didn't even need to write an algorithm to solve the riddle, we only needed to declare the riddle constraints, and then let prolog find a sequence that fits within said constraints!

**2-solution1.pl**

```
% Vixen should be behind Rudolph, Prancer and Dasher,
is_behind(vixen, rudolph).
is_behind(vixen, prancer).
is_behind(vixen, dasher).

% whilst Vixen should be in front of Dancer and Comet.
is_behind(dancer, vixen).
is_behind(comet, vixen).

% Dancer should be behind Donder, Blitzen and Rudolph.
is_behind(dancer, donder).
is_behind(dancer, blitzen).
is_behind(dancer, rudolph).

% Comet should be behind Cupid, Prancer and Rudolph.
is_behind(comet, cupid).
is_behind(comet, prancer).
is_behind(comet, rudolph).

% Donder should be behind Comet, Vixen, Dasher, Prancer and Cupid.
is_behind(donder, comet).
is_behind(donder, vixen).
is_behind(donder, dasher).
is_behind(donder, prancer).
is_behind(donder, cupid).

% Cupid should be in front of Comet, Blitzen, Vixen, Dancer and Rudolph.
is_behind(comet, cupid).
is_behind(blitzen, cupid).
is_behind(vixen, cupid).
is_behind(dancer, cupid).
is_behind(rudolph, cupid).

% Prancer should be in front of Blitzen, Donder and Cupid.
is_behind(blitzen, prancer).
is_behind(donder, prancer).
is_behind(cupid, prancer).

% Blitzen should be behind Cupid but in front of Dancer, Vixen and Donder.
is_behind(blitzen, cupid).
```

```
is_behind(dancer, blitzen).
is_behind(vixen, blitzen).
is_behind(donder, blitzen).

% Rudolph should be behind Prancer but in front of Dasher, Dancer and Donder.
is_behind(rudolph, prancer).
is_behind(dasher, rudolph).
is_behind(dancer, rudolph).
is_behind(donder, rudolph).

% Finally, Dasher should be behind Prancer but in front of Blitzen, Dancer and Vixen.
is_behind(dasher, prancer).
is_behind(blitzen, dasher).
is_behind(dancer, dasher).
is_behind(vixen, dasher).

follows(Last, First) :- is_behind(Last, First).
follows(Last, First) :- is_behind(Middle, First), follows(Last, Middle).

respects_order([First|[Second]]) :- follows(Second, First).
respects_order([First|[Second|Rest]]) :-
    follows(Second, First), respects_order([Second|Rest]).

solution(Permutation) :-
    findall(Reindeer, (is_behind(Reindeer, _); is_behind(_, Reindeer)), List),
    list_to_set(List, UniqueReindeer),
    permutation(UniqueReindeer, Permutation),
    once(respects_order(Permutation)).
```

3-solution2.pl

```
solution([Vixen, Rudolph, Prancer, Dasher, Dancer, Comet, Donder, Blitzen, Cupid]) :-
    permutation([Vixen, Rudolph, Prancer, Dasher, Dancer, Comet, Donder, Blitzen, Cupid], [1, 2, 3, 4, 5, 6, 7, 8, 9]),
    % Vixen should be behind Rudolph, Prancer and Dasher,
    Vixen > Rudolph, Vixen > Prancer, Vixen > Dasher,

    % whilst Vixen should be in front of Dancer and Comet.
    Vixen < Dancer, Vixen < Comet,

    % Dancer should be behind Donder, Blitzen and Rudolph.
    Dancer > Donder, Dancer > Blitzen, Dancer > Comet,

    % Comet should be behind Cupid, Prancer and Rudolph.
    Comet > Cupid, Comet > Prancer, Comet > Rudolph,

    % Donder should be behind Comet, Vixen, Dasher, Prancer and Cupid.
    Donder > Comet, Donder > Vixen, Donder > Dasher, Donder > Prancer, Donder > Cupid,

    % Cupid should be in front of Comet, Blitzen, Vixen, Dancer and Rudolph.
    Cupid < Comet, Cupid < Blitzen, Cupid < Vixen, Cupid < Dancer, Cupid < Rudolph,

    % Prancer should be in front of Blitzen, Donder and Cupid.
```

```
    Prancer < Blitzen, Prancer < Donder, Prancer < Cupid,

    % Blitzen should be behind Cupid but in front of Dancer, Vixen and Donder.
    Blitzen > Cupid, Blitzen < Dancer, Blitzen < Vixen, Blitzen < Donder,

    % Rudolph should be behind Prancer but in front of Dasher, Dancer and Donder.
    Rudolph > Prancer, Rudolph < Dasher, Rudolph < Dancer, Rudolph < Donder,

    % Finally, Dasher should be behind Prancer but in front of Blitzen, Dancer and Vixen.
    Dasher > Prancer, Dasher < Blitzen, Dasher < Dancer, Dasher < Vixen.
```