Chapter 4
Three Applications in Prolog

We will discuss a few more serious applications in Prolog in this unit, one on database, one on automaton simulator, and another one on puzzle solving.

Let's see how to retrieve structured information from a database first. Assuming that we have the following database

```
family(
 person(tom,fox,date(7,may,1960),works(bbc,15200)),
 person(ann,fox,date(9,may,1961),unemployed),
[person(pat,fox,date(5,may,1983),unemployed),
 person(jim,fox,date(5,may,1983),unemployed)]).
```

To start with, we can refer to all the fox family members with `family(person(_,fox,_,_),_,_)`, and refer to any family with three children with `family(_,_,[_,_,_])`.

To find all the women married with at least three children, we can say

```
?-family(_,person(Name,SurName,_,_),[_,_,_|_]).
```

Among other things, we can define the following queries, sometimes called *selectors*, to get information from the family database.

```
husband(X):-family(X,_,_).
wife(X):-family(_,X,_).
child(X):-family(_,_,C),member(X,C).
exists(P):-husband(P); wife(P); child(P).
dateofbirth(person(_,_,Date,_),Date).
salary(person(_,_,_,works(_,Salary)), Salary).
```

Using the above procedures, we can do even more. For example, find all the children born in 2001.

```
?-child(X),dateofbirth(X,date(_,_,2001))).
```

Find all the employed wives.

```
?-wife(person(Name,SurName,_,works(_,_)).
```

Find all the people born before 1963, and earns less than $50,000.

```
?-exists(P),dateofbirth(P,date(_,_,Y)),
  Y<1963,salary(P,Salary), Salary<50000.
```

To find the household income, we define the following `total` relation first.

```
total([],0).
total([Person|List],Sum):-
 salary(Person,S),
 total(List,Rest),
 Sum is S+Rest.
```

Then

```
?-family(H,W,C),
  total([H,W|C],Income).
```

Let `MinInc` be the minimum family income, we can identify those families whose household income is below this number as follows:

```
?-family(H,W,C),
  total([H,W|C],Income),
  length([H,W|C],N),
  Income/N<MinInc.
```

Homework: Exercises 4.1(b,d), and 4.2.

Automaton simulator

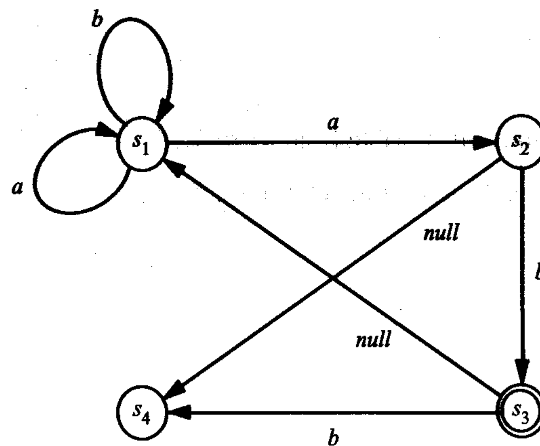
A (non-)deterministic finite automaton, (N)DFA, is an abstract machine that reads in a string of symbols, and decides that if it accepts it, or rejects it. We have covered this stuff in CS3780.

Technically, a DFA consists of an alphabet, a bunch of states, a transition function, and a set of accept states. Starting with a start state, it reads one symbol at a time, based on the current state, and the symbol it is reading, the transition function tells it which state to go to as the next one. When it reads all input symbols, if it is in one of the accept states, it accepts the input; otherwise, it rejects it.

An NDFA works the same way, except that with the same state, and the same input symbol, it can pick a next state among several candidates, and it does not always need an input symbol to go to a new state.

An example

In the following NDFA, if it is given an input string `aabaab`, and starts at s_1, it will accept it. On the other hand, if given `aba`, it will reject it.



Automata theory plays an important role in computational theory as well as compiler construction.

Represent NDFA in Prolog

It is easy to simulate such automata using three relations:

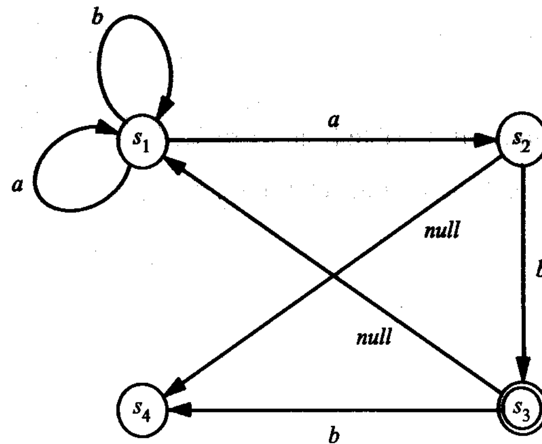
1) a unary relation of *final* to represent the accept states.

2) a *trans* relation to describe the transition rule. For example, `trans(S1,X,S2)` states that in state S_1, if the machines read X, then it may go to state S_2.

3) For NDFA, we use `silent(S1,S2)` to indicate that it may not need any input symbols to go into S_2 from S_1.

An example

Given the following NDFA



we can represent it as follows:

```
final(s3).
trans(s1,a,s1).
trans(s1,a,s2).
trans(s1,b,s1).
trans(s2,b,s3).
trans(s3,b,s4).
silent(s2,s4).
silent(s3,s1).
```

Simulate NDFA in Prolog

Given a machine, and an input string, we want to know if the machine accepts the input, using a relation `accept(State,String)`, as follows:

```
accept(State,[]):-final(State).

accept(State,[X|Rest]):-
 trans(State,X,State1),accept(State1,Rest).

accept(State,String):-
 silent(State,State1),accept(State1,String).
```

Then, if we pose a query `?-accept(S,[a,b])`, the answer will be `S=s1,S=s3`.

Homework: Exercises 4.4 and 4.5.

Eight Queens problem

Everybody should have solved it using some programming language. Let's see how to do it in Prolog.

We will use an unary relation, `solution(Pos)`, to represent the solution. It is true iff `Pos` provides a solution to this problem.

To begin with, we have to represent the chess board configuration to identify the location of all the queens.

One way to do it is to use a list of pairs, `X/Y`, such that $\{X, Y\}$ is the coordinate of a queen in the board.

An example

Thus, the following configuration, also a solution, can be represented with `[1/4,2/2,3/7,4/3,5/6,6/8,7/5,8/1]`.

A strategy

We fix up the template of the solution to be `[1/Y1,2/Y2,3/Y3,4/Y4,5/Y5,6/Y6,7/Y7,8/Y8]`, and generalize the solution relation `solution(Pos)` to be true for any number of queens. Then, it is clear that `solution(Pos)` is true if `Pos=[]`, i.e.,

```
solution([]).
```

In general, if `Pos=[X/Y|Others]`, then `Pos` provides a solution, if `Others` provides one for all the cells other than the one with its index being `X/Y`, where `Y` is anywhere between 1 and 8; and `X/Y` does not attack any queens in the `Others`.

```
solution([X/Y|Others]):-
  solution(Others),
  member(Y,[1,2,3,4,5,6,7,8]),
  noattack(X/Y,Others).
```

When `X/Y` does not attack `Others`?

The queen at `X/Y` does not attack any other queen at `X1/Y1` in `Others`, if

1. They are not at the same row, i.e., `X =\= X1`, which is guaranteed(?).

2. They are not at the same column, i.e., `Y =\= Y1`.

3. They are not at the same diagonal, i.e., their distance in the x direction is not the same as that in the y direction, i.e., `Y-Y1 =\= X-X1`, and `Y-Y1 =\= X1-X`.

As an example, for `X/Y=8/1`, `X1/Y1=6/3` and `X2/Y2=4/1`, `X/Y` and `X1/Y1` are in the same diagonal, so are `X1/Y1` and `X2/Y2`.

First solution

```
solution([]).

solution([X/Y|Others]):-
  solution(Others),
  member(Y,[1,2,3,4,5,6,7,8]),
  noattack(X/Y,Others).

noattack(_, [] ).
noattack(X/Y,[X1/Y1|Others]):-
  Y=\=Y1, Y1-Y=\=X1-X, Y1-Y=\=X-X1,
  noattack(X/Y,Others).

member(Item,[Item|Rest]).
member(Item,[First|Rest]):-
  member(Item,Rest).

template([1/Y1,2/Y2,3/Y3,4/Y4,
          5/Y5,6/Y6,7/Y7,8/Y8]).

?-template(S), solution(S).
```

A couple of thoughts

1. In representing the template, we can drop the `X` coordinate. Then, the solution is simply a permutation of $\{1, 2, 3, 4, 5, 6, 7, 8\}$.

2. Such a permutation will be a solution of the problem, if all the queens so represented are *safe.*

A list `S` is safe, either because it is empty; or when `S=[Queen|Others]`, the queens in the `Others` are safe, and `Queen` is not attacking any of the queens in `Others`.

When `X/Y` does not attack `Others`?

3. The non-attacking relation is a bit trickier. `noattack(Queen,Others)` ensures that `Queen` will not attack any queen in `Others` along either of the *proper* diagonals. Now, although we don't have $x-$axis information any more, it is still there explicitly: The first queen is in row 1, the second row 2, etc..

We introduce `noattack(Queen,Others,Xdist)` to take care of the new situation.

Noticing we start with putting the first queen in the first column. Hence, for the first queen in `Others`, the distance difference in both x and y direction is not equal to 1, for the second, neither difference can be 2, etc..

Second solution

```
solution(Queens):-
 permutation([1,2,3,4,5,6,7,8],Queens),
 safe(Queens).

safe([]).
safe([Queen|Others]):-
 safe(Others),
 noattack(Queen,Others,1).

noattack(_,[],_). %No one attacks nothing.
noattack(Y,[Y1|Ylist],Xdist):-
 Y1-Y=\=Xdist, Y-Y1=\=Xdist,
 Dist1 is Xdist + 1,
 noattack(Y,Ylist,Dist1).
```

The `permutation` relation is the same as defined before.

Prolog project 2: Complete Exercise 4.7.