CS370

Symbolic Programming
Declarative Programming

LECTURE 6: Using Structures

Jong C. Park
park@cs.kaist.ac.kr
Computer Science Department
Korea Advanced Institute of Science and Technology
http://nlp.kaist.ac.kr/~cs370

Using Structures

⊙Example Programs

- **Retrieving structured information from a database**
- **Doing data abstraction**
- **Simulating a non-deterministic automaton**
- **Travel agent**
- **The eight queens problem**

Retrieving structured information

- A database can be represented in Prolog as a set of facts.

```
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)]).
...
...
...
```

Retrieving structured information

- Sample queries

```
?- family(person(_,armstrong,_,_),_,_).

?- family(_,_,[_,_,_]).

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

Retrieving structured information

- A set of procedures defined as a utility

```
husband(X) :- family(X,_,_).
wife(X) :- family(_,X,_).
child(X) :- family(_,_,Children), member(X,Children).

exists(Person) :- husband(Person);
                  wife(Person);
                  child(Person).

dateofbirth(person(_,_,Date,_),Date).
salary(person(_,_,_,works(_,S)),S).
salary(person(_,_,_,unemployed),0).
```

Retrieving structured information

- Sample queries to the database

```
?- exists(person(Name,Surname,_,_)).
?- child(X),dateofbirth(X,date(_,_,2000)).
?- wife(person(Name,Surname,_,works(_,_))).

?- exists(person(Name,Surname,date(_,_,Year),
   unemployed)),Year < 1973.

?- exists(Person),dateofbirth(Person,date(_,_,Year)),
   Year < 1960, salary(Person,Salary), Salary < 8000.

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

Retrieving structured information

- Total income of a family

  total(List_of_people, Sum_of_their_salaries)

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

?- family(Husband,Wife,Children),
    total([Husband,Wife|Children],Income).
```

Doing data abstraction

Data abstraction

- A process of organizing various pieces of information into natural units, structuring the information into a conceptually meaningful form.
- All the details of implementing such a structure should be to the user of the structure.

Doing data abstraction

- Example selectors

```
FoxFamily = family(person(tom,fox,_,_),_,_).
% selector_relation(Object,Component_selected)
husband(family(Husband,_,_),Husband).
wife(family(_,Wife,_),Wife).
children(family(_,_,ChildList),ChildList).
secondchild(Family,Second) :-
   children(Family,[_,Second|_]).
firstname(person(Name,_,_,_),Name).
surname(person(_,Surname,_,_),Surname).
born(person(_,_,Date,_),Date).
```

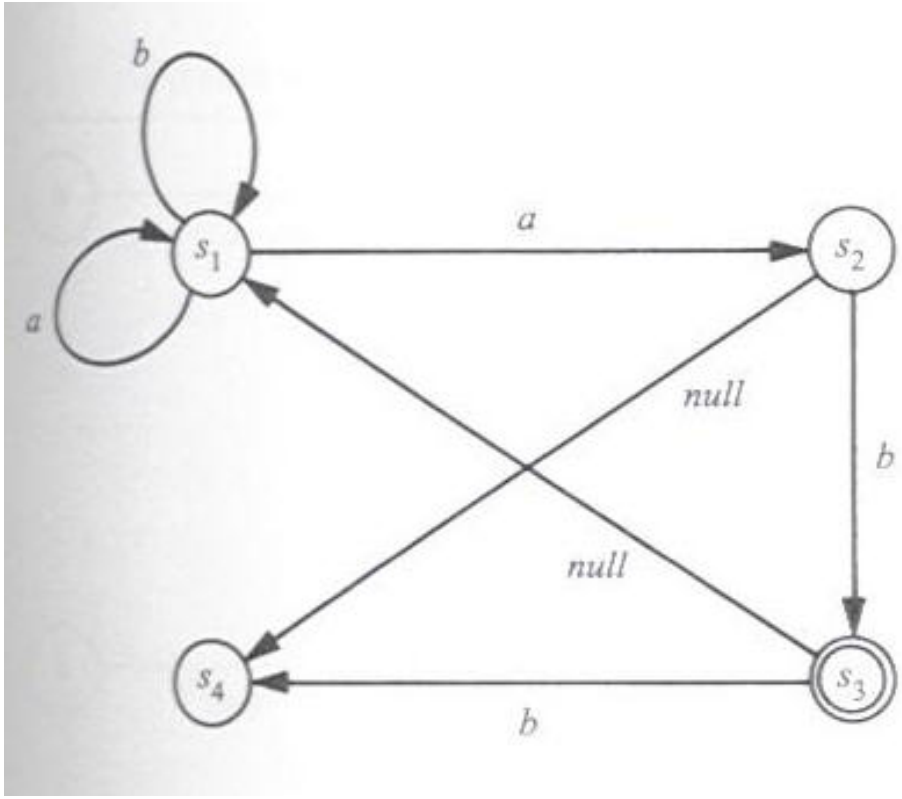
Doing data abstraction

- Example use of selectors

```
?- firstname(Person1,tom),
   surname(Person1,fox),
   firstname(Person2,jim),
   surname(Person2,fox),
   husband(Family,Person1),
   secondchild(Family,Person2).
```

Simulating an NFA

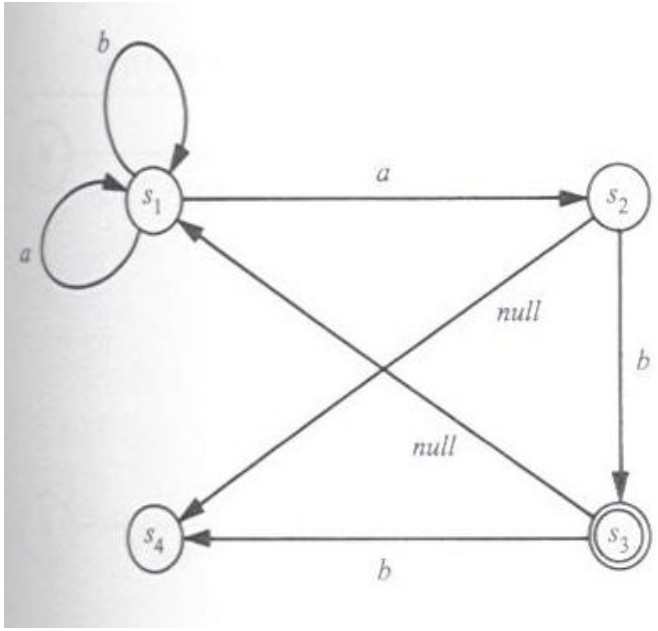
- Example NFA



Simulating an NFA

- Automaton specification in Prolog
  - a unary relation final(S)
  - a three-argument relation trans(S1,X,S2)
  - a binary relation silent(S1,S2)

```
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,s4).
```



Simulating an NFA

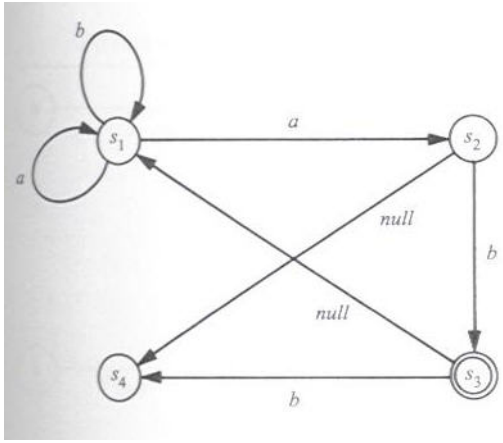
- The simulator is programmed as a binary relation accepts(State,String).

```
accepts(State,[ ]) :-
   final(State).
accepts(State,[X|Rest]) :-
   trans(State,X,State1),
   accepts(State1,Rest).
accepts(State,String) :-
   silent(State,State1),
   accepts(State1,String).
```

Simulating an NFA

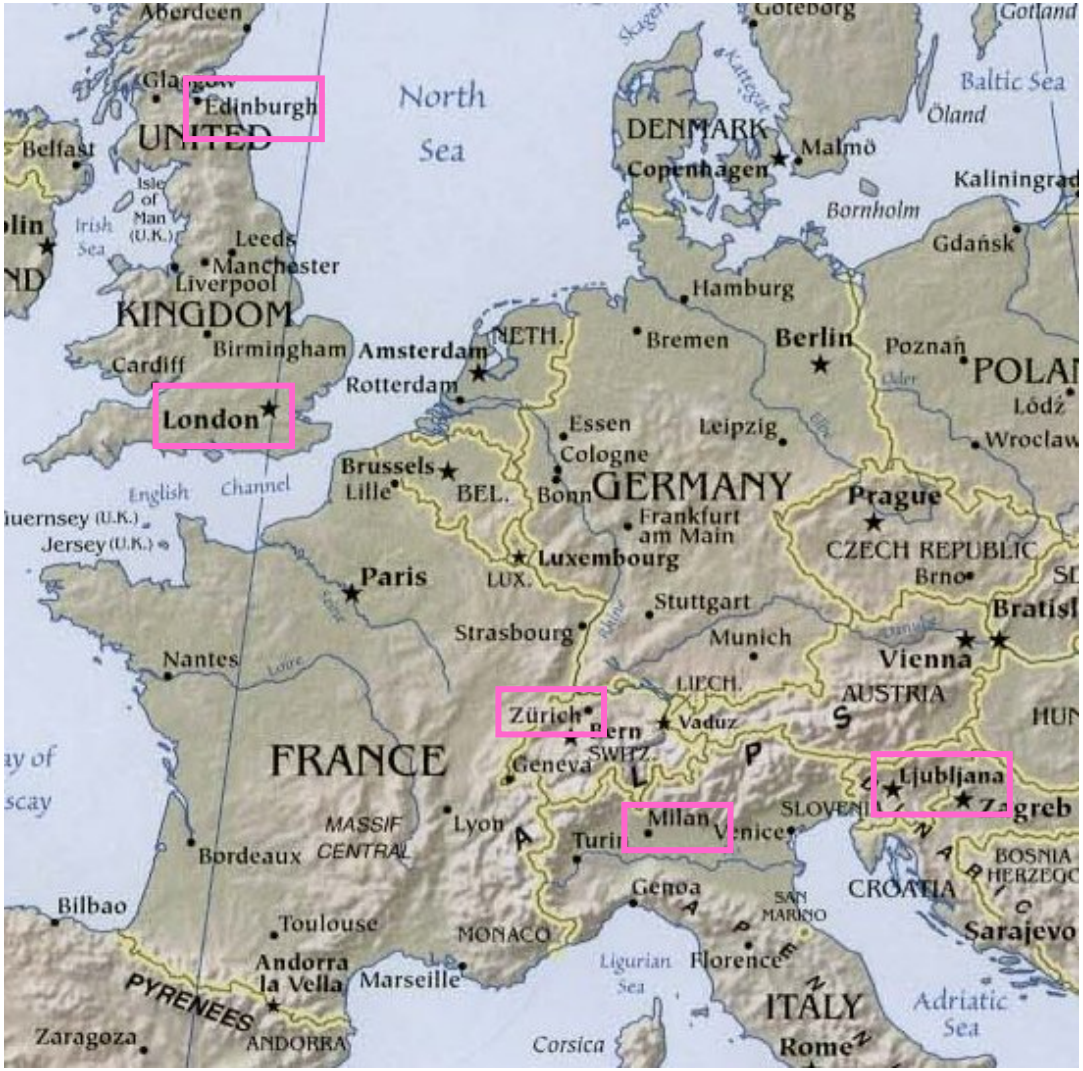
- Example use of the simulator

```
?- accepts(s1,[a,a,a,b]).
yes
?- accepts(S,[a,b]).
S = s1;
S = s3
```



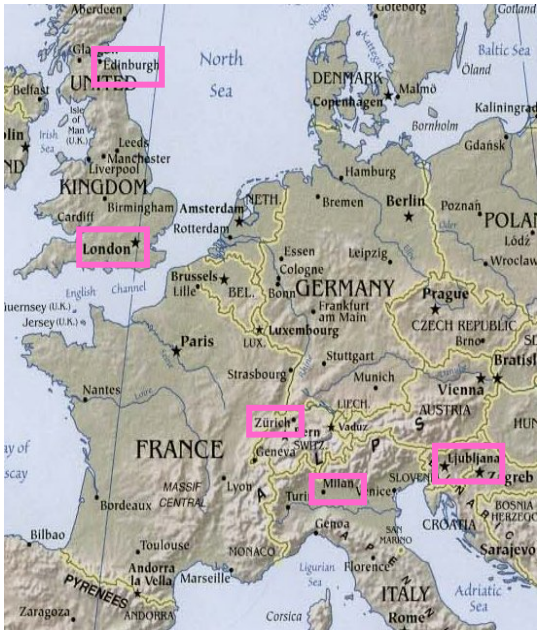

```
?- accepts(s1,[X1,X2,X3]).
X1 = a
X2 = a
X3 = b;
X1 = b
X2 = a
X3 = b;
no
```

Travel Agent



Travel Agent

- Advice on planning air travel
  - What days of the week is there a direct evening flight from Ljubljana to London?
  - How can I get from Ljubljana to Edinburgh on Thursday?
  - I have to visit Milan, Ljubljana and Zurich, starting from London on Tuesday and returning to London on Friday. In what sequence should I visit them?



Travel Agent

- Sample database with the flight information
  - timetable(Place1,Place2,ListOfFlights)
  - ListOfFlights
    - a list of structured items of the form:

      DepartureTime/ArrivalTime/FlightNumber/ListOfDays
  - ListOfDays
    - either a list of weekdays or the atom alldays
- Example
  - timetable(london, edinburgh,
    [9:40/10:50/ba4733/alldays,
    19:40/20:50/ba4833/[mo,tu,we,th,fr,su]]).

Travel Agent

- Exact routes between two given cities on a given day of the week:

  route(Place1,Place2,Day,Route)

  - Route is a sequence of flights such that:
    - the start point of the route is Place1;
    - the end point is Place2;
    - all the flights are on the same day of the week, Day;
    - all the flights in Route are in the timetable relation;
    - and there is enough time for transfer between flights.

Travel Agent

- The route is represented as a list of structured objects of the form:

  From / To / FlightNumber / Departure_time

  - Auxiliary predicates

    flight(Place1,Place2,Day,FlightNum,DepTime,ArrTime)

    deptime(Route,Time)

    transfer(Time1,Time2)

Travel Agent

Encoding the route relation

- Direct flight connection
  - If there is a direct flight between Place1 and Place2 then the route consists of this flight only:

```
route(Place1,Place2,Day, [Place1/Place2/Fnum/Dep]) :-
        flight(Place1,Place2,Day,Fnum,Dep,Arr).
```

- Indirect flight connection
  - The route between P1 and P2 consists of the first flight, from P1 to some intermediate place P3, followed by a route between P3 to P2.
  - There is also enough time for transfer.

```
route(P1,P2,Day,[P1/P3/Fnum1/Dep1|RestRoute]) :-
        route(P3,P2,Day,RestRoute),
        flight(P1,P3,Day,Fnum1,Dep1,Arr1),
        deptime(RestRoute,Dep2), transfer(Arr1,Dep2).
```

Travel Agent

- A Flight Route Planner
  - Figure 4.5

Travel Agent

- Sample questions

```
?- flight(ljubljana,london,Day,_,DeptHour:_,_),
   DeptHour >= 18.
Day = mo;
Day = we;
...
?- route(ljubljana,edinburgh,th,R).
R = [ljubljana / zurich / jp322 / 11:30, zurich / london /
   sr806 / 16:10, london /edinburgh / ba4822 / 18:40]
```

Travel Agent

- Sample question

```
?- permutation([milan,ljubljana,zurich],[City1,City2,City3]),
   flight(london,City1,tu,FN1,_,_),
   flight(City1,City2,we,FN2,_,_),
   flight(CIty2,City3,th,FN3,_,_),
   flight(City3,london,fr,FN4,_,_).
City1 = milan
City2 = zurich
City3 = ljubljana
FN1 = ba510
FN2 = sr621
FN3 = jp323
FN4 = jp211
```

Travel Agent

- Indefinite loops
  - ?- route(moscow,edinburgh,mo,R).
  - How do we address this problem?
    - Use conc

```
?- conc(R,_,[_,_,_,_]),
   route(moscow,edinburgh,mo,R).
no
```

    - Any other ways?
      -

The eight queens problem

- The problem
  - to place eight queens on the empty chessboard in such a way that no queen attacks any other queen.
- The solution
  - programmed as a unary predicate solution(Pos), which is true iff Pos represents a position with eight queens that do not attack each other.

The eight queens problem

- Program 1
  - Figure 4.6
  - Find a list of the form [1/Y1,2/Y2,...,8/Y8].
  - The solution relation has two cases.
    - The list of queens is empty.
    - The list of queens is non-empty: [X/Y|Others]

    where there must be no attack between the queens in the list Others; X and Y must be integers between 1 and 8; and a queen at square X/Y must not attack any of the queens in the list Others.

The eight queens problem

- Program 1
  - Figure 4.7

```
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).
```

The eight queens problem

Program 2

- Omit the X-coordinates: [Y1,Y2,...,Y8]
- The solution is then a permutation of the list [1,2,3,4,5,6,7,8].

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

- Figure 4.9

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

The eight queens problem

- Program 3
  - Figure 4.11
  - Each queen must be placed in a different column, a different row, a different upward and a different downward diagonal: x, y, u, v where u = x - y and v = x + y.
  - Select the position of the first queen, delete the corresponding items from the four domains, and then use the rest of the domain for placing the rest of the queens.

Summary

- **Prolog database**
  - a set of Prolog facts
- **Data abstraction**
  - easier use of complex data structures
  - clear programs