Artificial Intelligence

Lecture 21

- **Prolog Programming for AI**

Prepared by:

Md. Mijanur Rahman, Prof. Dr.

Dept. of CSE, Jatiya Kabi Kazi Nazrul Islam University

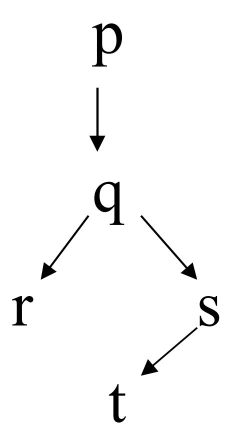
Email: mijanjkkniu@gmail.com

Lecture Outlines

- **Prolog Programs:**
  - Prolog Programs on Tree and Graph Problems.
  - Prolog Programs for DFA and NFA

Example-20: Prolog program for Tree…

- Consider acyclic directed graph:



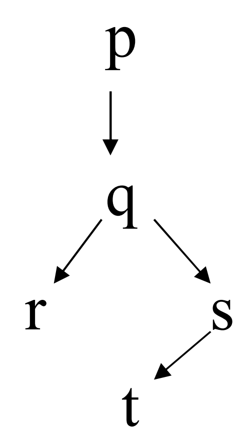
- Graph G is represented by a set of connected edges:

  G = {edge(p, q), edge(q, r), edge(q, s), edge(s, t)}
- Write Prolog program to check whether there is a route from one node to another node.

Example-20: Prolog program for Tree

- **Define Route:**
- Route from node A to B is defined as follows:
  - Route from A to B, if there is an edge from A some node C and a route from C to B.
  - Route from A to B, if there is a direct edge from A to B.
- **Prolog Program:**

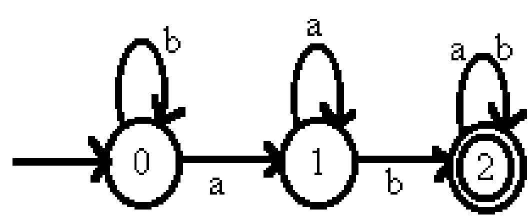
```
edge(p, q).
edge(q, r).
edge(q, s).
edge(s, t).
route(A, B) :- edge (A, C), route(C, B).
route(A, B) :- edge(A, B).
```



- **Query:** Check whether there exist a route between p and t.

Example-21: Prolog program for Graph & DFA…

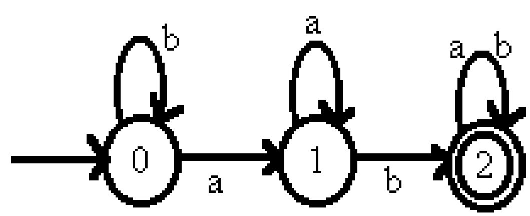
- When a state table program are loaded into Prolog, the parser is used to check whether inputs to the DFA are acceptable or not.
- Consider a state diagram for a DFA that accepts the language (a,b)*ab(a,b)* is as follows:



- Write a program to simulate a parser for an arbitrary deterministic finite automaton (DFA).

Example-21: Prolog program for Graph & DFA…

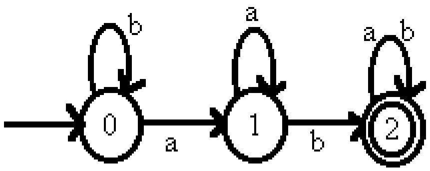
- In Prolog, an automaton can be specified by three relations:
  1) A unary relation '**start**' which defines the initial state of the automation;
  2) A unary relation '**final**' which defines the final states of the automation;
  3) A three-argument relation '**delta**' which defines the state transitions so that: **delta(S1, X, S2);** this means that a transition from S1 to S2 is possible when the current input symbol X is read.



Example-21: Prolog program for Graph & DFA…

- **Prolog code:**
- **Facts:**

```
start(0).
delta(0,a,1).
delta(0,b,0).
delta(1,a,1).
delta(1,b,2).
delta(2,a,2).
delta(2,b,2).
final(2).
```



Example-21: Prolog program for Graph & DFA…

- The simulator is programmed as a unary relation `parse (L)` and a binary relation `trans(S,L)`:
- The `parse (L)` relation can be defined by:
  1) Starting from the initial state S and calling the relation `trans(S,L)`.
- The `trans(S,L)` relation can be defined by two clauses:
  1) The empty string, [], is accepted from a state S if S is a final state.
  2) A non-empty string is accepted from S if reading the first symbol in the string can bring the automation into some state, S1 and the rest of the string is accepted from S1.

- **Prolog Rules:**

```
parse(L)    :- start(S), trans(S,L).
trans(S,[]):-  final(S), write(S), write(' '), write([]), nl.
trans(S,[A|B]) :-  delta(S,A,S1), /* S ---A---> S1 */
                   write(S), write(' '),
                   write([A|B]), nl, trans(S1,B).
```

Example-21: Prolog program for Graph & DFA

- Suppose that both the driver program and the state table program are loaded ...
- `?- parse([b,b,a,a,b,a,b]).`

```
0 [b,b,a,a,b,a,b]
0 [b,a,a,b,a,b]
0 [a,a,b,a,b]
1 [a,b,a,b]
1 [b,a,b]
2 [a,b]
2 [b]
2 []
yes
```

```
?- parse([b,b,a]).
   0 [b,b,a]
   0 [b,a]
   0 [a]
   no
```

Example-22: Prolog program for Graph & NFA…

- Simulating a NFA in Prolog, an automaton can be specified by three relations:
  1) A unary relation ‘**final**’ which defines the final states of the automation;
  2) A three-argument relation ‘**trans**’ which defines the state transitions so that:
     trans(S1, X, S2)

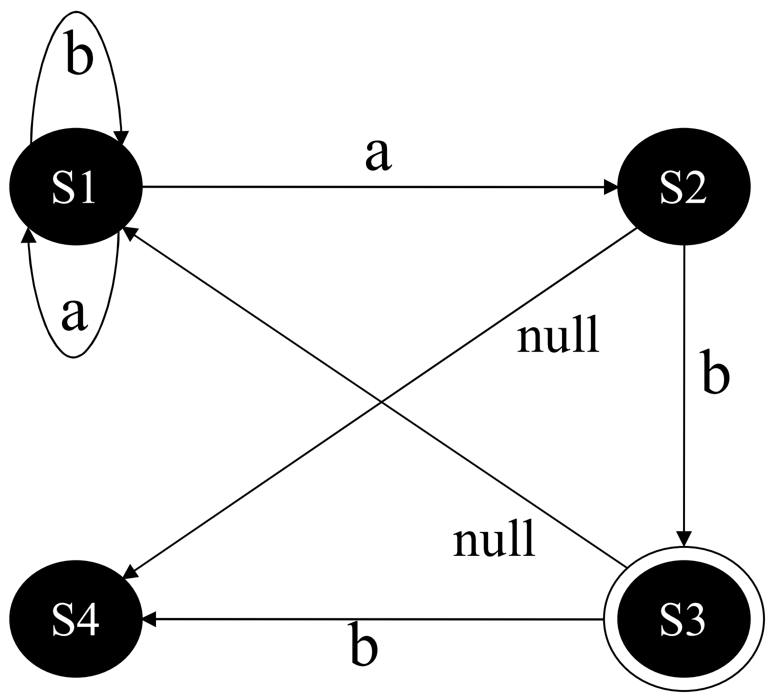
     This means that a transition from S1 to S2 is possible when the current input symbol X is read;
  3) A binary relation ‘**silent**’

     Silent(S1, S2)

     Meaning that a silent move is possible from state S1 to S2.

Example-22: Prolog program for Graph & NFA…

- Consider the following non-deterministic finite automaton:



- For the given automation, the three relations are:
  - 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).

Example-22: Prolog program for Graph & NFA…

- We will represent input strings as Prolog lists. So, the string 'aab' will be represented by [a, a, b].
- The simulator is programmed as a binary relation, **accepts(S, X).**
- The '**accepts**' relation can be defined by three clauses:
  1) The empty string, [], is accepted from a state **S** if State is a final state.
  2) A non-empty string **X** is accepted from **S** if reading the first symbol in the string **X** can bring the automation into some state, **S1** and the rest of the string is accepted from **S1**.
  3) A string is accepted from State if the automation can make a silent move from **S** to **S1** and then accept the whole input string from **S1**.

Example-22: Prolog program for Graph & NFA…

- These rules can be translated into Prolog as:

```
accepts(String, []) :-              % Accept empty string
                  final(S).
accepts(S, [X | R]) :-      % Reading 1st symbol
                  trans(S, X, S1), accepts(S1, R).
accepts(S, String) :-           % Silent move
                  silent(S, S1), accepts(S1, String).
```

Example-22: Prolog program for Graph & NFA

- **Query**:
- Acceptance of the input string *aaab*:

  ?- accepts(s1, [a,a,a,b]).

  yes
- Initial state for input string *ab*:

  ?- accepts(S, [a,b]).

  S = s1;

  S = s3
- Find the string of length 3 that are accepted from state1.

  X1 = a X2 = a X3 = b;

  X1 = b X2 = a X3 = b;

Prolog Programming for AI

THE END