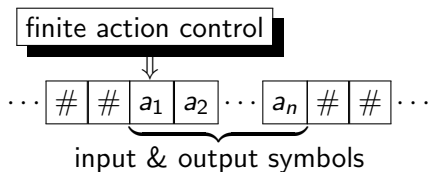
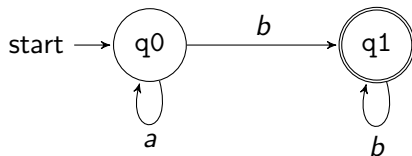


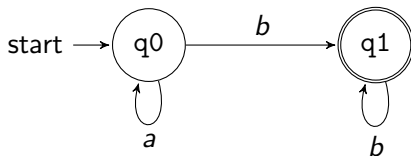
CHURCH-TURING THESIS: Program \approx Turing machine



Finite state machine (fsm)



Finite state machine (fsm)



A **fsm** M is a triple $[\text{Trans}, \text{Final}, Q_0]$ where

- Trans is a list of triples $[Q, X, Q_n]$ such that M may, at state Q seeing symbol X , change state to Q_n
- Final is a list of M 's final (i.e. accepting) states
- Q_0 is M 's initial state.

E.g. $\text{Trans} = [[q_0, a, q_0], [q_0, b, q_1], [q_1, b, q_1]]$

$\text{Final} = [q_1]$

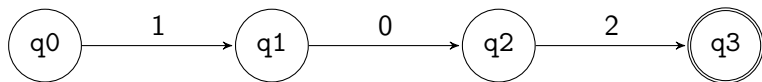
$Q_0 = q_0$

From strings to fsm's

Encode strings as lists; e.g. 102 as [1,0,2].

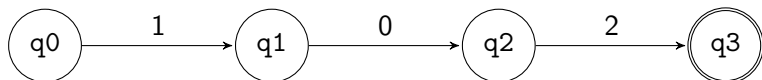
From strings to fsm's

Encode strings as lists; e.g. 102 as $[1,0,2]$.



From strings to fsm's

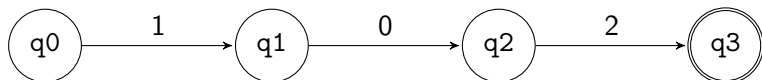
Encode strings as lists; e.g. 102 as [1,0,2].



```
% string2fsm(+String, ?TransitionSet, ?FinalStates)
string2fsm([], [], [q0]).
string2fsm([H|T], Trans, [Last]) :-
    mkTL(T, [H], [[q0, H, [H]]], Trans, Last).
```


From strings to fsm's

Encode strings as lists; e.g. 102 as [1,0,2].



```
% string2fsm(+String, ?TransitionSet, ?FinalStates)
string2fsm([], [], [q0]).
string2fsm([H|T], Trans, [Last]) :-
    mkTL(T, [H], [[q0, H, [H]]], Trans, Last).

% mkTL(+More, +LastSoFar, +TransSoFar, ?Trans, ?Last)
mkTL([], L, Trans, Trans, L).
mkTL([H|T], L, TransSoFar, Trans, Last) :-
    mkTL(T, [H|L], [[L,H,[H|L]]|TransSoFar],
        Trans, Last).
```

States as histories (in reverse)

Exercise

Define a 4-ary predicate

`accept(+Trans,+Final,+Q0,?String)`

that is true exactly when `[Trans,Final,Q0]` is a fsm that accepts `String` (encoded as a list).

Exercise

Define a 4-ary predicate

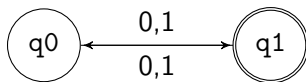
`accept(+Trans,+Final,+Q0,String)`

that is true exactly when `[Trans,Final,Q0]` is a fsm that accepts `String` (encoded as a list).

That is, write a Prolog program to answer queries such as

```
|?- accept([[q0,0,q1],[q0,1,q1],[q1,0,q0],[q1,1,q0]],  
          [q1], q0, [1,0,0]).
```

yes



Exercise

Define a 4-ary predicate

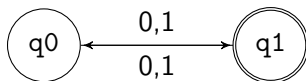
`accept(+Trans,+Final,+Q0,?String)`

that is true exactly when `[Trans,Final,Q0]` is a fsm that accepts `String` (encoded as a list).

That is, write a Prolog program to answer queries such as

```
|?- accept([[q0,0,q1],[q0,1,q1],[q1,0,q0],[q1,1,q0]],  
           [q1], q0, [1,0,0]).
```

yes



```
test(String) :- string2fsm(String, Trans, Final),  
                 accept(Trans, Final, q0, String).
```