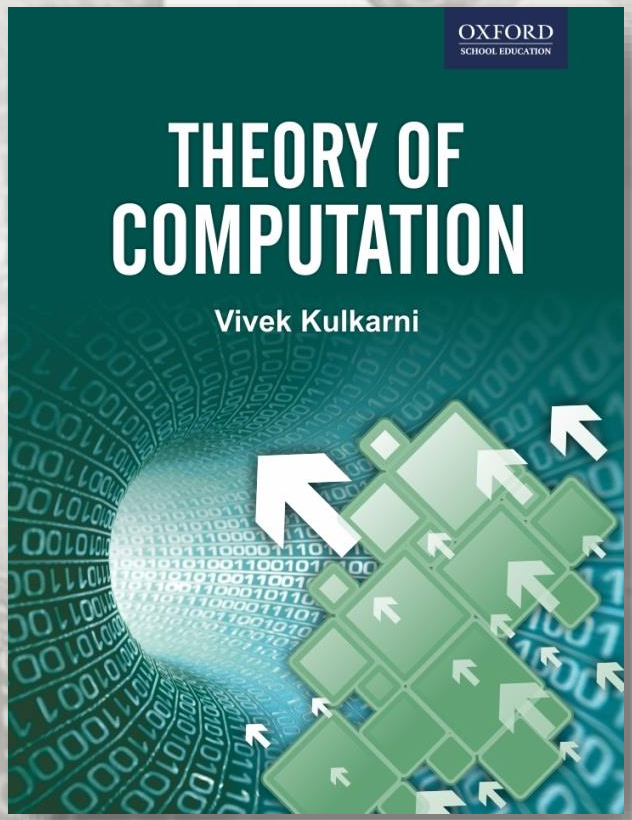
OXFORD
HIGHER EDUCATION

THEORY OF COMPUTATION

Vivek Kulkarni

Slides for Faculty Assistance



Chapter 2

Finite State Machine

Author: Vivek Kulkarni
vivek_Kulkarni@yahoo.com

Chapter 2: Finite State Machine

- Following topics are covered in the slides –
  - Finite State Machine (FSM) – concept and notations
  - Formalism of FSM as finite automata (FA)
  - FA as language acceptor (or recognizer)
  - Deterministic finite automata (DFA) and non-deterministic finite automata (NFA)
  - Equivalence of NFA and DFA
  - NFA with $\in$ -transitions
  - Equivalence of NFA with $\in$ -transitions, NFA, and DFA
  - Moore and Mealy machines and their equivalence
  - Moore's algorithm for FSM equivalence
  - FSM – properties and limitations
  - Two-way Finite Automata (2FA)

Finite State Machine (FSM)

- A **basic machine** can be viewed as a function, which maps the input set, I, to the output set, O. This function is known as a machine function (MAF); it is expected to be an 'onto' relation:

  $$MAF: I \rightarrow O$$

- An **FSM** is represented by a pair of functions, namely:

  Machine function: $MAF: I \times S \rightarrow O$

  State transition function: $STF: I \times S \rightarrow S$

  where, S: Finite set of internal states of the machine

  I: Finite set of input symbols (or input alphabet)

  O: Finite set of output symbols (or output alphabet)

FSM Example

Below are the MAF and STF for a simple binary adder

I \ S	Carry	No- carry
(0,0)	1	0
(0,1)	0	1
(1,0)	0	1
(1,1)	1	0

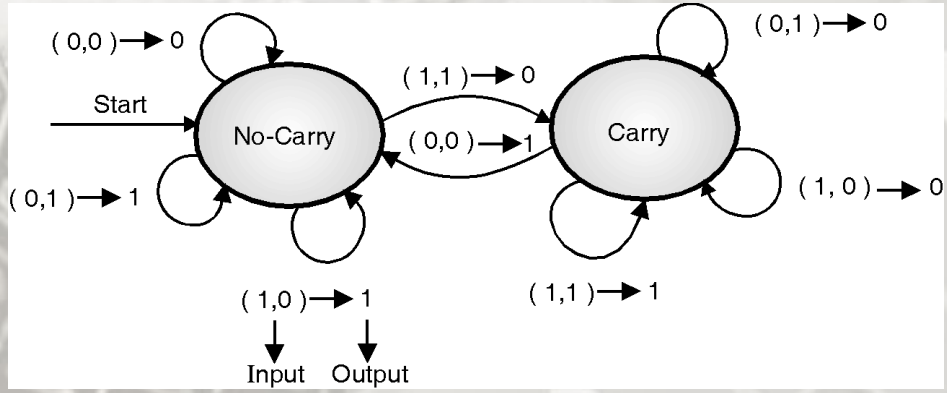
Ou

I \ S	Carry	No-carry
(0,0)	No- carry	No-carry
(0,1)	Carry	No-carry
(1,0)	Carry	No-carry
(1,1)	Carry	Carry

Next states

Transition Graph

- A transition diagram, also known as transition graph, is a directed graph (or digraph), whose vertices correspond to the states of an FSM and the directed edges correspond to the transitions from one state to another on reading the input symbol, which is written above the directed edge. For example, below is the transition graph for a simple binary adder.



Finite Automata (FA)

- **Finite automata** (FA) is the mathematical model (or formalism) of an FSM. Mathematical models of machines are the abstractions and simplifications of how certain machines actually work. They help us understand specific properties of these machines. An FA portrays the FSM as a language acceptor.
- FA is denoted by a five-tuple: $M = (Q, \Sigma, \delta, q_0, F)$

  where,

  Q : Finite set of states

  Σ : Finite input alphabet

  q_0: Initial state of FA, $q_0 \in Q$

  F : Set of final states, $F \subseteq Q$

  δ : STF that maps $Q \times \Sigma$ to Q, i.e., $\delta : Q \times \Sigma \rightarrow Q$
- The above definition of the transition function, δ , is for deterministic FA, or **DFA**. The transition function for non-deterministic FA, or **NFA** is,

  $\delta : Q \times \rightarrow 2^Q$

FA as Language Acceptor

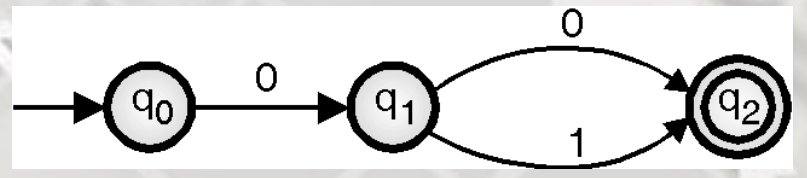
- A string 'x' is said to be accepted by an FA given by: M = (Q, , δ , q_0, F),

  if δ (q_0, x) = p, for some p ∈ F (i.e., p is a member of F)

- If there is a language L such that: L = {x | δ (q_0, x) = p, for some p ∈ F}, then it is said to be accepted by the FA, M, and is denoted by L(M). In such a case the language accepted by automata M is also called a '**regular set**' or '**regular language**'.

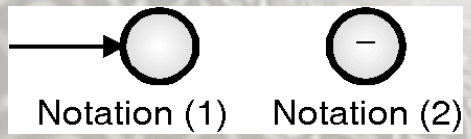
- The FA in the figure accepts,

  L = {00, 01}



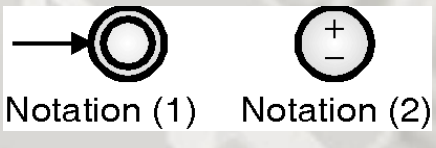
Transition Graph for FA

- The final states of a FA are represented by two concentric circles, instead of a single circle that is used to represent the normal (or non-final) states.
- There is also a slightly different notation that is used at times for representing the transition graph for an FA. Refer figure below.



(a) Initial state

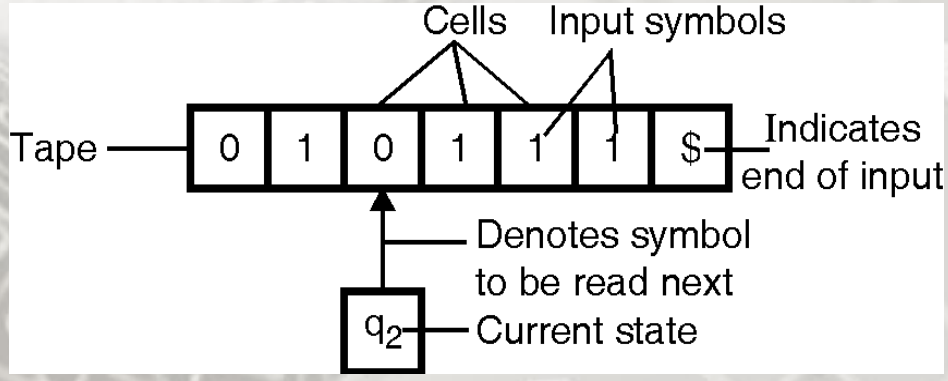
(b) Final state



(c) If initial and final states are same

FA as Finite Control

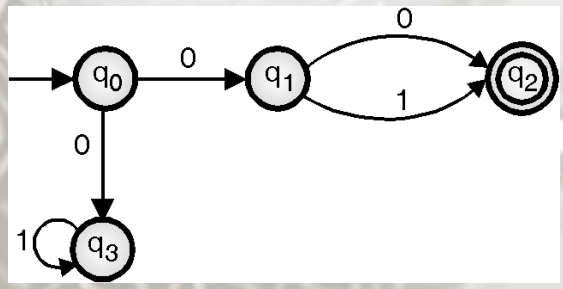
- The FA can be visualized as a finite control with the input string written on a tape, and each tape cell containing one symbol from the string; let '$' denote the end of the string. A pointer or head points to a cell on the tape from which the next symbol will be read. The head is labelled by the state label that represents the current state of the machine. Refer figure below.



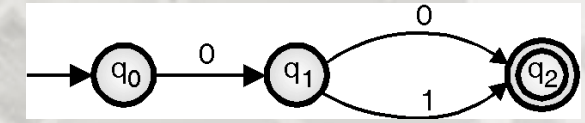
DFA vs NFA

- An FA is said to be deterministic if for every state there is a unique input symbol, which takes the state to the required next unique state.
- In a non-deterministic FA (NFA) model, it is possible to have more than one transition on reading the same input symbol from a given state. Such a machine is also known as a 'possibilistic' machine.

NFA



DFA



- NFA and DFA have equal powers.

NFA to DFA Conversion

- while converting an NFA into its equivalent DFA, let us consider $Q' = 2^Q$, as the set of states for the resulting DFA. Then, the state function for the DFA will be: $\delta' : Q' \times \rightarrow Q'$
- This means that every combination of states can be considered as a new state, and can be given a new label. For example, NFA with $Q = \{q_0, q_1, q_2\}$, $Q' = \{[q_0], [q_1], [q_2], [q_0, q_1], [q_0, q_2], [q_1, q_2], [q_0, q_1, q_2]\}$
- The transition $\delta' ([q_0, q_1], 1)$ can be obtained as:

$$\delta' ([q_0, q_1], 1) = \delta (q_0, 1) \cup \delta (q_1, 1)$$

- A combination of states is considered as a final state for the resultant DFA, if at least one of the states constituting the combination is a final state for the given NFA. The initial state remains unchanged for the resultant DFA.

Unreachable and Trap States

- A state is said to be an '**unreachable state**' if it cannot be reached from the initial state on reading any input symbol. These unreachable states do not take part in string acceptance, and hence must be removed in order to minimize the DFA.
- **Dead states** (or **trap states**) are those non-final (or non-accepting) states whose transitions for every input symbol terminate on them. This means that these states have no outgoing transitions, except to themselves. These are called trap states because once entered, there is no escape (as in the hang state of a computer program). Just as the unreachable states, these dead states and transitions that are incident on these dead states must be removed in order to minimize the DFA.

NFA with ∈ -Transitions

- NFA with ∈ -transitions is an NFA that includes transitions on the empty input symbol, i.e., '∈ ' (epsilon). This is also called an NFA with ∈ -moves.

- **Formal Definition:**

  NFA with ∈ -moves is denoted by a 5-tuple: $M = (Q, \Sigma, \delta, q_0, F)$ where,

  Q: Finite set of states

  Σ : Finite input alphabet

  q_0: Initial state contained in Q

  F: Set of final states $\subseteq$ Q

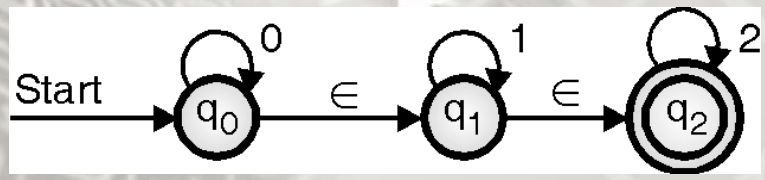
  δ : State function mapping $Q \times (\Sigma \cup \{\in\})$ to 2^Q

  i.e., $\delta: Q \times (\Sigma \cup \{\in\}) \rightarrow 2^Q$

∈-Closure of a State

- The set of all states p, such that there is a path from state q to state p labelled '∈', is known as ∈ -closure (q). In other words, it is the set of all the states having distance zero from state q. It is pronounced as 'epsilon closure of q'.
- For example, let us consider the NFA with ∈ -moves in the figure below,

$$\in \text{-closure } (q_0) = \{q_0, q_1, q_2\}$$



Equivalence of NFA and NFA with ∈ -Transitions

Let us consider an NFA with ∈ -moves, which is given by:

$$M_1 = (Q, \Sigma, \delta, q_0, F)$$

This can be converted to an NFA without ∈ -moves, as follows:

$$M_2 = (Q, \Sigma, \delta', q_0, F')$$

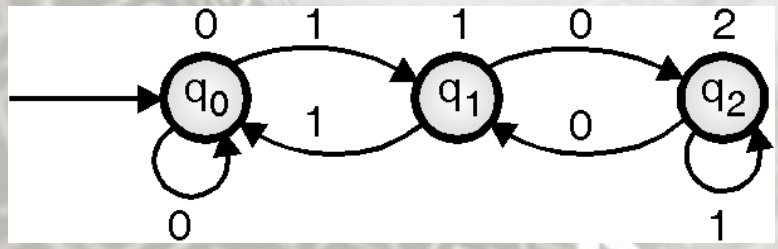
The state transition function, δ', for the required NFA is given as:

$$\delta'(q, a) = \in\text{-closure}(\delta((q, \in), a))$$

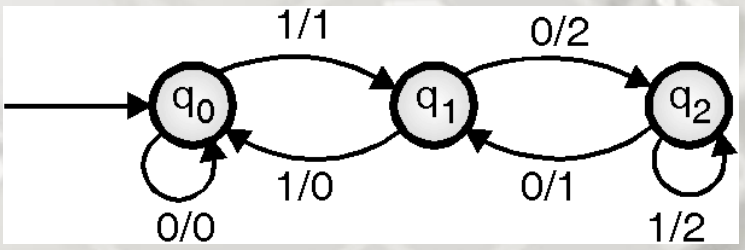
where, $(q, \in) = \in\text{-closure}(q)$

Finite Automata with Output

- There are two different types of the FSM that generate output, namely:
  - Moore machine: Output only depends on the current state
  - Mealy machine: Output depends on current state as well as the symbol read.
  - Moore and Mealy machines are equivalent in power.



(a) Moore machine



(b) Equivalent Mealy machine

Finite Automata with Output

- **Moore machine**: A Moore machine is a machine with finite number of states, and for which the output symbol at any given time depends only upon the current state of the machine (and not on the input symbol read).

  Formal definition: A Moore machine is a six-tuple that is defined as follows:

  $M = (Q, \Sigma, \Delta, \delta, \lambda, q_0)$

  where, Q: Finite set of states

  Σ : Finite input alphabet

  Δ: An output alphabet

  δ: State transition function (STF); $\delta : Q \times \Sigma \rightarrow Q$

  λ: Machine function (MAF); $\lambda : Q \rightarrow \Delta$

  q_0: Initial state of the machine TS: please insert symbols

- **Mealy machine:** The only change in case of a Mealy machine is the machine which is written as,

  λ: Machine function (MAF); $\lambda : Q \times \Sigma \rightarrow \Delta$

Moore to Mealy Conversion

- If a Moore machine is given as: $M_1 = (Q, \Sigma, \Delta, \delta, \lambda, q_0)$ then, its equivalent Mealy machine is: $M_2 = (Q, \Sigma, \Delta, \delta, \lambda', q_0)$ where,

  $\lambda' (q, a) = \lambda (\delta (q, a)) \forall (q \subset Q$ and $\forall a \subset \Sigma$

- The rule stated above, i.e., $\lambda' (q, a) = \lambda (\delta (q, a))$, associates the output of the next state, i.e., $\lambda (\delta (q, a))$ with the transition for the resultant Mealy machine, i.e., $\lambda' (q, a)$.

Mealy to Moore Conversion

- Let us consider a given Mealy machine: $M_1 = (Q, \Sigma, \Delta, \delta, \lambda, q_0)$, then its equivalent Moore machine is: $M_2 = ([Q \times \Delta], \Sigma, \Delta, \delta', \lambda', [q_0, b_0])$ where,

  'b_0' is an arbitrarily selected member of Δ

  $\delta' ([q, b], a) = [\delta (q, a), \lambda (q, a)]$

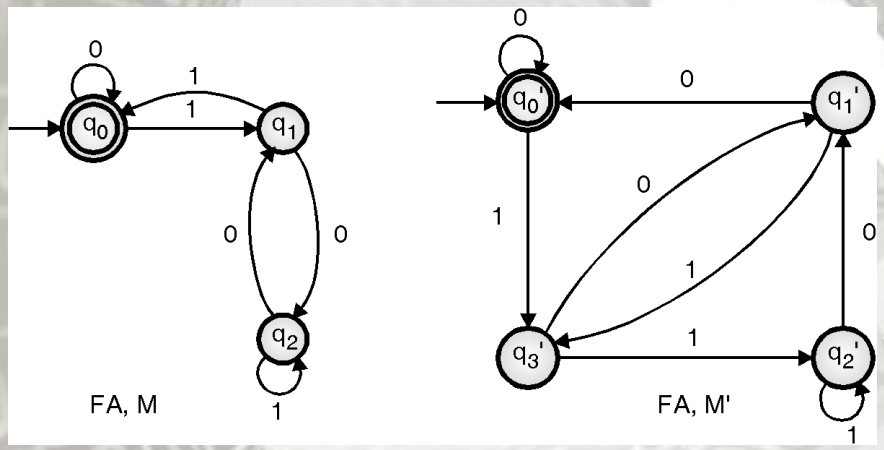
  $\lambda' ([q, b]) = b$ TS: please insert symbols

- This conversion is slightly non-trivial. As we cannot determine the output symbol that the equivalent Moore machine holds in each state; we end up creating all possible combinations of the state symbols and the output symbols, $Q \times \Delta$. It is almost like creating multiple variants of the same state that only differ i n the associated output symbol.

Moore's Algorithm for FSM Equivalence

- If M and M′ are two FSM's over Σ, where | Σ | = n ('n' is the number of input signals); then, apply the following steps to check the equivalence of M and M′:
  - **Step 1:** Construct a comparison table consisting of (n + 1) columns. In column 1, list all ordered pairs of vertices of the form (v, v′), where v ⊂ M, and v′ ⊂ M′. In column 2, list all pairs of the form (v_a, v_a'), if there is a transition labelled 'a' leading from v to v_a and from v′ to v_a'. In column 3, list all pairs of the form (v_b, v_b'), if there is a transition labelled 'b' leading from v to v_b and from v′ to v_b'. Repeat this for all 'n' symbols from .
  - **Step 2:** If the pair (v_a, v_a'), has not occurred previously in column 1, place it in the column 1 and repeat Step 1 for that pair. Repeat Step 2 for each pair, (v_b, v_b'), (v_c, v_c'), and so on.
  - **Step 3:** If in the table, suppose a pair (v, v′) is reached in which, v is the final vertex (or final state) of M, and v′ is a non-final vertex of M′; or vice-versa, then stop and declare that the two FSMs, M and M′, are not equivalent. Otherwise, stop when there are no more new pairs in columns 2, 3, …, n that do not occur in column 1. In this case, the two FSM's M and M′ are equivalent.

Moore's Algorithm Examples

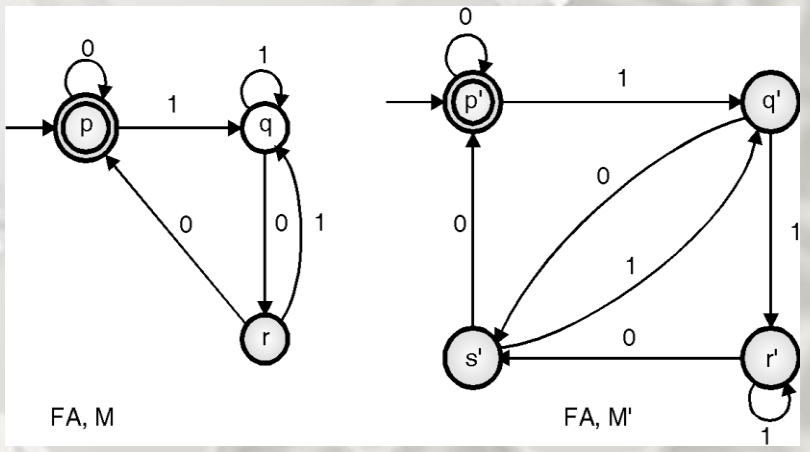


(v, v')	(v_0, v_0')	(v_1, v_1')
(q_0, q_0')	(q_0, q_0')	(q_1, q_3')
(q_1, q_3')	(q_2, q_1')	(q_0, q_2')

Non-equivalent FSMs

(v, v')	(v_0, v_0')	(v_1, v_1')
(p, p')	(p, p')	(q, q')
(q, q')	(r, s')	(q, r')
(r, s')	(p, p')	(q, q')
(q, r')	(r, s')	(q, r')

Equivalent FSMs

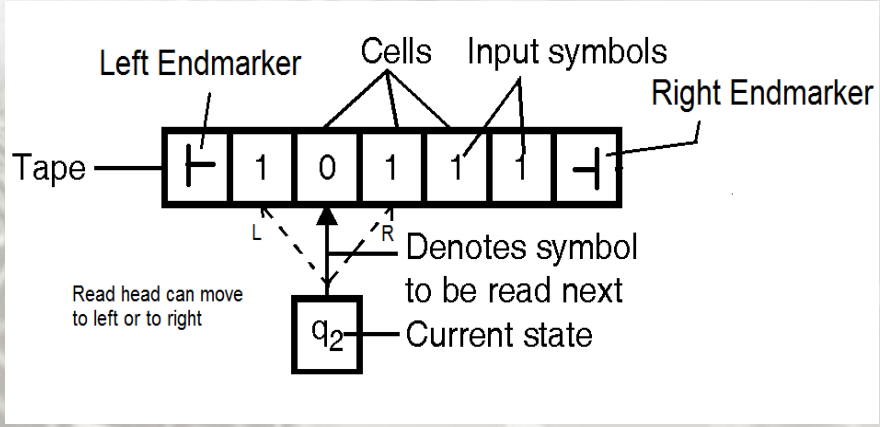


FSM – Properties and Limitations

- **Periodicity:** FSM eventually will always repeat a state or produce a periodic sequence of states.
- **State determination:** Given the initial state and the input sequence, it is always possible to determine the FSM state upon reading the sequence.
- **Impossibility of multiplication:** Multiplication being the repetitive addition, it is required to store the intermediate sums to perform the operation. FSM does not have any memory and hence, cannot perform multiplication.
- **Impossibility of palindrome recognition:** No FSM can recognize a palindrome string, because it does not have the capability to remember all the symbols it reads until the half-way point of input sequence. Hence, it cannot match them in reverse order, with the symbols in second half of the sequence.
- **Impossibility to check if parentheses are well-formed**: As an FSM has no capability to remember the earlier input symbols that it reads, it cannot compare with the remaining input symbols to check for well-formed parentheses.

Two-way Finite Automata (2FA)

- The 2FA machines have a read head that can move in any direction from the current position, either left or right. The left and the right end-markers are the tape bounds, and at any given point in time, the read head cannot move beyond these two bounds. The input tape is thus finite just as that of the FA.
- Though 2FA sounds to be more powerful than the normal single-way FA, its power is equivalent to that of the FA. The 2FA can accept only the regular sets as does the FA.



2DFA vs 2NFA

- A 2FA is formally denoted by an eight-tuple (or octuple):

  $M = (Q, \Sigma, \vdash, \dashv, \delta, q_0, q_A, q_R)$

  Where,

  Q: Finite set of states

  Σ : Finite input alphabet

  q_0: Initial state of FA , $q_0 \in Q$

  q_A: Accept halt (final) state of 2FA, $q_A \in Q$

  q_R: Reject halt (final) state of 2FA, $q_R \in Q$

  $\vdash$: Left end-marker symbol; $\vdash \notin \Sigma$

  $\dashv$: Right end-marker symbol; $\dashv \notin \Sigma$

The 'δ' function for 2DFA is given by:

$$\delta : Q \times (\Sigma \cup \{ \vdash, \dashv \}) \rightarrow (Q \times \{ L, R \})$$

The 'δ' function for 2NFA is given by:

$$\delta : Q \times (\Sigma \cup \{ \vdash, \dashv \}) \rightarrow \text{finite subsets of } (Q \times \{ L, R \})$$