Non-determinism

- An important abstraction in computer science. Refers to situations where next state of a computation is not uniquely determined by the current state.

- Arises when there is incomplete information about the state or when there are external forces at work that can affect the course of a computation. eg. the behaviour of a process in a distributed system might depend on messages that arrive at unpredictable times with unpredictable contents.

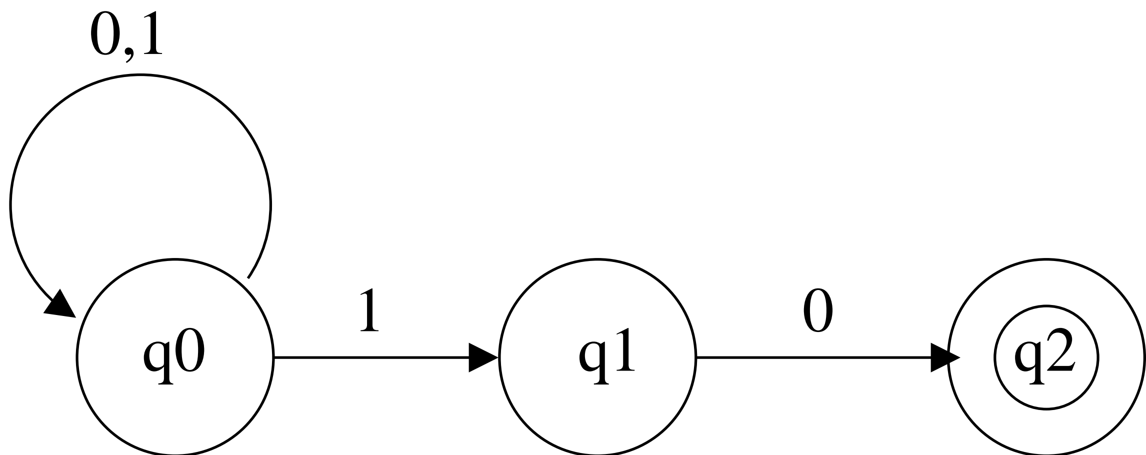
- A system is *deterministic* if, for any possible state it could be in, and for any possible outside stimulus it could receive, it will change to a *unique* pre-determined state.

- A *non-deterministic* system, in a particular state and receiving a particular input, may change to any one of several states.

- Our model of non-deterministic computation is going to say that a computation is "successful" if there is *some* possible way for the system to do what we want.

Non-deterministic FA's

- In non-deterministic situations, we may not know how a computation will evolve, but we should have some idea of the range of possibilities. We model this by allowing an automaton to be in **several states at once**.



- This NFA accepts all and only strings that end in 10.

- Returns to state $q0$ when it "guesses" that final 10 has not yet begun.

- If next symbol is a "1", then it can also "guess" that final 10 has begun.

Non-deterministic FA's

- *Deterministic* automaton, exactly one start state and exactly one transition out of each state for each symbol in Σ.

- Non-deterministic automaton, may be one, more than one, or zero transitions, for each (state, symbol) pair.

- Informally, a non-deterministic automaton accepts a string s if it is *possible* to start in the start state and scan s, moving according to the transition rules, and making choices along the way whenever the next state is not uniquely determined, such that when the end of s is reached, the machine is in an accept state.

- May be several possible paths through the automaton in response to s, some may lead to accept states, and some may lead to reject states.

  The machine accepts s if there is *at least one* path, labelled with s, from some start state to some accepting state.

  The machine rejects s if there is *no* path, labelled with s, from *any* start state to an accepting state.

Formal definition

- Formally,

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

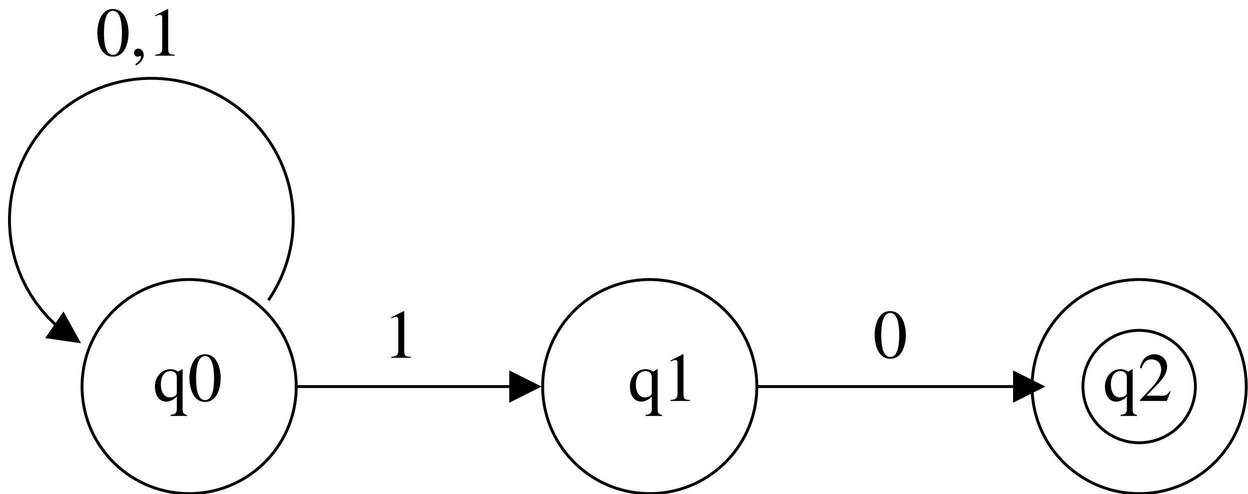
  where everything is the same as in a DFA except for one thing :

  – δ is a function

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

    where 2^Q denotes the *power set* of Q, set of all subsets of Q.

    $\delta(p, a)$ is the set of all states that N is allowed to move to from p, in one step, on input a.



Formal description:

$$(\{q_0, q_1, q_2\}, \{0, 1\}, \delta, q_0, \{q_2\})$$

where δ is given by this table:

	0	1
$\rightarrow q_0$	$\{q_0, q_1\}$	$\{q_0\}$
q_1	$\emptyset$	$\{q_2\}$
q_2 F	$\emptyset$	$\emptyset$

each entry in the table is a **set**.

Extended transition function

- Want to define $\widehat{\delta}$, the function that takes a state q and a string s, and returns the **set** of states that the NFA can reach, if it starts in state q and processes s.

- **Base case:**
  $\widehat{\delta}(q, \epsilon) = \{q\}$

- **Inductive step**
  let s be of the form xa, a is the final symbol of s, and x is the rest.

  Suppose:

$$\widehat{\delta}(q, x) = S = \{p_1, p_2, ..., p_k\}$$

  and

$$\bigcup_i \delta(p_i, a) = \{r_1, r_2, ..., r_m\}$$

  Then:

$$\widehat{\delta}(q, s) = \{r_1, r_2, ..., r_m\}$$

Language of an NFA:

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

$$L(N) = \{s \mid \widehat{\delta}(q_0, s) \cap F \neq \emptyset\}$$

Deterministic simulation

- The action of the NFA gives rise to a computation tree, which branches for each choice that can be made.

- We can run the NFA trying out all the possible choices that can be made, keeping a record of what is happening on each path. A deterministic computation can *interleave* the steps on each path, and eventually decide if s is accepted after trying out all the possibilities.

- Note that an NFA is essentially a tool of *succinctness*. We will use it as a *design tool*.

- In fact, for any NFA, there is a DFA accepting exactly the same set of strings. The DFA may require many more states since the DFA needs to have the "guesses" coded into it somehow.

Subset construction

- Given an NFA, N, think of putting pebbles on the states to keep track of all the states that N could be in after scanning some prefix of the input string.

  Start with some pebbles on the start state(s).

  After scanning some prefix y of the input string, we have pebbles on some set of states P.

  If an input symbol, b, is next, we pick up all the pebbles off the states in P and put them down on each state reachable from the states in P under input b.

  The new set of states P' is the set of states that N can be in after scanning yb.

- P' is uniquely determined by b and the set P. So we have found a *deterministic* way to use the NFA.

- We build a DFA M, whose states are these *sets of states* in N.

  The start state of M will be a singleton set containing the start state in N.

  A final state of M will be any set P containing a *final* state of N.

More formally:

- Start with

$$N = (, Q_N, \Sigma, \delta_N, q_0, F_N)$$

  want to produce

$$D = (Q_D, \Sigma, \delta_D, \{q_0\}, F_D)$$

  with

$$L(D) = L(N)$$

- Q_D is the set of all subsets of Q_N, usually don't need them all.

- Σ, the input alphabet is the same for both N and D.

- Start state of D is the singleton set $\{q_0\}$, containing only the start state of N.

- F_D is the set of all sets of states in N containing at least one accepting state in N.

- For each $S \subseteq Q_N$, and each input symbol a,

$$\delta_D(S, a) = \bigcup_{p \in S} \delta_N(p, a)$$

  Look at all states p in S, see where N goes to from p on input a, take the union of those states.

Proof of correctness

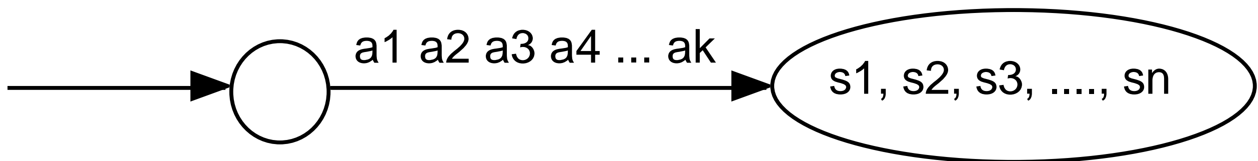
- If M is constructed from N using the subset construction, then M is *deterministic*. For each input symbol a and each state S of M, we defined a specific state T such that M goes from S to T under a.

- But, how do we know that M and N are *equivalent* (accept the same language)?

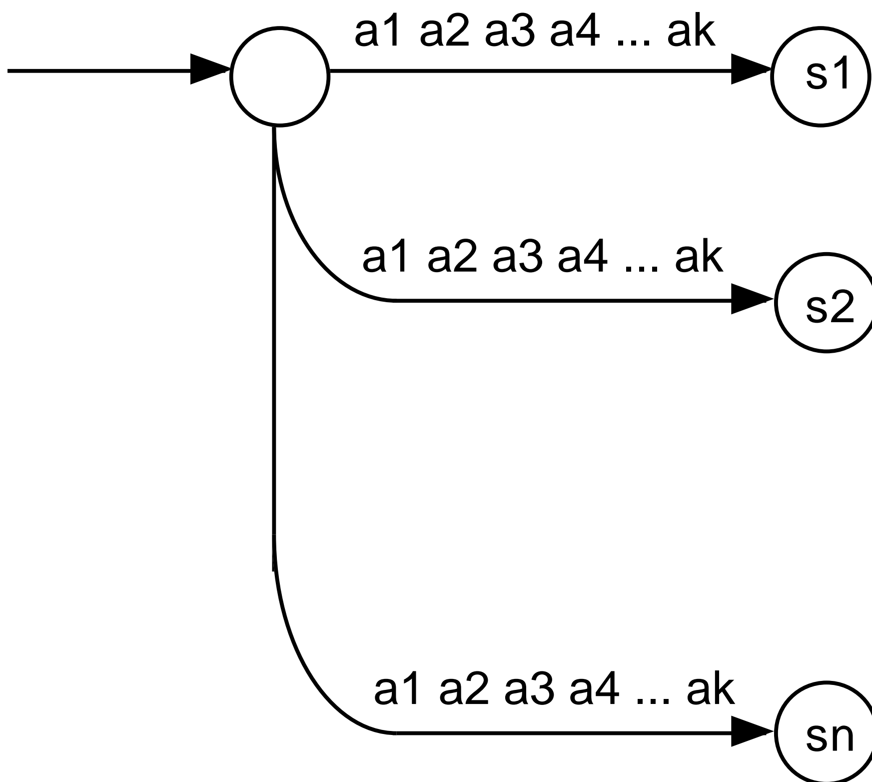
- Need to know that for any input sequence $a_1, a_2, \cdots a_k$, the state T that M reaches on input $a_1, a_2, \cdots a_k$ is an accepting state iff N accepts $a_1, a_2, \cdots a_k$.

- We can prove the relationship by induction on the length of the input string.

Relationship between N and M



Deterministic Automaton



Non-Deterministic Automaton

Inductive proof

- Want to show that the state reached by M on any input sequence $a_1, a_2, \cdots a_k$, is exactly the set of states of N that can be reached by N on input $a_1, a_2, \cdots a_k$.

- Use induction on the length of the input string.

- **Basis**

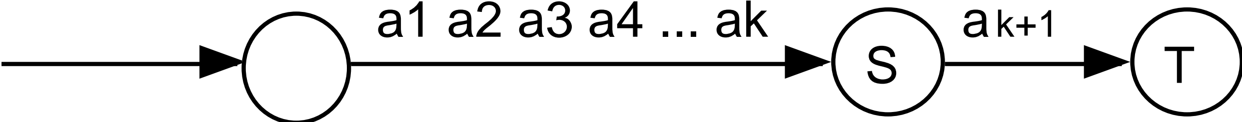
  Let $k = 0$. A path of length 0 leaves us where we started, in the start state of both N and M. The start state of N is contained in the start state of M, so if the start state of N is accepting then the start state of M is accepting.

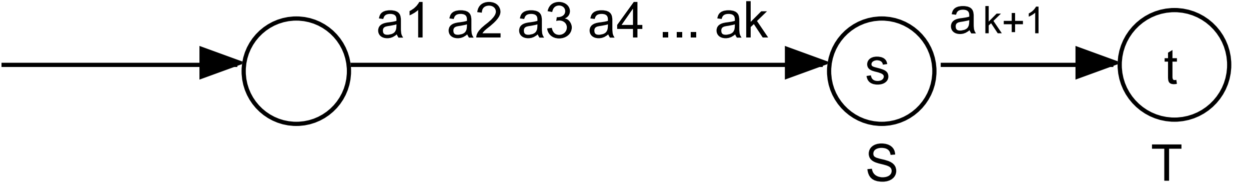
- **Inductive step**

  Suppose the statement holds for k. Consider the input string $a_1, a_2, \cdots a_k, a_{k+1}$.

  We can assume that the state S reached by M on input $a_1, a_2, \cdots a_k$ is exactly the set of states that can be reached by N on input $a_1, a_2, \cdots a_k$.

  We need to show that T, the state that M reaches on input $a_1, a_2, \cdots a_k, a_{k+1}$, is exactly the set of states that can be reached by N on input $a_1, a_2, \cdots a_k, a_{k+1}$.


a1 a2 a3 a4 ... ak
S
a k+1
T


a1 a2 a3 a4 ... ak
s
a k+1
t
S
T

Inductive step

1. To show that T doesn't contain too much.Let t be in T, then there must be s in S that justifies t being in T. That is, $s \stackrel{a_{k+1}}{\rightarrow} t$ for some state s in S.

   By the induction hypothesis, s is reachable in N under $a_1, a_2, \cdots a_k$. So t is reachable in N under $a_1, a_2, \cdots a_k, a_{k+1}$.

2. To show that T contains enough. Suppose t is reachable in N under input $a_1, a_2, \cdots a_k, a_{k+1}$, then there is a state s in N that is reachable under input $a_1, a_2, \cdots a_k$, and $s \stackrel{a_{k+1}}{\rightarrow} t$.

   By the induction hypothesis, s is in S. The subset construction demands that we put t in T.

- Since, the state reached by M on any input sequence $a_1, a_2, \cdots a_k$, is exactly the set of states of N that can be reached by N on input $a_1, a_2, \cdots a_k$, and the accepting states of M are those that include an accepting state of N, we can conclude that M and N accept the same strings.

- The subset construction "works".

Algorithm NFA-to-DFA

Input $N = (Q, \Sigma, S, \Delta, F)$

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

begin
Set Q' to contain $S_0 = \{q_0\}$, a state denoting the set of start states.
Set P to be $\emptyset$, P records the states in the DFA that have been processed.

while $Q' - P \neq \emptyset$ **do**

Select $q \in Q' - P$

if $q \cap F \neq \emptyset$ **then** add q to F' **endif**

for each $a \in \Sigma$ **do**

Set q' to be $\emptyset$
for each state in q **do**
add $\delta(q, a)$ to q'
endfor

if $q' = \emptyset$ **then** set $\delta(q, a) = \emptyset$ (error state in DFA)
else add q' to Q', set $\delta(q, a) = q'$
endif

endfor

endwhile
end