Notes, 9(a)

ECE 606

Non-Determinism and Computational Complexity

State of an algorithm: values in all its storage at any given moment that it is running.

$\textsc{IsIn}(A[1,\ldots,n], i)$
  $ret \leftarrow$ false
  foreach j *from* 1 *to* n **do**
    if $A[j] = i$ **then** $ret \leftarrow$ true
  return ret

The state of the above algorithm characterized by $\langle ret, j \rangle$. Note that our notion of correctness can be specified in terms of state: IsIn is correct if and only if when it halts, the state is $\langle \mathsf{false}, n \rangle$ if $i \notin A$ and $\langle \mathsf{true}, n \rangle$ if $i \in A$.

So far, we have allowed for only the *deterministic* model of computation: an algorithm is in exactly one state at any given moment that it is running.

The *non-deterministic* model of computation differs from the deterministic model in two ways (we focus on decision problems):

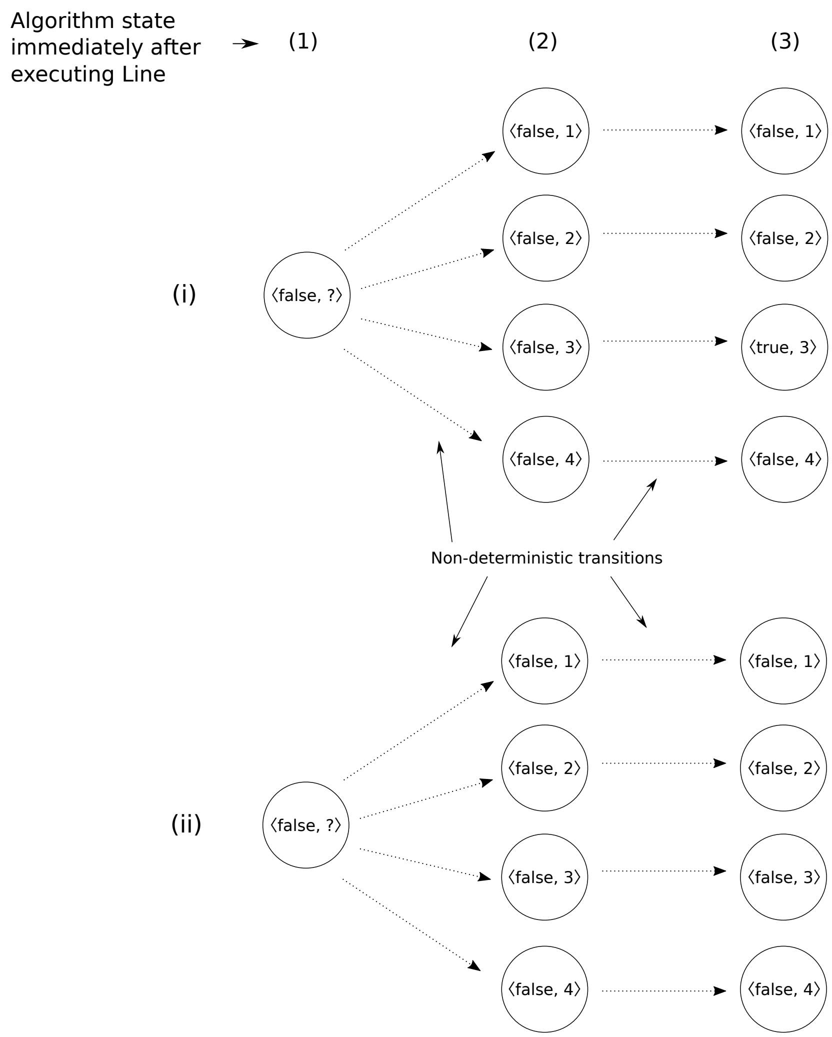
- An algorithm is allowed to simultaneously be in unboundedly many states while it runs, and,
- Notion of algorithm correctness changed to:
  - (i) if the correct output is true, then at least one of the states in which the algorithm is when it halts must correspond to a true output, and,
  - (ii) if the correct output is false, then all of the states in which the algorithm is when it halts must correspond to a false output.

IsIn-withND$(A[1,\ldots,n], i)$

1 $ret \leftarrow$ false
2 Non-deterministically pick $j \in \{1,\ldots,n\}$
3 **if** $A[j] = i$ **then** $ret \leftarrow$ true
4 **return** ret

(Note that the original IsIn can also be considered a non-deterministic algorithm.)

Visualizing the states of IsIn-withND, where each state is a pair $\langle ret, j\rangle$ on input: (i) $\langle [11, 41, 28, 32], 28\rangle$, and, (ii) $\langle [11, 41, 28, 32], 53\rangle$.



Another example: shortest-distance in graphs, decision version.

We will usually restrict ourselves to decision problems in the context of non-deterministic algorithms. Consider the following problem: given input (i) undirected $G = \langle V, E \rangle$, (ii) $a, b \in V$, (iii) $k \in \{0, 1, \ldots, |V| - 1\}$, does there exist a path $a \rightsquigarrow b$ in G of at most k edges?

c: counting the # of edges
u: being the current vertex, it's like the source vertex in BFS.

```
ShortDist-Det(V, E, a, b, k)
    BFS(V, E, a)
    if d[b] ≤ k then
        return true
    else return false
```

```
ShortDist-NonDet(V, E, a, b, k)
    c ← 0, u ← a
    while c ≤ k do
        if u = b then return true
        if Adj[u] = ∅ then return false
        Non-deterministically pick v ∈ Adj[u]
        c ← c + 1, u ← v
    return false
```

There is no difference between deterministic and non-deterministic algorithms from the standpoint of existence.

Claim 1. *A non-deterministic algorithm exists for a problem if and only if a deterministic algorithm exists for it.*

Proof. "if": a deterministic algorithm is a non-deterministic algorithm.

"only if": replace any non-deterministic choices by iterating one-by-one through all deterministic choices. □

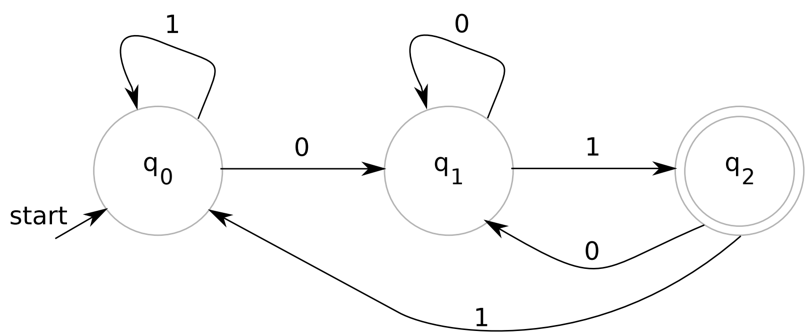
So – the only possible consequence of adopting the non-deterministic model of computation is that perhaps our algorithms are more efficient.

Just to note that the complexity NP stands for Non-Deterministic Polynomial Time.

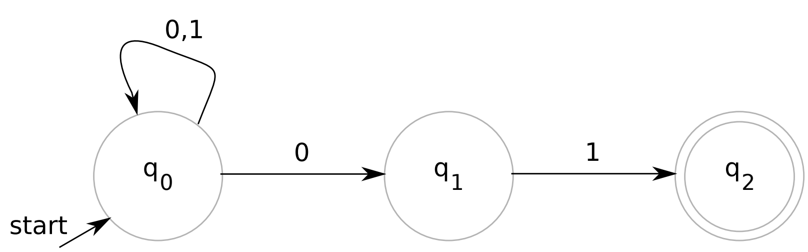
To understand non-determinism better: Deterministic (DFA) and Non-deterministic Finite Automata (NFA). These are restricted algorithms. They are allowed to only:

- Read the input.
- Change state. In NFA's case, non-deterministically.
  - A state is marked as the start state.
  - Some states are marked as "accepting" states. Non-accepting states are "rejecting" states.
  - Question is: in which of those two kinds of states is the FA when it is done reading the entire input.
    * For an NFA, we ask whether there exists a sequence of transitions that ends in an accepting state.

Example: DFA that accepts all binary strings that end in 01.

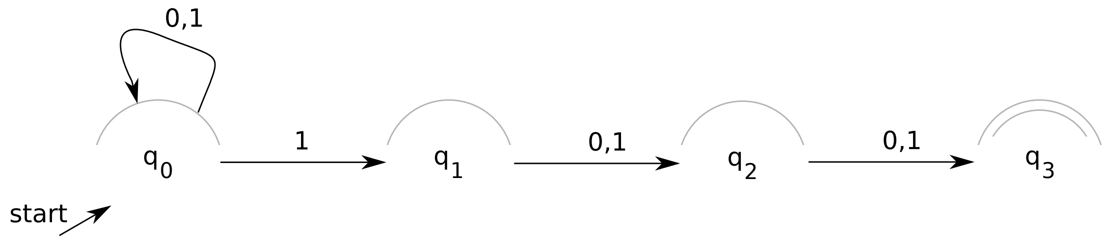


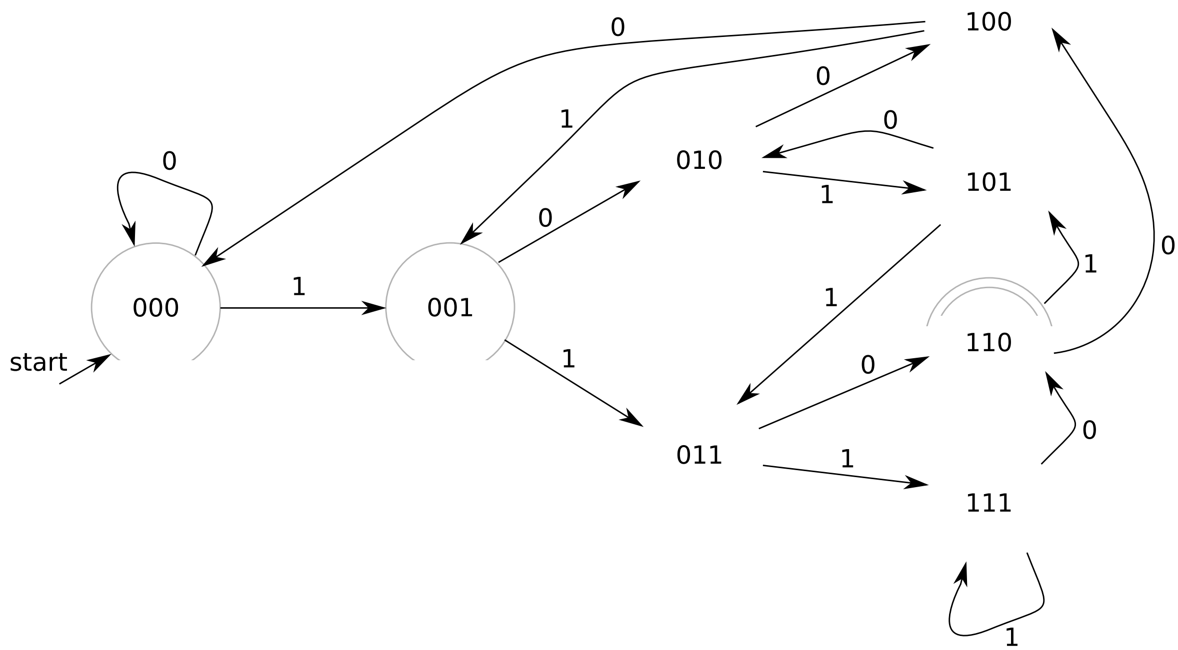
An NFA for that language.



Claim 2. *There exists a language for which there exists a DFA that requires at least 2^n states whereas an NFA exists which requires n states only.*

Example: NFA and DFA for strings of length ≥ 3 whose last-from-3rd symbol must be a 1.





For more general algorithms, of the kind we consider, we do not know, provably, whether non-determinism provides such a benefit.

First an assumption: a non-deterministic choice is exactly as expensive as a corresponding deterministic choice.

Now – suppose we compare ShortDist-Det and ShortDist-NonDet.

Worst-case time: both are $\Theta(n)$ for input-size n.

Worst-case space: ShortDist-~~S~~Det is $\Theta(n)$, ShortDist-NonDet is $\Theta(\lg n)$.

Another example: given a connected undirected graph $G = \langle V, E \rangle$, two distinct vertices $a, b \in V$ and an integer k, does there exist a simple path $a \rightsquigarrow b$ of $\geq k$ edges?

It is unlikely that a polynomial-time deterministic algorithm exists for this problem. A polynomial-time non-deterministic algorithm is the following.

NDLongSimplePath($G = \langle V, E \rangle, a, b, k$)

$c \leftarrow a$, $S \leftarrow \{c\}$
foreach i *from* 1 *to* $|V| - 1$ **do**
    Non-deterministically pick a neighbour, n, of c from $V \setminus S$
    if $n = b$ *and* $i \geq k$ **then return** true
    else
        $c \leftarrow n$
        $S \leftarrow S \cup \{n\}$
return false