



Nondeterminism

Teodor Rus

`rus@cs.uiowa.edu`

The University of Iowa, Department of Computer Science

Introduction

- Nondeterminism is a useful concept that has a great impact on the theory of computation;
- So far in our discussion, every step of an FA computation follows in a unique way from the preceding step: *when the machine is in a given state and reads next input symbol we know what the next state will be*
- We call this a *deterministic computation*.

Note: in a *nondeterministic* computation, choices may exist for the next state at any point!

How does it occur?

- Nondeterminism is a generalization of determinism; hence, every DFA is a nondeterministic finite automaton, NFA.
- However, an NFA may have additional features, as N_1 in Figure 1 shows:

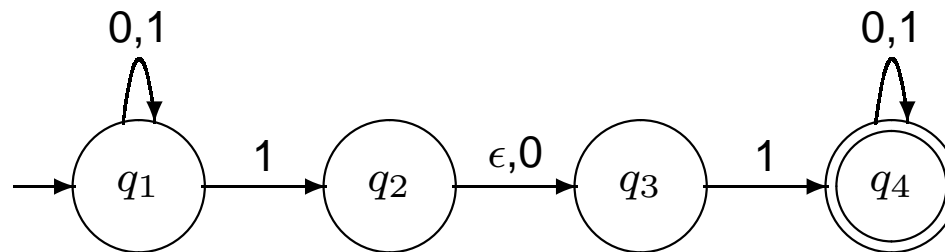


Figure 1: Nondeterministic Finite Automaton N_1

DFA versus NFA

- The difference between a deterministic finite automaton, DFA, and a nondeterministic finite automaton, NFA, is immediately apparent:

Every state of DFA always has exactly one exiting transition arrow for each symbol in the alphabet; NFA violates this rule.

Example: $\delta(q_1, 1) = \{q_1, q_2\}$ in Figure 1.

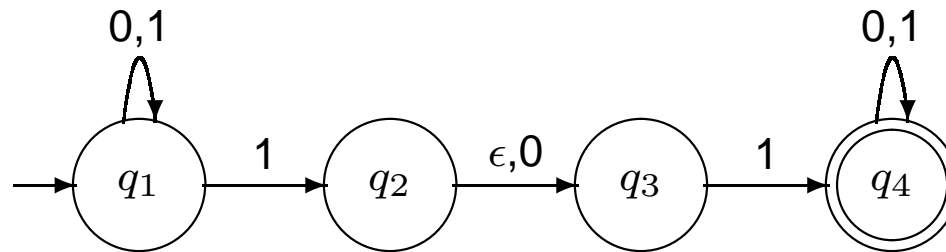
- In an NFA a state may have zero, one, or more exiting arrows for each symbol of the alphabet.

More differences

- In a DFA, labels on the transition arrows are symbols of the alphabet;
- An NFA may have its arrows labeled with symbols of the alphabet or with ϵ ;
- Zero, one, or many arrows may exit from each state with the label ϵ .

How does an NFA compute?

Look at NFA N_1 acting on an input:



- If N_1 is in state q_1 and the input symbol is 1 the machine split into multiple copies of itself and follows all the possibilities in parallel;
- Each copy of the machine takes one of the possible ways to proceed and continues as before, i.e., it may split again;
- If a state with an ϵ symbol on some exiting arrow is encountered, the machine split into multiple copies without reading any input.

Dying computation

If the next input symbol does not appear on any of the arrows exiting the state occupied by a copy of the machine, that copy dies along with the branch of computation associated with it.

Accepting computation

If any one of the copies of the machine is in an accept state at the end of the input, the NFA accepts the input; otherwise it rejects the input.

Behavior on ϵ -transitions

- The behavior of an NFA on an ϵ -transition differs from its behavior on non- ϵ transitions:
it performs a transition without reading the input.

Hence, *when an ϵ -transition is encountered the machine must split.*

- Each copy follows the exiting ϵ labeled arrow, as the one staying to the current state.
- Then the machine proceeds nondeterministically as before.

Observations

- Nondeterminism may be seen as a kind of **parallel computation** where several “processes” run concurrently;
- If at least one of these processes accepts then the entire computation accepts.

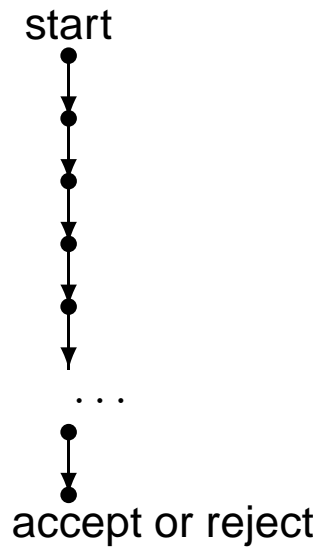
Tree computation

Another way to think of a nondeterministic computation is as **a tree of possibilities**:

- The root of the tree corresponds to the start of the computation;
- Every branching point in the tree corresponds to a point in the computation at which the machine has multiple choices;
- The machine accepts if at least one of the computation branches ends in an accept state as shown in Figure 2.

Comparison of DFA and NFA

DFA computation



NFA computation

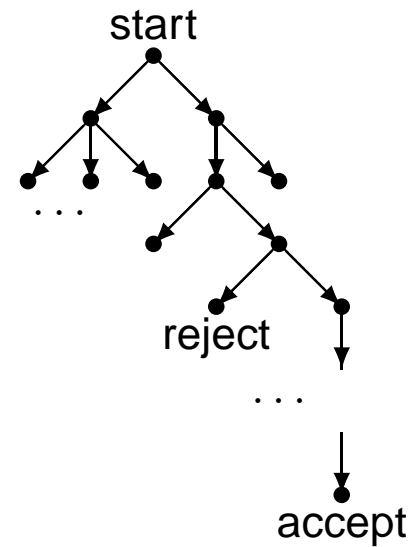
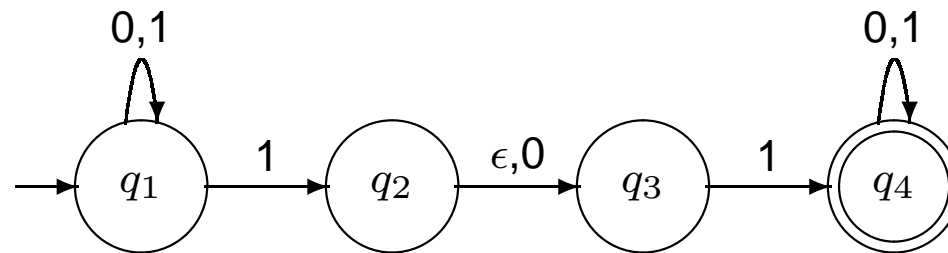


Figure 2: Deterministic and nondeterministic computations

Example run of N_1



On input 010110 start in q_1 and read 0 :

- There is only one place to go on 0 : back to q_1 ;
- Next, read the symbol 1 . There are two choices: either stay in q_1 or move to q_2 . Keep track of possibilities by placing a token on each state where machine can be, i.e., tokens are on q_1 and q_2 ;
- An ϵ arrow exits q_2 , so the machine splits again: keep one token on q_2 and put another on q_3 .

Example run, continuation

- When the third symbol, 0, is read take each token in turn: keep token on q_1 in place, move the token from q_2 to q_3 , and remove the previous token from q_3 . The last token had no 0 arrow to follow and the corresponding process dies.
- When the fourth symbol, 1, is read there are tokens on q_1 and q_3 . Thus, split the token on q_1 keeping it on q_1 and q_2 and move the token on q_3 on q_4 ; since there is an ϵ arrow leaving q_2 split the token on q_2 to q_3 too. Now there are tokens on each state.
- When fifth symbol, 1, is read the tokens on q_1 and q_3 result in tokens on q_1 , q_2 q_3 and q_4 . The previous token on q_2 is removed and the token previously on q_4 stays on q_4 .

Eliminate token duplication

There are two valid tokens on q_4 . Remove one of them because we only need to remember that q_4 is a possible state at this point, not that it is possible to reach this state for multiple reasons.

Example run, continuation

- When the sixth symbol, 0 is read, keep the token on q_1 in place, move the token from q_2 to q_3 , remove previous token from q_3 , and leave the token on q_4 on q_4
- Now the end of the input is detected, since q_4 has a token and it is final, the input is accepted.

Formalizing NFA tree computation

The computation of an NFA N on an input $w = w_1w_2 \dots w_iw_{i+1} \dots w_n$ can be represented by a family of trees T_w^N whose nodes are labeled by the states of the automaton N .

The trees in the family T_w^N are constructed on levels, by the following iterative procedure:

Constructing T_w^N

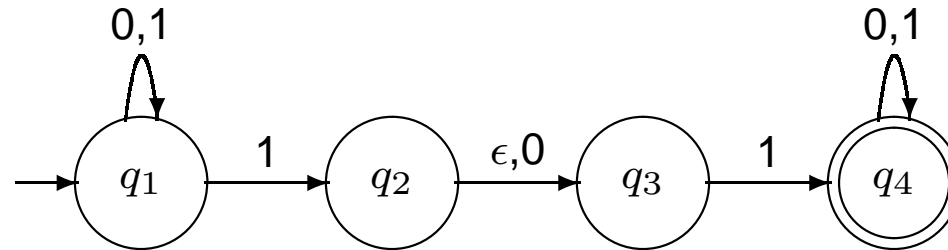
1. The level 0 of the family T_w^N records the roots of the trees in T_w^N . The start state of N is by definition the root of a tree in T_w^N .
2. For $i = 0$ to n construct the nodes on the level $i + 1$ of T_w^N by the rules:
 - (a) **Expansion of the current level:** If there are nodes on the level i that have ϵ -transitions, add the target of these transitions as new nodes on level i . Apply this rule until no more nodes can be added to level i ;
 - (b) **Construction of the next level:** Read next input symbol w_i of w . For each node n on the level i do: if n has transitions labeled by w_i add the target of these transitions as the children of n on level $i + 1$ and mark n ; if n has no transitions on w_i simply mark it.

Computation termination

If there are nodes in level n of T_w^N , $|w| = n$, labeled by accept states then *accept* otherwise *reject*

Example

Consider the NFA N_1 and the string
0 1 0 1 1 0



The trees in $T_w^{N_1}$ are in Figure 3

An NFA tree computation

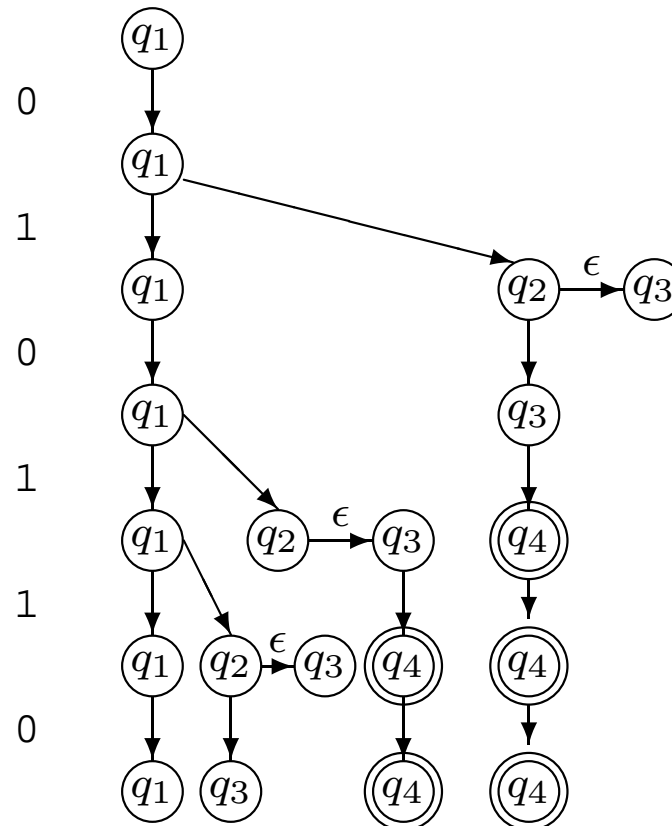


Figure 3: Tree computation performed by N_1

Observations

- N_1 rejects the string 010 , as can be seen on Figure 3.
- By performing other similar experiments we can see that N_1 accepts all strings that contain either 101 or 11 as substrings.

Why nondeterministic automata

- Every NFA can be converted into an equivalent DFA;
- Constructing NFA is sometimes easier than constructing DFA;
- An NFA may be much smaller than a DFA that performs the same task;
- NFA provides good introduction to nondeterminism in more powerful computational models.

Example 1.14

Let A be the language consisting of all strings over $\{0, 1\}$ containing a 1 in the third position from the end, i.e. $A = \{x \in \{0, 1\}^* \mid x = s_1 1 s_2, |s_2| = 2\}$

Example: $000100 \in A$ but $0011 \notin A$

The NFA N_2 in Figure 4 recognizes this language

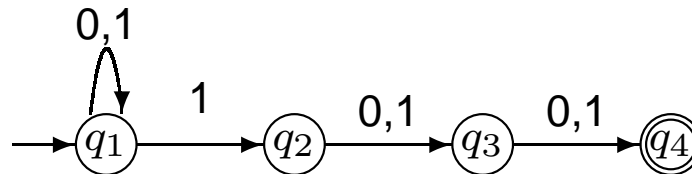


Figure 4: The NFA N_2 recognizing A

Observations

- N_2 stays in q_1 until it “guess” that it is three places from the end;
- When it is three places from the end, if it reads a 1 symbol, it branches to q_2 and then uses q_3 and q_4 to check if its “guess” was correct.

A DFA equivalent to N_2

- As we mentioned earlier, every NFA can be converted into an equivalent DFA.
- Sometimes that DFA may have many more states.
- The smallest DFA, D_2 , equivalent to N_2 , in Figure 5, contains 8 states.

Note: while understanding the functionality of N_2 is easy, the functionality of D_2 is more difficult to understand

The DFA D_2

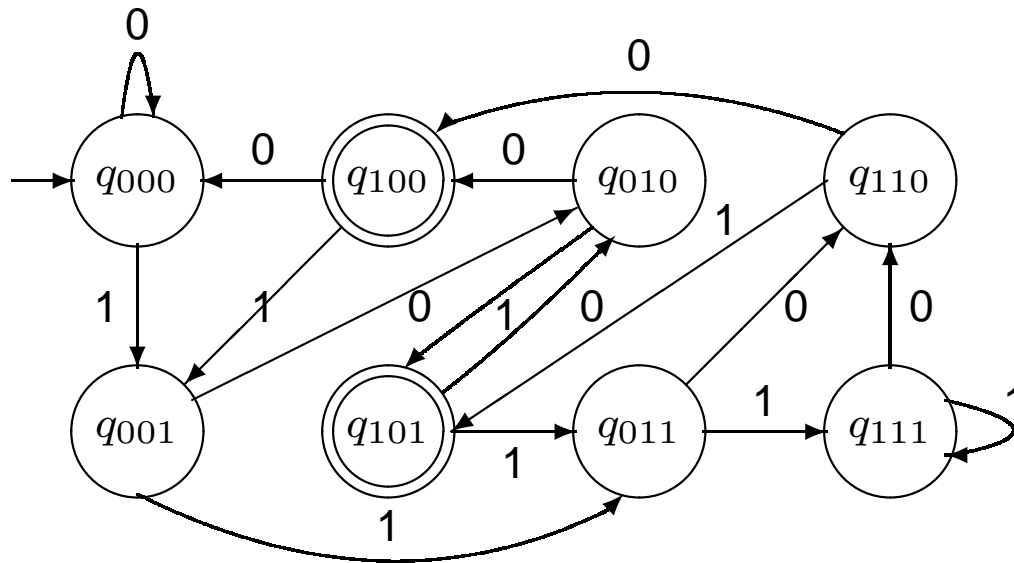


Figure 5: DFA D_2 equivalent to NFA N_2

Problem

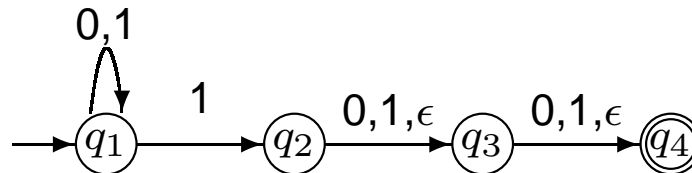


Figure 6: The NFA N_2 recognizing A

Suppose that we added ϵ to the labels on the arrows going from q_2 to q_3 and from q_3 to q_4 in machine N_2 , Figure 4, getting the machine in Figure 6.

1. What language would the new N_2 recognize? Obviously,
 $L(N_3) = \{x \in \{0, 1\}^* \mid x = s_1 1 s_2, |s_2| \leq 2\}$.
2. Modify the DFA D_2 in Figure 5 to recognize the same language.

The use ϵ transitions

Consider the NFA N_3 in Figure 7

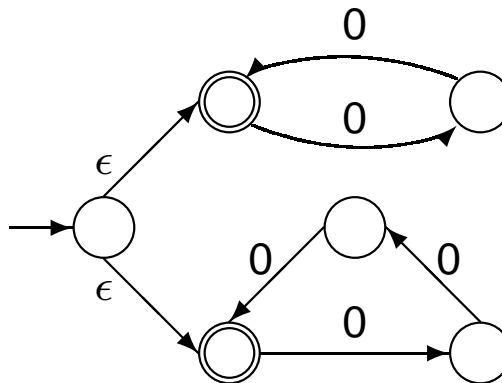


Figure 7: The NFA N_3

Observations

- The NFA N_3 in Figure 7 has a unary input alphabet, $\{0\}$
- N_3 demonstrates the use of ϵ transition for the construction of an NFA that accepts all strings of the form 0^k where k is a multiple of 2 or 3.
- **Example:** ϵ , 00 , 000 , 000000 are accepted by N_3 but 0 , 00000 are rejected

Notation 0^k here means a string of k -zeros, i.e., $0^k = 00 \dots 0$ k -times.

Note

- N_3 illustrates a machine operating by initially guessing whether to test one case or another (multiple of 2 or multiple of 3) by branching into either one loop or another and then checking whether it was correct
- If incorrect the machine rejects, if one or both cases are correct, the machine accepts
- We may try to construct a DFA for this job, but the N_3 is much simpler to understand

A working example

This example will be used to illustrate the procedure for the conversion $NFA \rightarrow DFA$

- The NFA N_4 to be used to illustrate this procedure is in Figure 8.
- N_4 accepts strings ϵ , a , $baba$, baa , etc., but does not accept strings b , bb , $babba$, etc.

The NFA N_4

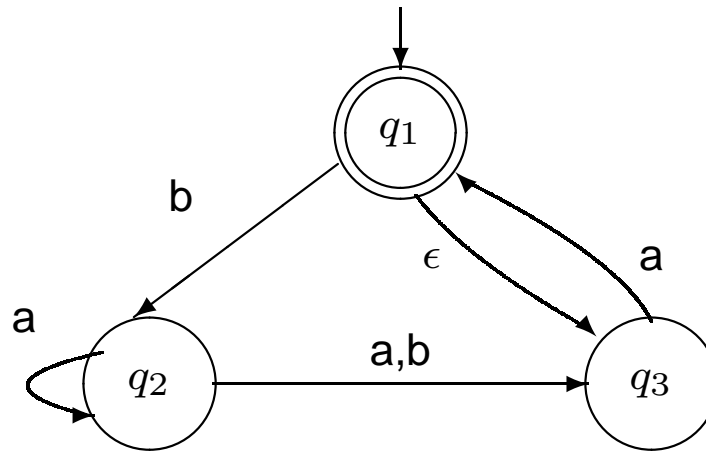


Figure 8: The NFA N_4