ΓΕ 77 COMPUTATIONAL LINGUISTICS

Athanasios N. Karasimos

akarasimos@gmail.com

BA in Linguistics | School of English Language and Literature
National and Kapodistrian University of Athens

Lecture 5 | Wed 28 Mar 2018

MORPHOLOGY: FINITE-STATE AUTOMATA

LECTURE 4 RECAP

Language Modeling with N-Grams

LANGUAGE MODELING WITH N-GRAMS

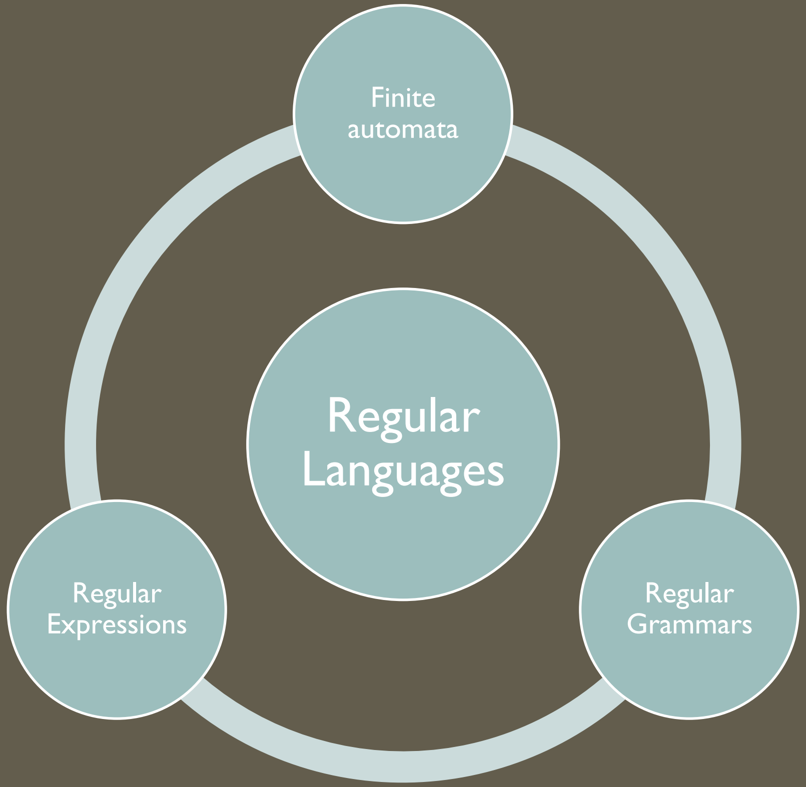
- Language models offer a way to assign a probability to a sentence or other sequence of words, and to predict a word from preceding words.
- N-grams are Markov models that estimate words from a fixed window of previous words. N-gram probabilities can be estimated by counting in a corpus and normalizing (the maximum likelihood estimate).
- N-gram language models are evaluated extrinsically in some task, or intrinsically using perplexity. The perplexity of a test set according to a language model is the geometric mean of the inverse test set probability computed by the model.
- Smoothing algorithms provide a more sophisticated way to estimate the probability of N-grams. Commonly used smoothing algorithms for N-grams rely on lower-order N-gram counts through backoff or interpolation.

FINITE-STATE AUTOMATA

FINITE-STATE AUTOMATA: DEFINITION

- **Finite-state automata** (FSA) are the theoretical foundation of a good deal of the computational work.
- Any regular expression can be implemented as a FSA.
- Symmetrically, any FSA can be described with a regular expression.
- A regular expression is one way of characterizing a particular kind of formal language called a **regular language**.
- Both regular expressions and FSA can be used to describe regular languages.
- A third equivalent method of characterizing the regular languages, the **regular grammar.**

REGULAR LANGUAGES



REGULAR LANGUAGE: DEFINITION

- **Regular language** is a formal language that can be expressed using a regular expression, in the strict sense of the latter notion used in theoretical computer science.
- Alternatively, a regular language can be defined as a language recognized by a finite automaton. The equivalence of regular expressions and finite automata is known as Kleene's theorem. In the Chomsky hierarchy, regular languages are defined to be the languages that are generated by regular grammars.

FSA: DEFINITION

- Devices for recognizing finite state grammars (including regular expressions)
- Two types
  - **Deterministic Finite State Automata** (DFSA)

    Rules are unambiguous
  - **NonDeterministic Finite State Automata** (NDFSA)

    Rules are ambiguous

    Sometimes more than one sequence of rules must be attempted to determine if a string matches the grammar
    - ≫ Backtracking
    - ≫ Parallel Processing
    - ≫ Look Ahead
- Any NDFSA can be mapped into an equivalent (but larger) DFSA

USING AN FSA TO RECOGNIZE COWTALK

- Let’s begin with the “cow language” we discussed previously. Recall that we defined the cow language as any string from the following (infinite) set:
- muu!
- muuu!
- muuuu!
- muuuuu!
- muuuuuu!

USING AN FSA TO RECOGNIZE COWTALK

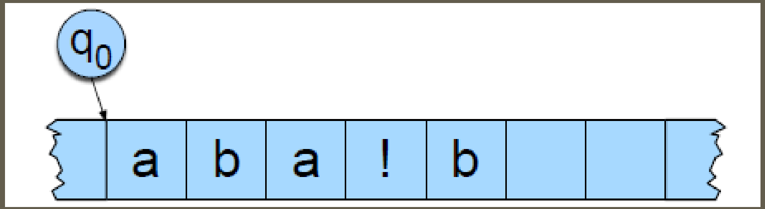
$$\begin{tikzcd}
{} \arrow[r] & q_0 \arrow[r, bend left, "m"] & q_1 \arrow[r, bend left, "u"] & q_2 \arrow[r, bend left, "u"] & q_3 \arrow[loop above, "u"] \arrow[r, bend left, "!"] & \boxed{q_4}
\end{tikzcd}$$

USING AN FSA TO RECOGNIZE COWTALK

- The regular expression for this kind of "cowtalk" is /muu+!/.
- The FSA from the previous slide recognizes a set of strings, in this case the strings characterizing cow talk, in the same way that a regular expression does.
- We represent the automaton as a directed graph: a finite set of vertices (also called nodes), together with a set of directed links between pairs of vertices called arcs.
- We'll represent vertices with circles and arcs with arrows.

USING AN FSA TO RECOGNIZE COWTALK

- State 0 is the **start state**.
- To mark another state as the start state, we can add an incoming arrow to the start state.
- State 4 is the **final state** or **accepting state**, which we represent by the double circle. It also has four **transitions**, which we
- represent by arcs in the graph.
- The FSA can be used for recognizing (we also say **accepting**) strings in the following way.



FSA REJECTION

- If the machine never gets to the final state, either because it runs out of input, or it gets some input that doesn't match an arc, or if it just happens to get stuck in some non-final state, we say the machine **rejects** or fails to accept an input.

FSA	m	u	!
0	1	0	0
1	0	2	0
2	0	3	0
3	0	3	4
4	0	0	0

PARAMETERS OF FSA

- **Σ** is the input alphabet (a finite, non-empty set of symbols).
- **Q** is a finite, non-empty set of states.
- $\mathbf{q_0}$ is an initial state, an element of Q.
- ***δ (q, i)*** is the state-transition function or transition matrix between states. Given a state q ∈ Q and an input symbol *i* ∈ Σ , δ (q, i) returns a new state q´ ∈ Q. δ : Q x Σ -> *Q (is the relation from Q x Σ to Q).*
- F is the set of final states, a (possibly empty) subset of Q (F ≤Q).
- For the cowtalk FSA *Q* = {*q*0,*q*1,*q*2,*q*3,*q*4}, Σ = {*m*,u, !}, *F* = {*q*4}, and d(*q*, *i*) is defined by the transition arrows (or transition table).

DETERMINISTIC FSA

- A **deterministic** algorithm is one that has no choice points; the algorithm always knows what to do for any input.
- The next section will introduce non-deterministic automata that must make decisions about which states to move to.
- D-RECOGNIZE takes as input a tape and an automaton.
- It returns *accept* if the string is pointing to on the tape is accepted by the automaton, and *reject* otherwise. Note that since D-RECOGNIZE assumes it is already pointing at the string to be checked, its task is only a subpart of the general problem that we often use regular expressions for finding a string in a corpus.

DETERMINISTIC FSA

- The algorithm will fail whenever there is no legal transition for a given combination of state and input.
  - The input *ume* will fail to be recognized since there is no legal transition out of state *q*0 on the input u.
  - Even if the automaton had allowed an initial *a* it would have certainly failed on e since e isn't even in the cowtalk alphabet!
- We can think of these "empty" elements in the table as if they all pointed at one "empty" state, which we might call the **fail state** or **sink state**. In a sense then, we could view any machine with empty transitions *as if* we had augmented it with a fail state, and drawn in all the extra arcs, so we always had somewhere to go from any state on any possible input.

FORMAL LANGUAGES

FORMAL LANGUAGE

- The FSA starts at state *q*0, and crosses arcs to new states, printing out the symbols that label each arc it follows.
- When the automaton gets to the final state it stops.
- Notice that at state 3, the automaton has to chose between printing out a **!** and going to state 4, or printing out an **u** and returning to state 3. Let's say for now that we don't care how the machine makes this decision; maybe it flips a coin.
- For now, we don't care which exact string of cowtalk we generate, as long as it's a string captured by the regular expression for cowtalk above.

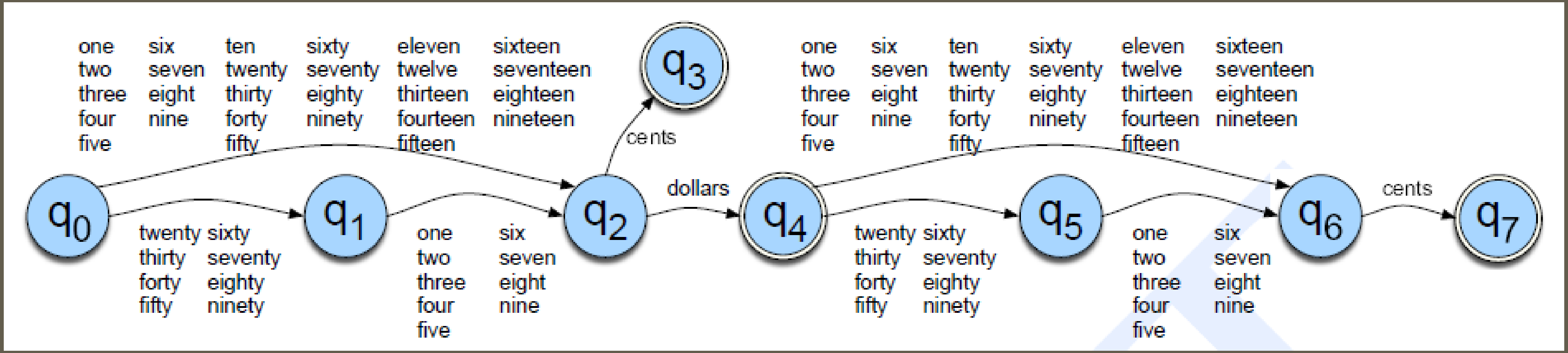
FORMAL LANGUAGE

- **Formal Language:** A model which can both generate and recognize all and only the strings of a formal language acts as a *definition* of the formal language.
- A **formal language** is a set of strings, each string composed of symbols from a finite symbol-set called an **alphabet** (the same alphabet used above for defining an automaton!). The alphabet for the cow language is the set S = {m, u, !}. Given a model *m* (such as a particular FSA), we can use *L*(*m*) to mean "the formal language characterized by *m*".
- *L(m)* = {*muu!, muuu!, muuuu!, muuuuu!, muuuuu!, muuuuuu!....*}

FORMAL LANGUAGE

- The usefulness of an automaton for defining a language is that it can express an infinite set (such as this one above) in a closed form. Formal languages are not the same as **natural languages**, which are the kind of languages that real people speak.
- In fact, a formal language may bear no resemblance at all to a real language (e.g., a formal language can be used to model the different states of a soda machine).
- But we often use a formal language to model part of a natural language, such as parts of the phonology, morphology, or syntax.
- The term **generative grammar** is sometimes used in linguistics to mean a grammar of a formal language; the origin of the term is this use of an automaton to define a language by generating all possible strings.

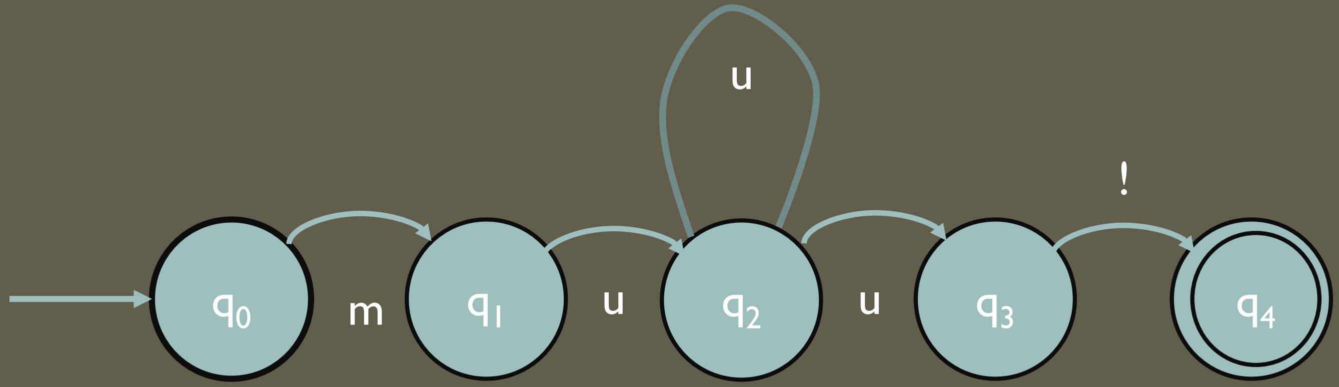
FORMAL LANGUAGE


q0
q1
q2
q3
q4
q5
q6
q7
one two three four five
six seven eight nine
ten twenty thirty forty fifty
sixty seventy eighty ninety
eleven twelve thirteen fourteen fifteen
sixteen seventeen eighteen nineteen
twenty thirty forty fifty
sixty seventy eighty ninety
one two three four five
six seven eight nine
cents
dollars
one two three four five
six seven eight nine
ten twenty thirty forty fifty
sixty seventy eighty ninety
eleven twelve thirteen fourteen fifteen
sixteen seventeen eighteen nineteen
twenty thirty forty fifty
sixty seventy eighty ninety
one two three four five
six seven eight nine
cents

NON-DETERMINISTIC FSA

NON-DETERMINISTIC FSA

- Let's extend our discussion now to another class of FSAs: **non-deterministic FSAs** (or **NFSAs**).


$$\begin{tikzcd} \arrow[r] & q_0 \arrow[r, bend left, "m"] & q_1 \arrow[r, bend left, "u"] & q_2 \arrow[loop above, "u"] \arrow[r, bend left, "u"] & q_3 \arrow[r, bend left, "!"] & q_4 \end{tikzcd}$$

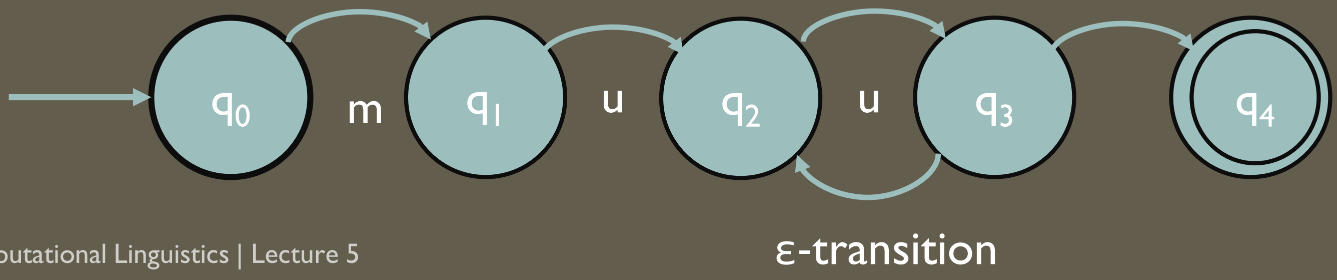
FSA	m	u	!
0	1	0	0
1	0	2	0
2	0	2,3	0
3	0	0	4
4	0	0	0

NON-DETERMINISTIC FSA

- The only difference between the NFSA and the FSA is the first has the self-loop is on state 2 instead of state 3. Consider using this network as an automaton for recognizing cowtalk. When we get to state 2, if we see an **a** we don't know whether to remain in state 2 or go on to state 3.
- Automata with decision points like this are called **non-deterministic FSA**s (or **NFSA**s). Recall by contrast NFSA that the regular FSA specified a **deterministic** automaton, i.e., one whose behavior during recognition is fully *determined* by the state it is in and the symbol it is looking at.
- A DFSA deterministic automaton can be referred to as a **DFSA**.

NON-DETERMINISTIC FSA

- There is another common type of non-determinism, caused by arcs that have no symbols on them (called ε-**transitions**).
- The following NFSA defines the exact same language as the last one, or our first one, but it does it with an ε-transition.
- We interpret this new arc as follows: If we are in state 3, we are allowed to move to state 2 *without* looking at the input, or advancing our input pointer. So this introduces another kind of non-determinism — we might not know whether to follow the ε-transition or the **!** arc.



NON-DETERMINISTIC FSA

- TASK I:
  - Give a simple regular expression (no more than 4 symbols).
- TASK II:
  - Give the five parameters of your NFSA.
- TASK III:
  - Create a NFSA for your expression. It can contain loops and ε-transition.

NON-DETERMINISTIC FSA

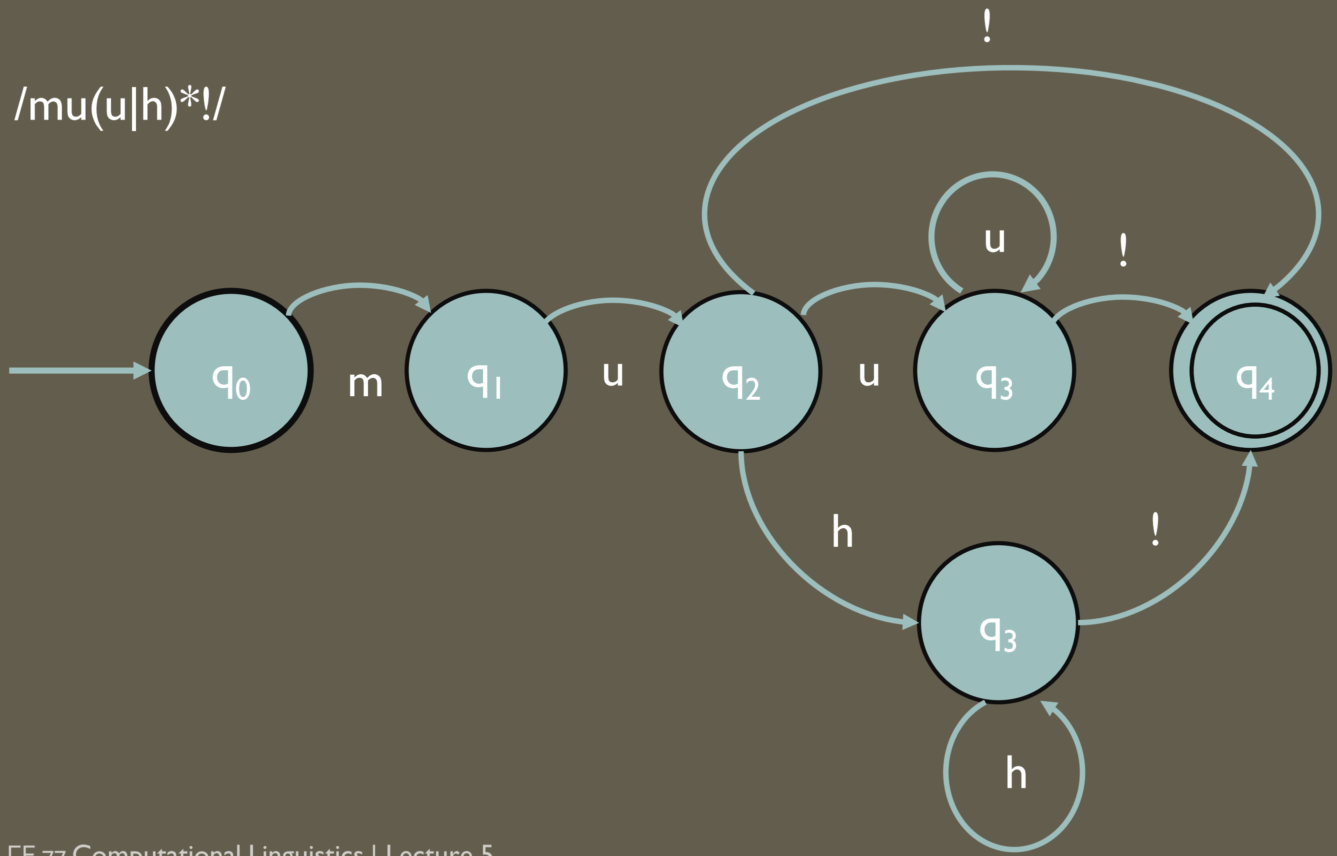
- **Using an NFSA to Accept Strings**
  - if we use a non-deterministic machine to recognize it, we might follow the wrong arc and reject it when we should have accepted it.
- This problem of choice in non-deterministic models will come up again and again as we build computational models, particularly for parsing.
- There are three standard solutions to the problem of non-determinism:

SOLUTIONS FOR NFSA

- **Backup:** Whenever we come to a choice point, we could BACKUP put a *marker* to mark where we were in the input, and what state the automaton was in. Then if it turns out that we took the wrong choice, we could back up and try another path.
- **Look-ahead:** We could look ahead in the input to help us decide which path to take.
- **Parallelism:** Whenever we come to a choice point, we could look at every alternative path in parallel.

A PARALLEL FSA

- /mu(u|h)*!/

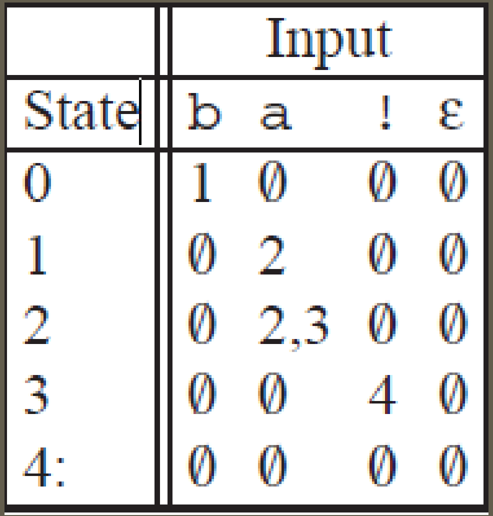


THE BACK-UP APPROACH

- The **backup approach** suggests that we should blithely make choices that might lead to dead-ends, knowing that we can always return to unexplored alternative choices.
- There are two keys to this approach: we need to remember all the alternatives for each choice point, and we need to store sufficient information about each alternative so that we can return to it when necessary.
- When a backup algorithm reaches a point in its processing where no progress can be made (because it runs out of input, or has no legal transitions), it returns to a previous choice point, selects one of the unexplored alternatives, and continues from there. Applying this notion to our non-deterministic recognizer, we need only remember two things for each choice point: the state, or node, of the machine that we can go to and the corresponding position on the string.

THE BACK-UP APPROACH

- We will call the combination of the node and position the **search-state** of the recognition algorithm. To avoid confusion, we will refer to the state of the automaton (as opposed to the state of the search) as a **node** or a **machine-state**.
- If a node has an e-transition, we list the destination node in the e-column for that node's row.
- TASK IV: Create a table for your NFSA.



	Input			
State	b	a	!	ε
0	1	∅	∅	∅
1	∅	2	∅	∅
2	∅	2,3	∅	∅
3	∅	∅	4	∅
4:	∅	∅	∅	∅

FROM NFSA TO DFSA

- It may seem that allowing NFSAs to have non-deterministic features like e-transitions would make them more powerful than DFSAs.
- There is a simple algorithm for converting an NFSA to an equivalent DFSA, although the number of states in this equivalent deterministic automaton may be much larger.
- Recall that the difference between NFSAs and DFSAs is that in an NFSA a state q_i may have more than one possible next state given an input *i* (for example q_a and q_b).
- The algorithm for converting a NFSA to a DFSA is like this parallel algorithm; we build an automaton that has a deterministic path for every path our parallel recognizer might have followed in the search space.
- We imagine following both paths simultaneously, and group together into an equivalence class all the states we reach on the same input symbol (i.e., q_a and q_b).

SUMMARY

SUMMARY

- Any regular expression can be realized as a **finite state automaton** (**FSA**).
- An automaton implicitly defines a **formal language** as the set of strings the automaton **accepts**.
- An automaton can use any set of symbols for its vocabulary, including letters, words, or even graphic images.
- The behavior of a **deterministic** automaton (**DFSA**) is fully determined by the state it is in.
- A **non-deterministic** automaton (**NFSA**) sometimes has to make a choice between multiple paths to take given the same current state and next input.
- Any **NFSA** can be converted to a **DFSA.**

READING

- Jurafsky D. & J. Martin (2008). SPEECH and LANGUAGE PROCESSING
  An introduction to Natural Language Processing, Computational Linguistics and Speech Recognition (2nd Edition). CHAPTER 2.