

Lecture 3: ϵ -NFAs and Regular Expressions

David Dill

Department of Computer Science

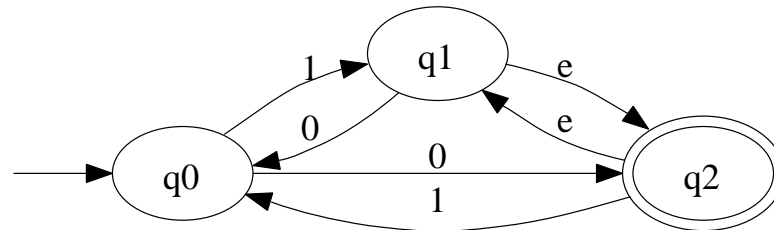
Outline

- ϵ -NFAs
- More operations on languages
- Regular expression syntax and semantics

ϵ -NFA

Idea; Allow state changes that don't consume input symbols ("silent moves").

Choice of whether to take ϵ -transition is non-deterministic.



("e" in the drawing is ϵ)

The input "001" is accepted. The path followed is $q_0q_2q_1q_0q_1q_2$.

Definition of ϵ -NFA

The only difference in the mathematical definition is in δ :

$$\delta : Q \times (\Sigma \cup \{\epsilon\}) \rightarrow 2^Q$$

ϵ -closure

Preliminary: the ϵ -closure of state q , $ECLOSE(q)$, is the set of all states reachable from q by silent moves.

$$ECLOSE(q_0) = \{q_0\}$$

$$ECLOSE(q_1) = \{q_1, q_2\}$$

$$ECLOSE(q_2) = \{q_1, q_2\}$$

Definition of *ECLOSE*

Def: S is the least set satisfying property P if whenever $P(S')$ holds, $S \subseteq S'$.

$ECLOSE(q)$ is the least set satisfying:

- $q \in ECLOSE(q)$
- $\delta(s, \epsilon) \subseteq ECLOSE(q)$ for all $s \in ECLOSE(q)$

Also, $ECLOSE(S) = \bigcup \{ECLOSE(q) \mid q \in S\}$

(“Extend to sets” – this is overloading)

Acceptance by an ϵ -NFA

We can define $\hat{\delta}$ for an ϵ -NFA taking *ECL**OSE* into account. Here is a recursive definition on the structure of strings:

Base: $w = \epsilon$: $\hat{\delta}(q, \epsilon) = ECL*OSE*(q)$

Induction: $w = xa$: $\hat{\delta}(q, xa) = ECL*OSE*($\bigcup_{s \in \hat{\delta}(q, x)} \delta(s, a)$)$

(Same as $\hat{\delta}$ for NFA, with *ECL**OSE* wrapped around state sets.)

$$L(E) = \{x \mid \hat{\delta}(q_0, x) \cap F \neq \emptyset\}$$

Example:

$$\hat{\delta}(q_0, 001) = \{q_1, q_2\} - \text{accepted because } q_2 \in F.$$

Converting an ϵ -NFA to an DFA

Theorem: For every ϵ -NFA E , there exists an DFA D such that $L(D) = L(E)$.

The proof uses a modified subset construction, where every subset is ϵ -closed.

Given $E = (Q, \Sigma, q_0, \delta_E, F_E)$, we can define

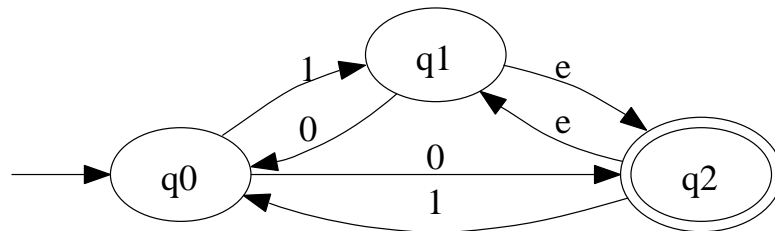
$D = (2^Q, \Sigma, ECLOSE(q_0), \delta_D, F_D)$ as follows:

$$F_D = \{S \subseteq Q \mid S \cap F \neq \emptyset\}$$

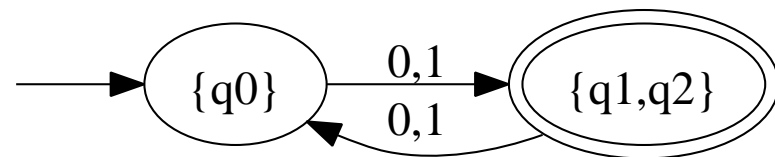
$$\delta_D(S, a) = ECLOSE(\bigcup_{s \in S} \delta_E(s, a))$$

Note: δ_D has no ϵ transitions!

ϵ -NFA-to-DFA example



converts to the DFA:



ϵ -NFA to DFA Proof [not in lecture]

Lemma: If D is the DFA obtained from the ϵ -NFA N by the ϵ -elimination, then $\hat{\delta}_D(ECLOSE(q_0), x) = \hat{\delta}_E(q_0, x)$ for all $x \in \Sigma^*$.

Proof: By induction on strings:

Basis:

$$\begin{aligned} \hat{\delta}_D(ECLOSE(q_0), \epsilon) &= ECLOSE(q_0) && \text{Def of } \hat{\delta} \text{ for DFA} \\ &= \hat{\delta}_E(q_0, \epsilon) && \text{def of } \hat{\delta} \text{ for } \epsilon\text{-NFA} \end{aligned}$$

Induction:

$$\begin{aligned} \hat{\delta}_D(ECLOSE(q_0), xa) &= \delta_D(\hat{\delta}_D(ECLOSE(q_0), x), a) && \text{Def of } \hat{\delta} \text{ for DFA} \\ &= \delta_D(\hat{\delta}_E(q_0, x), a) && \text{Ind. Hyp.} \\ &= ECLOSE(\bigcup_{s \in \hat{\delta}_E(q_0, x)} \delta_E(s, a)) && \text{Def. of } \delta_D \\ &= \hat{\delta}_E(q_0, xa) && \text{Def. of } \hat{\delta}_E \end{aligned}$$

Theorem: $L(E) = L(D)$.

The proof is as in equivalence of NFAs and DFAs

Summary: DFAs, NFAs, and ϵ -NFAs all have *equal expressive power*.

I.e. If one kind accepts a language, all the others do, too.

Regular Expressions

Regular expressions are an *algebraic notation* for regular languages.

Extensively used

- String pattern matching utilities (e.g., “grep” in unix)
- Computer language definitions (“lexical structure”)
- Compiler generation tools

Regular expressions are a completely different “formalism” from finite automata — but they have *exactly the same expressive power*.

Tools based on regular expressions usually translate them to finite automata, which are more suitable than regular expressions for many applications.

Concatenation of Languages

If L_1 and L_2 are languages, we can define the concatenation L_1L_2 to be $\{xy \mid x \in L_1 \wedge y \in L_2\}$.

Examples:

What is $\{ab, ba\}\{cd, dc\}$? $\{abcd, abdc, bacd, badc\}$

What is $\emptyset\{ab, bc\}$? \emptyset

Exponentiation of Languages

L^i is the language L concatenated with itself i times.

Recursive definition:

Base: $L^0 = \{\epsilon\}$

Induction: $L^{i+1} = LL^i$.

Example: $\{ab, ba\}^2 =? \{abab, abba, baab, baba\}$

Example: $\emptyset^0 =? \{\epsilon\}$

Example: $\emptyset^2 = ? \emptyset$

Kleene Closure

$$L^* = \bigcup_{i=0}^{\infty} L^i = L^0 \cup L^1 \cup \dots$$

This operator can build infinite sets from finite sets.

Examples:

$$\{ab, ba\}^* = ? \{\epsilon, ab, ba, abab, abba, \dots\}$$

$$\emptyset^* = ? \{\epsilon\}$$

$$\{\epsilon\}^* = ? \{\epsilon\}$$

Regular Expression Definition

Regular expressions are defined relative to some alphabet Σ .

This is a recursive definition of the structure of regular expressions.

The structure of regular expressions is the basis for other recursive definitions and induction proofs. Every recursive definition and proof has to handle these six cases:

- \emptyset is a regular expression.
- ϵ is a regular expression.
- a is a regular expression, if $a \in \Sigma$.
- $(E_1 + E_2)$ is a regular expression if E_1 and E_2 are.
- $(E_1 E_2)$ is a regular expression if E_1 and E_2 are.
- (E^*) is a regular expression if E_1 and E_2 are.

Notation

We will often omit parentheses. The default assumption is that $*$ is “stickiest,” $E_1 E_2$ is next stickiest, and $E_1 + E_2$ is the least sticky, so $a + bc^* = a + (b(c^*))$.

This notation should be thought of as a way to write down expression trees, which is what we use for definitions and proofs about regular expressions.

In other words, it’s an abstract data structure with six different constructors.

Recursive definitions and induction on the structure of regular expressions will have six cases, corresponding to the six cases in the definition above.

The first three definitions can be considered *base cases* while the last three are inductive.

Language of a Regular Expression

If E is regular expression, the language of E ($L(E)$) is defined recursively on the structure of regular expressions.

- $L(\emptyset) = \emptyset$
- $L(\epsilon) = \{\epsilon\}$
- $L(\mathbf{a}) = \{a\}$
- $L(E_1 + E_2) = L(E_1) \cup L(E_2)$
- $L(E_1 E_2) = L(E_1)L(E_2)$
- $L(E^*) = (L(E))^*$

Regular Expression Examples

What is 0^*1^* ?

All strings ending in “11”?

What is $(a^*b^*)^*$?

With $\Sigma = \{0, 1\}$ How do we write “at least one 0”?

With $\Sigma = \{0, 1\}$ How do we write “at least one 0 and at least one 1”?

$L^+ = L^1 \cup L^2 \cup \dots$ How do we write it using existing operators?

$L^?$ means “an optional L ”. How do we write it?