CA320 - Computability & Complexity

Regular Languages

Dr. David Sinclair

Regular Languages

The set of *regular languages* $\mathcal{R}$ over an alphabet Σ is defined as:

- The language $\emptyset$ is an element of $\mathcal{R}$.
- $\forall a \in \Sigma$, the language $\{a\}$ is in $\mathcal{R}$.
- $\forall L_1, L_2 \in \mathcal{R}$, then the following languages are in $\mathcal{R}$.
  - $L_1 \cup L_2$
  - $L_1 L_2$
  - L_1^*

Regular Expressions

Every *regular language* can be represented by a *regular expressions*, and every *regular expression* represents a *regular language*.

- $\emptyset$ and ϵ are regular expressions.
- $\forall a \in \Sigma, a$ is a regular expression.
- If R_1, R_2 are regular expressions, them the following are regular expressions in order of precedence with the first having the highest precedence.

(R_1)	parentheses
R_1^*	closure
$R_1 R_2$	concatenation
$R_1 + R_2$	alternation

Given regular expression R, $L(R)$ stands for the language represented by R.

Regular Expressions (2)

Some Examples:

Regular Language	Corresponding Regular Expression
$\emptyset$	$\emptyset$
$\{a\}$	a
$\{a, b\}^*$	$(a+b)^*$
$\{aab\}^* a, ab$	$(aab)^*(a+ab)$
$(\{aa, bb\} \cup \{ab, ba\}\{aa, bb\}^*\{ab, ba\})^*$	$(aa+bb+(ab+ba)(aa+bb)^*(ab+ba))^*$

Some properties of regular expressions.

- $R^* = R^*R^* = (R^*)^* = R + R^*$
- $R_1(R_2R_1)^* = (R_1R_2)^*R_1$
- $(R_1^*R_2)^* = \epsilon + (R_1 + R_2)^*R_2$
- $(R_1R_2^*)^* = \epsilon + R_1(R_1 + R_2)^*$

Some More Examples

Let $\Sigma = \{a, b\}$.

Regular Expression	Language	Comments
$(a+b)(a+b)$	$\{aa, ab, ba, bb\}$	$(a+b)(a+b) = aa+ab+ba+bb$
$(a+b)^*$	all strings of a's and b's *including* ϵ	$(a+b)^* = (a^*b^*)^*$
$a+a^*b$	$\{a, b, ab, aab, aaab, \ldots\}$	note order of precedence
$b^*ab^*(ab^*ab^*)^*$	string with an odd number of a's	Other valid expressions are $b^*a(b+ab^*a)^*$ or $(b+ab^*a)^*ab^*$

Deterministic Finite Automata

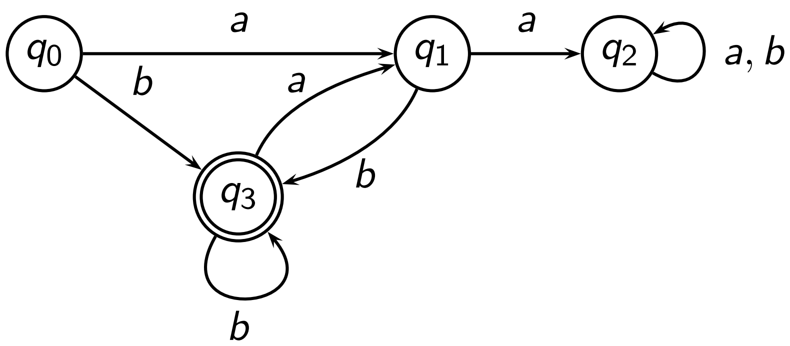
A *deterministic finite automata* (DFA) is a 5-tuple $(Q, \Sigma, q_0, A, \delta)$ where

- Q is a **finite** set of *states*;
- Σ is a **finite** *input alphabet*;
- $q_0 \in Q$ is the *initial state*;
- $A \subseteq Q$ is the set of *accepting states*;
- $\delta : Q \times \Sigma \to Q$ is the *transition function*.

For any element $q \in Q$ and any symbol $\sigma \in \Sigma$, $\delta(q, \sigma)$ is the state the DFA moves to if it receives input σ while in state q.

Deterministic Finite Automata (2)

Here is the DFA that accepts strings ending in b but does not contain aa.

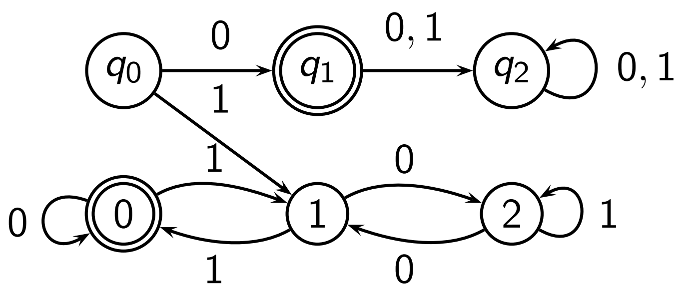


What is the corresponding regular expression?

Regular Languages

Deterministic Finite Automata (3)

Here is a DFA that accepts binary number that are divisible by 3. Adding a 0 onto a binary number x is doubling it. Adding a 1 onto a binary number x is doubling it and adding 1. The same happens to remainder, though if the remainder is greater than 3 we need to do an additional *mod* 3 operations.
The states labelled 0,1 and 2 correspond to states in which the x *mod* 3 is equal to 0,1 and 2 respectively.



What sort of binary strings will it not accept?

Deterministic Finite Automata (4)

The *extended transition function* $\delta^* : Q \times \Sigma^* \rightarrow Q$ is defined as:

- for every $q \in Q, \delta^*(q, \epsilon) = q$
- for every $q \in Q$, every $y \in \Sigma^*$ and every $\sigma \in \Sigma$
$$\delta^*(q, y\sigma) = \delta(\delta^*(q, y), \sigma)$$

Let $M = (Q, \Sigma, q_0, A, \delta)$ be a finite automata and let $x \in \Sigma^*$. The string x is *accepted by M* if

$$\delta^*(q_0, x) \in A$$

and is *rejected by M* otherwise.
The *language* accepted by M is the set

$$L(M) = \{x \in \Sigma | x \text{ is accepted by } M\}$$

If L is a language over Σ, L is accepted by M if and only if $L = L(M)$.

Nondeterministic Finite Automata

A *nondeterministic finite automaton* (NFA) is a 5-tuple $(Q, \Sigma, q_0, A, \delta)$ where

- Q is a **finite** set of *states*;
- Σ is a **finite** *input alphabet*;
- $q_0 \in Q$ is the *initial state*;
- $A \subseteq Q$ is the set of *accepting states*;
- $\delta : Q \times (\Sigma \cup \{\Lambda\}) \rightarrow 2^Q$ is the *transition function*.

For any element $q \in Q$ and any symbol $\sigma \in \Sigma \cup \{\Lambda\}$, ($\Lambda$ is the null symbol), $\delta(q, \sigma)$ is the *set of states* the NFA moves to if it receives input σ while in state q.

Nondeterministic Finite Automata (2)

Let $M = (Q, \Sigma, q_0, A, \delta)$ be an NFA and $S \subseteq Q$ be a set of states. The Λ-*closure of* S is the set $\Lambda(S)$ and is defined as

- $S \subseteq \Lambda(S)$
- $\forall q \in \Lambda(S), \delta(q, \Lambda) \subseteq \Lambda(S)$

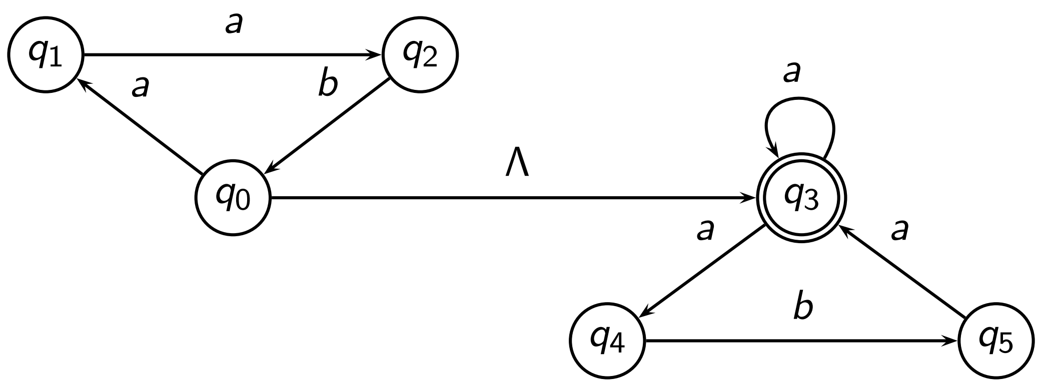
The *extended transition function* for an NFA, $\delta^* : Q \times \Sigma^* \rightarrow 2^Q$ is defined as

- $\forall q \in Q, \delta^*(q, \Lambda) = \Lambda(\{q\})$
- $\forall q \in Q, y \in \Sigma^*, \sigma \in \Sigma,$
$$\delta^*(q, y\sigma) = \Lambda(\bigcup\{\delta(p, \sigma) | p \in \delta^*(q, y)\})$$

A string $x \in \Sigma^*$ is *accepted by* M if $\delta^*(q_0, x) \cap A \neq \emptyset$.
The language $L(M)$ accepted by M is the set of all strings accepted by M.

Nondeterministic Finite Automata (3)

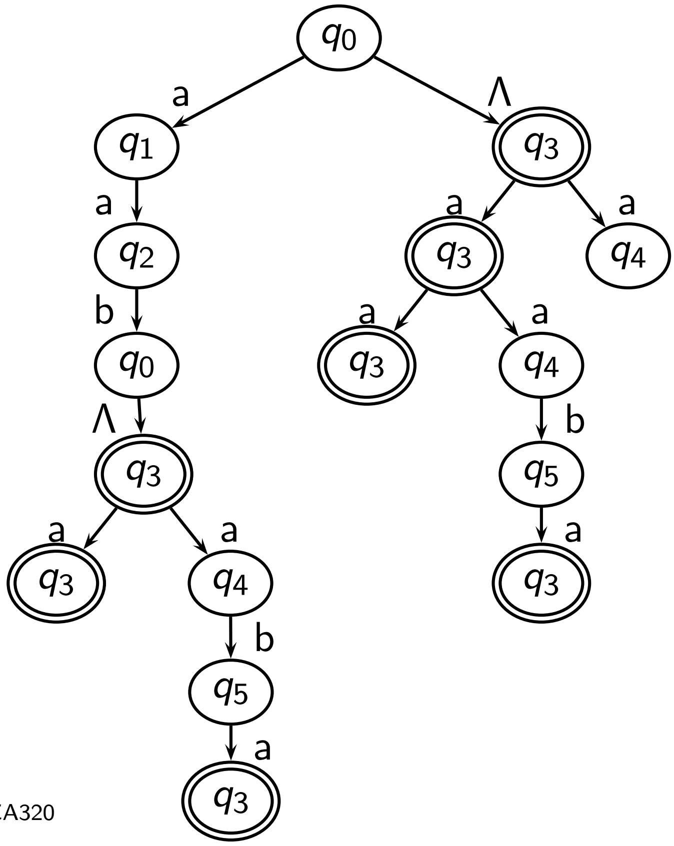
The concept of acceptance for an NFA is quite different than that corresponding concept for a DFA.
Consider the language $\{aab\}^*\{a, aba\}^*$. An NFA that accepts this language is:



Consider how this NFA would process the string *aababa*.

Nondeterministic Finite Automata (4)

We can represent this by a *computation tree*.



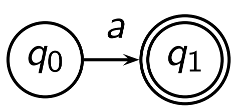
input: $\epsilon aababa$

Regular Expressions to NFA

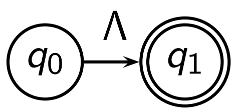
It is straightforward to convert a regular expression into a nondeterministic finite automaton.

Regular Expression NFA

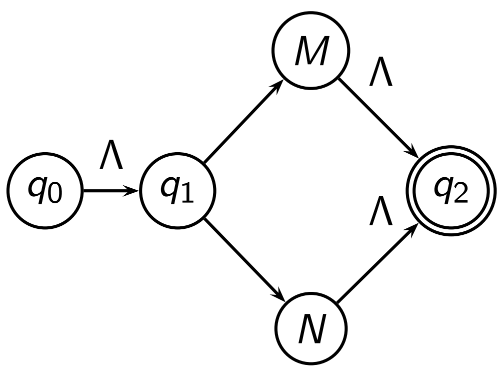
a



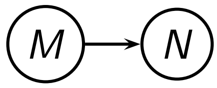
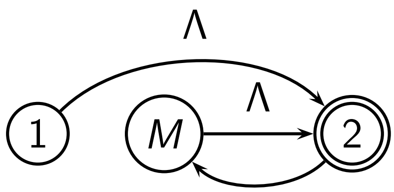
ϵ



$M + N$



Regular Expressions to NFA [2]

Regular Expression	NFA
MN	
M^*	

Converting an NFA to a DFA

While NFAs are nice theoretical devices we do not know how to build such a device. DFAs, on the other hand, are devices we can build.

Fortunately we can convert an NFA into a DFA in two steps:

- First remove the Λ transitions.
- Secondly redefine the states so that there is only one possible next state from the current state given any input.

Theorem

$\forall L \subseteq \Sigma^*$ accepted by an NFA $M = (Q, \Sigma, q_0, A, \delta)$ there is an NFA M_1 with no Λ-transitions that also accepts L.

Converting an NFA to a DFA (2)

Proof.

$$\text{Let } M_1 = (Q, \Sigma, q_0, A_1, \delta_1)$$

where

$$A_1 = \begin{cases} A \cup \{q_0\} & \text{if } \Lambda \in L \\ A & \text{otherwise} \end{cases}$$

$$\delta_1(q, \sigma) = \delta^*(q, \sigma)$$

We need to prove that $\delta_1^*(q, x) = \delta^*(q, x)$ for $|x| \geq 1$ and this is done by structural induction on x.
If $x = a \in \Sigma$ then by definition $\delta_1(q, x) = \delta^*(q, x)$ and because M_1 has no Λ-transitions $\delta_1(q, x) = \delta_1^*(q, x)$, hence $\delta_1^*(q, x) = \delta^*(q, x)$.

Converting an NFA to a DFA (3)

Proof contd.

Let $x = y\sigma, y \in \Sigma^*, \sigma \in \Sigma$

$$\begin{aligned} \delta_1^*(q, y\sigma) &= \bigcup\{\delta_1(p, \sigma) | p \in \delta_1^*(q, y)\} \\ &= \bigcup\{\delta_1(p, \sigma) | p \in \delta^*(q, y)\} && \text{by induction hypothesis} \\ &= \bigcup\{\delta^*(p, \sigma) | p \in \delta^*(q, y)\} && \text{by definition of } \delta_1 \\ &= \delta^*(q, y\sigma) \end{aligned}$$

Hence $L(M_1) = L(M) = L$.

□

Theorem

$\forall L \in \Sigma^*$ accepted by an NFA $M = (Q, \Sigma, q_0, A, \delta)$ there is a DFA $M_1 = (Q_1, \Sigma, q_1, A_1, \delta_1)$ that also accepts L.

Converting an NFA to a DFA (4)

Proof.

The previous theorem means we can remove all Λ-transitions. We can remove the last source of nondeterminism, the multiple next states, by redefining the states as the set of states that can be reached given a specific input symbol.

$$Q_1 = 2^Q$$
$$q_1 = \{q_0\}$$
$$A_1 = \{q \in Q_1 | q \cap A \neq \emptyset\}$$
$$\delta_1(q, \sigma) = \bigcup\{\delta(p, \sigma) | p \in q\}$$

To prove M and M_1 accept the same languages we need to show $\delta_1^*(q_1, x) = \delta^*(q_0, x), \forall x \in \Sigma^*$. This is done by structural induction on x.

Regular Languages

Converting an NFA to a DFA (5)

Proof contd.

If $x = \epsilon$, then

$$\begin{array}{rcll} \delta_1^*(q_1, x) & = & \delta_1^*(q_1, \Lambda) & \\ & = & q_1 & \text{by definition of } \delta_1^* \\ & = & \{q_0\} & \text{by definition of } q_1 \\ & = & \delta^*(q_0, \Lambda) & \text{by definition of } \delta^* \\ & = & \delta^*(q_0, x) & \end{array}$$

If $x = y\sigma$, then

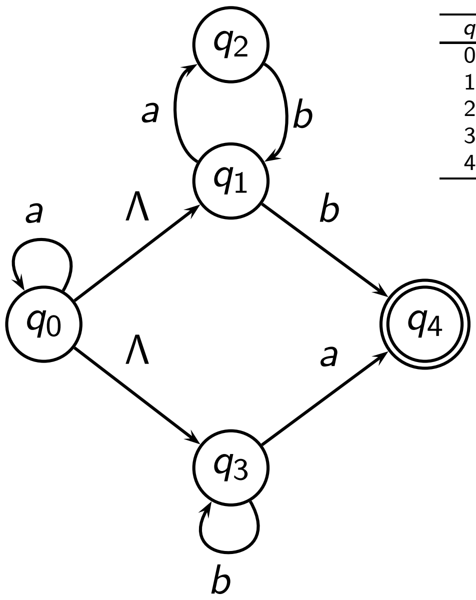
$$\begin{array}{rcll} \delta_1^*(q_1, y\sigma) & = & \delta_1(\delta_1^*(q_1, y), \sigma) & \text{by the definition of } \delta_1^* \\ & = & \delta_1(\delta^*(q_0, y), \sigma) & \text{by the induction hypothesis} \\ & = & \bigcup\{\delta(p, \sigma) | p \in \delta^*(q_0, y)\} & \text{by definition of } \delta_1 \\ & = & \delta^*(q_0, y\sigma) & \text{by the definition of } \delta^* \end{array}$$

Hence $L(M_1) = L(M) = L$.

□

Example NFA to DFA

Consider the following NFA.

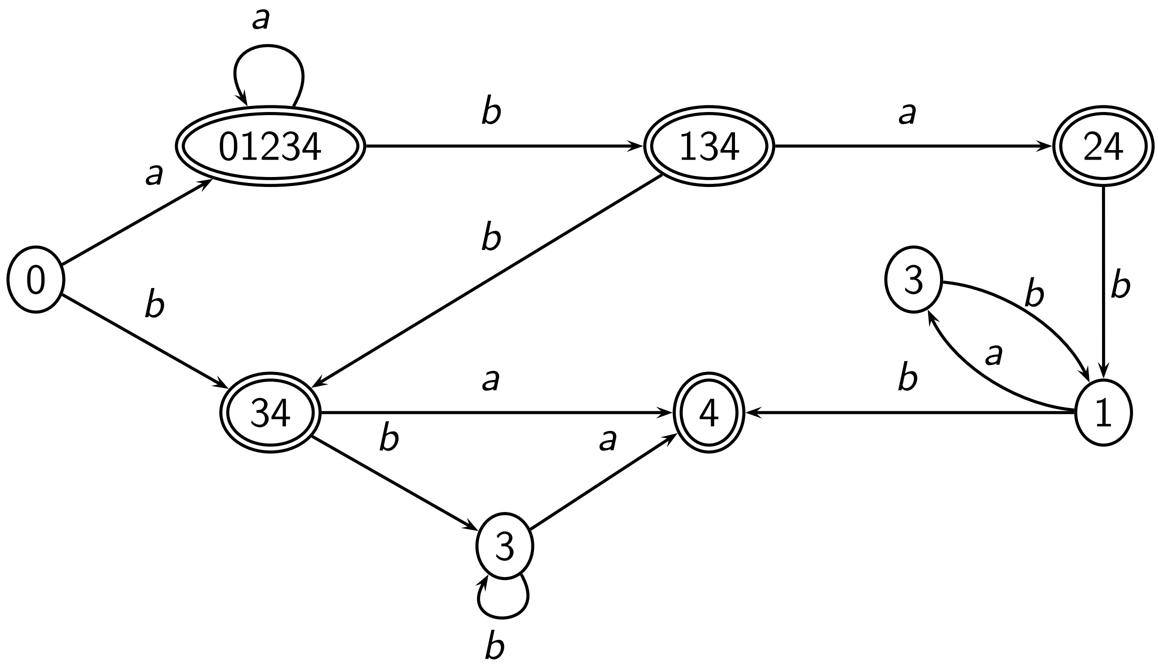


Transition function (in tabular form)

q	$\delta(q,a)$	$\delta(q,b)$	$\delta(q,\Lambda)$	$\delta^*(q,a)$	$\delta^*(q,b)$
0	$\{0\}$	$\emptyset$	$\{1,3\}$	$\{0,1,2,3,4\}$	$\{3,4\}$
1	$\{2\}$	$\{4\}$	$\emptyset$	$\{2\}$	$\{4\}$
2	$\emptyset$	$\{1\}$	$\emptyset$	$\emptyset$	$\{1\}$
3	$\{4\}$	$\{3\}$	$\emptyset$	$\{4\}$	$\{3\}$
4	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$

Example NFA to DFA (2)

This results in the following DFA.



This technique is called *subset construction*.

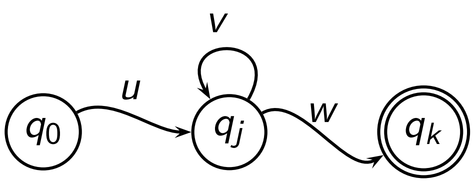
Pumping Lemma

A finite automaton, whether it is deterministic or nondeterministic, has a finite number of states, n. So, can it deal with an input x whose length is longer than n, i.e. $|x| > n$?
The *Pumping Lemma for Regular Languages* describes the conditions that x must adhere to if the automaton is to accept it.

Lemma (Pumping Lemma for Regular Languages)

Let $L \in \Sigma^*$ and $M = (Q, \Sigma, q_0, A, \delta)$ such that $L = L(M)$. If M has n states then for every $x \in L$ satisfying $|x| \geq n$, there are three strings u, v and w such that $x = uvw$ and:

- $|uv| \leq n$
- $|v| > 0$
- $\forall i \geq 0, uv^iw \in L$



Limitations of Regular Languages

Consider the language $AnBn = \{a^n b^n | n \geq 0\}$.

Let's assume there is a finite automaton that accepts $AnBn$.

Let $x = a^n b^n$. Since $x \in AnBn$ and $|x| \geq n$, the Pumping Lemma conditions must apply.

Condition 1	$|uv| \leq n$ and since the first n symbols of x are a's then all the symbols of u and v are a.
Condition 2	$v = a^k$ for some $k > 0$.
Condition 3	$uv^iw \in AnBn$ but $uv^iw = a^{n+k}b^n \notin AnBn$. Hence the contradiction implies the initial assumption that there exists a FA that accepts $AnBn$ must be false.

- DFAs cannot count.
- Not all languages are regular languages. In fact $AnBn$ is not a regular language.