Languages and Automata

Lecture notes, draft version

Alexandra Silva

Preface

This document contains the first draft of lecture notes for the course *Talen en Automata* (Languages and Automata) that I teach at the Radboud University Nijmegen. It is a first year, first semester course, that runs for 8 weeks, with one 90 min lecture a week. The aim of the course is to introduce students to the mathematics of regular languages, context-free languages and finite state machines. It covers core material in Computer Science that every student should eventually be familiar with. The subjects included in this course are best learned trough practice and hence every section ends with an extensive set of exercises. The exercises marked with a † are more challenging. In the appendix, I include a special set of exercises on induction and exams/tests from previous years.
There are a lot of great books and lecture notes on automata theory. I recommend the following

- J. Hopcroft, R. Motwani, J. Ullman. *Introduction to Automata Theory, Languages and Computation*.
- D. Kozen. *Automata and Computability*.
- E. Rich. *Automata, Computability and Complexity: Theory and Applications*.
- T. Sudkamp. *Languages and Machines*.
- Andrew Pitts. *Lecture Notes on Regular Languages and Finite Automata*. Available from `http://www.cl.cam.ac.uk/Teaching/1213/RLFA/materials.html`.

Sudkamp's book is also used in the *Berekenbaarheid* (Computability) course, taught by Freek Wiedijk.
The material in these notes is based on several sources including the above books/notes, and the author's own notes. Any errors are of course the author's responsibility. I kindly ask the readers to please report any error found to `alexandra@cs.ru.nl`. Updated versions of these notes will be produced and made available through the author's webpage.
I am grateful for the help and feedback of all the students and teaching assistants in the various editions of the course. A particular thanks to: Henning Basold, for designing exercises for the edition 2013/2014 of the course; Tim Steenvoorden, for his constant feedback on all aspects of the course which has definitely helped me greatly improving the lectures; and Bas Westerbaan, for his help in designing and grading exercises and for his constant availability to discuss about subtleties of proofs and various aspects of Mathematics.

Alexandra Silva
Nijmegen, December 2013

Note

Various typos were fixed in the 2014/2015 edition of course. I am grateful for feedback from students, in particular Lars Jellema, Rom Nijholt, and Rick Schouten, who thoroughly read the notes in the first week of the course and pointed out several typos and inconsistencies.

Alexandra Silva
Nijmegen, November 2014

Lecture 1

Regular Languages and Expressions

In this first lecture, we introduce languages over an alphabet and operations on languages. We characterize the subclass of regular languages and introduce regular sets.

1.1 Languages

The word *language* is used in various contexts – think, for instance, programming language (C, Java, ...) or natural language (English, Ducth, ...) – and the intuitive meaning is easily understood. In this course, we will be studying formal languages over a given alphabet and subclasses thereof. For the purpose of this course, a *language* is a (sub)set of words over an alphabet. Let us now precisely define all the ingredients involved.

1.1.1 Definition (Alphabet). An alphabet is a finite set, denoted by capital letter A, B, ..., whose elements are referred to as symbols or letters. ♣

1.1.2 Example (Alphabets).

1. The set $A = \{x, y, +, 1, 2, 3\}$ is an alphabet containing alphanumeric symbols.
2. The set $A = \{a, b, c, \dots, z\}$ is an alphabet containing the usual letters of the English alphabet.
3. The set $A = \{dog, cat, mother, is\}$ is an alphabet with 4 symbols.
4. The set $\mathbb{N} = \{1, 2, 3, 4, 5, \dots\}$ is not an alphabet because it is not finite. ♠

1.1.3 Definition (Word). A word of length n over an alphabet A is just a finite list/sequence of elements of A. The empty word is the empty sequence which will be denoted by λ. ♣

We will denote by $length(u)$ or $|u|$ the number of symbols of a given word u. We will denote by $|u|_a$ the function that counts the number of occurrences of a letter a in the word u.

1.1.4 Example (Strings).

1. *students, dogs, set* are words over the usual latin alphabet $A = \{a, b, c, \dots, z\}$ of length 8, 4 and 3, respectively. We have $|students|_s = 2$ and $|set|_t = 1$.

2. *x+1*, *x=2* and *3+2* are words over the alphabet $A = \{x, 1, 2, 3, +, =\}$.
3. *The students will do their homework* is a word, of length 11, over the alphabet

$$A = \{The, homework, will, do, students, their, \sqcup, dog, cat\}.$$

1.1.5 Definition (The set of all words). We shall denote by A^* the set defined by

$$A^* = \text{ set of all words over } A \text{ of any finite length.}$$

The set A^* can be defined inductively by the following two rules.

1. λ, the empty word, is an element of A^*.
2. If $w \in A^*$, then, for all $a \in A$, the word $aw \in A^*$. ♣

The second rule in the above definition could be equivalently replaced by

$$\text{If } w \in A^*, \text{ then, for all } a \in A, \text{ the word } wa \in A^*.$$

Note that the set A^* has the following two properties.

– A^* is non-empty, since it always contains the empty word λ.

– if A is non-empty, then A^* is infinite, since, for any $a \in A$ $\lambda, a, aa, aaa, \ldots$ are in A^*.

1.1.6 Example (Examples of A^*).

1. If $A = \{a\}$ then A^* contains exactly $\lambda, a, aa, aaa, \ldots$.
2. If $A = \{a, b\}$ then A^* contains

$$\lambda, a, b, aa, ab, ba, bb, aaa, \ldots$$

3. If $A = \emptyset$ then $A^* = \{\lambda\}$. ♠

1.1.7 Definition (Language). Let A be an alphabet. A language L is a *subset of all the words* over A, that is, $L \subseteq A^*$. ♣

1.1.8 Example (Languages).

1. $\emptyset$, the language with no words, and $\{\lambda\}$, the language containing only the empty word, are languages over any alphabet.
2. $\{\lambda, a, aaaa\}$ and $\{w \mid |w| \text{ is odd}\}$ are languages over $\{a\}$. Note that the second language, of words of odd length, does *not* contain the empty word. The empty word is always in A^* but not necessarily in a given language.
3. $\{w \mid w = aa \cdots abb \cdots b \text{ and } |w|_a = |w|_b\}$ is a language over $\{a, b\}$ containing all words that have a number of a's followed by the the same number of b's.

1.2 Operations on Words and Languages

First we define the following operations on words: concatenation and reverse.

If u and v are words we can form the word uv which is called the concatenation of u and v. For instance, the concatenation of abc and de is $abcde$.

1.2.1 Definition (Concatenation of words). The concatenation is a binary operation $\cdot : A^* \times A^* \to A^*$ defined inductively as follows.

$$\lambda \cdot u = u \qquad (aw)\cdot u = a(w \cdot u). \qquad \clubsuit$$

We will often drop the $\cdot$ and just write uv for $u \cdot v$.

1.2.2 Example. Let $A = \{a, b, c\}$ and consider the words $u = ca$, $v = b$ and $w = aaa$. We have

$$uv = cab \qquad wu = aaaca \qquad vv = bb \qquad vu = bca.$$

Note that $uv = cab \neq bca = vu$: the concatenation is not a commutative operation. ♠

1.2.3 Example (Intermezzo: a proof by induction). Let us illustrate the use of induction in the definition of functions over A^*. The function $|-|_a$ that returns the number of occurrences of a in a word is defined as

$$|\lambda|_a = 0 \qquad |bw|_a = \begin{cases} 1 + |w|_a & \text{if } b = a \\ |w|_a & \text{otherwise} \end{cases}.$$

Let us now prove that, given two words $u, v \in A^*$ and $a \in A$,

$$|uv|_a = |u|_a + |v|_a.$$

By induction on u. Base case: $u = \lambda$.

$$|\lambda v|_a = |v|_a = 0 + |v|_a = |\lambda|_a + |v|_a.$$

Inductive step: $u = bw$. Suppose $|wv|_a = |w|_a + |v|_a$ as induction hypothesis. If $b = a$, then

$$\begin{aligned} |bwv|_a &= 1 + |wv|_a && \text{by definition of } |-|_a\text{, since } a = b \\ &= 1 + |w|_a + |v|_a && \text{by induction hypothesis} \\ &= |bw|_a + |v|_a. && \text{by definition of } |-|_a\text{, since } a = b \end{aligned}$$

If $b \neq a$, then

$$\begin{aligned} |bwv|_a &= |wv|_a && \text{by definition of } |-|_a\text{, since } a \neq b \\ &= |w|_a + |v|_a && \text{by induction hypothesis.} \\ &= |bw|_a + |v|_a. && \text{by definition of } |-|_a\text{, since } a \neq b \end{aligned}$$

♠

The exercises contain more material on induction and there is also an extended exercise provided in the appendix.
We now define another operation on words, namely the reverse of a word u, which we denote by u^R and which is the word with the order of the letters reversed. E.g. $(abc)^R = cba$ and $(aa)^R = aa$.

1.2.4 Definition (Reverse of a word). The function $(-)^R\colon A^* \to A^*$ is defined by

$$\lambda^R = \lambda \qquad\qquad (aw)^R = w^R \cdot a.$$

This function has interesting properties such as $(u^R)^R = u$ and $(uv)^R = v^R u^R$. These can be proved by induction.
The above operations on words, concatenation and reverse, can be extended to languages.

1.2.5 Definition (Concatenation and reverse of languages). Given two languages $U, V \subset A^*$ their concatenation is defined as

$$UV = \{uv \mid u \in U, v \in V\}$$

and the reverse of U as

$$U^R = \{u^R \mid u \in U\}.$$

♣

These operations inherit some properties of the corresponding word operations. For instance, we have $(U^R)^R = U$ and $UV \neq VU$ in general.

1.2.6 Example (Concatenation and reverse of languages). Let $U = \{acb, a, \lambda\}$, $V = \{bca, a, \lambda\}$ and $W = \{cc\}$ be languages over the alphabet $A = \{a, b, c\}$. We have

- $UV = \{acbbca, acba, acb, abca, aa, a, bca, a, \lambda\}$ (why is $\lambda \in UV$?).
- $VW = \{bcacc, acc, cc\}$.
- $V^R = \{acb, a, \lambda\} = U$ and $W^R = W$.

♠

Since languages are just sets we also have the union of languages

$$U \cup V = \{w \mid w \in U \text{ or } w \in V\}$$

which contains all the words of both languages. The concatenation operation, together with the union, can be used to define an iteration operator on languages, usually referred to as *Kleene star*. Let U^n denote the concatenation of U with itself n times ($U^0 = \{\lambda\}$). For instance $U^3 = UUU = \{uvw \mid u, v, w \in U\}$. The Kleene star of language $U \subseteq A^*$ is defined as

$$U^* = \bigcup_{n \in \mathbb{N}} U^n = U^0 \cup U^1 \cup U^2 \cup \cdots$$

U^* contains all words that can be built from elements of U, that is

$$U^* = \{u_1 u_2 \cdots u_n \mid u_1, \ldots, u_n \in U, n \in \mathbb{N}\}.$$

1.2.7 EXAMPLE (Kleene star). Let $A = \{a, b\}$ and $U = \{ab, ba\}$. We have

$$\begin{aligned} &U^0 = \{\lambda\} \\ &U^1 = U = \{ab, ba\} \\ &U^2 = UU = \{abab, abba, baab, baba\} \\ &\vdots \end{aligned}$$

So, the set $U^* = U^0 \cup U^1 \cup \cdots$ contains exactly the words

$$\lambda, ab, ba, abab, abba, baba, baab, bababa, ababba, \ldots$$ ♠

Operations on languages can be used to specify/restrict the words that belong to a certain languages. For instance,

1. $L = \{a, b\}^*\{a\}$ contains all words, over $\{a, b\}$, that end with an a.
2. $L = \{a, b\}^*\{b\}\{a, b\}^*$ contains all words, over $\{a, b\}$, that have at least one b.
3. $L = \{bb\}^*$ contains all words, over $\{b\}$, that have an even length.

Sometimes it is convenient to denote all the *non-empty* words in X^* that is $X^* \setminus \{\lambda\}$. We will use X^+ to denote $X^* \setminus \{\lambda\} = XX^*$.

1.3 Regular sets and Expressions

If one defines languages using the operations defined above and starting from the basic languages $\emptyset$, $\{\lambda\}$ and $\{a\}$ this gives exactly the class of languages known as *regular languages*.

1.3.1 DEFINITION (Regular sets over A). Let A be an alphabet. The *regular sets* over A are defined inductively as follows

1. $\emptyset$, $\{\lambda\}$ and $\{a\}$ (for every $a \in A$) are regular sets over A.
2. Let U and V be regular sets over A. Then,

   $$U \cup V, \qquad UV, \qquad \text{and} \qquad U^*$$

   are regular sets over A.

A language is called *regular* if it is defined by a regular set. ♣

1.3.2 EXAMPLE (Regular Languages). Let $A = \{a, b\}$.

1. A is a regular set because
   $$\{a, b\} = \{a\} \cup \{b\}$$

2. The language
$$L = \{w \in A^* \mid |w|_b \geq 1\}$$
is regular. In particular, $L = A^*\{b\}A^*$ (and we know from above that A is regular).

3. The language
$$L = \{w \in A^* \mid |w| \text{ is even}\}$$
is regular, because $L = (AA)^* = (\{a,b\}\{a,b\})^*$ (and we know from above that A is regular). ♠

1.4 Exercises

1. Define formally the length function
$$|-|: A^* \to \mathbb{N}$$
which returns the number of characters/symbols of a word in A^*, using structural induction.

2. Prove by induction that, for all $u, w \in A^*$,
$$|uw| \quad = |u| + |w|$$

3. Which of the following properties hold for all $u, v, w \in A^*$?
   a) $uv = vu$
   b) $(uv)w = u(vw)$
   c) $u^R = u$

   For the ones that *do not hold* provide a counter-example.

4. Prove that, for any $u, v \in A^*$,
   a) $(u^R)^R = u$ and
   b) $(uv)^R = v^R u^R$.

5. Let $L_k = \{w \in A^* \bigm| |w|_a = k\}$ be the languages with k occurrences of the letter a in a word w. Write down the sets for the following concatenations
   a) $L_5\emptyset$,
   b) $L_5\{\lambda\}$, and
   c) L_5L_2.

6. Describe the following languages over the alphabet $A = \{a, b\}$ using regular set operations.

a) $\{x \mid x \in A^*, |x| > 3,$ and x contains at least one $a\}$.

b) $\{x \mid x \in A^*, |x| > 3,$ and x contains an even number of b's$\}$.

c) $\{x \mid x \in A^*, |x| > 3,$ and x contains an even number of a's and such that all a's occur before any occurrence of $b\}$.

d) $\{x \mid x \in A^*, |x|$ is even and x contains exactly two a's$\}$.

7. Consider the language

$$L = \left\{w \in A^* \,\middle|\, |w|_a \text{ is odd}\right\}$$

   a) Show that L is regular. This means, that you have to construct a regular language X using the basic building blocks given in the lecture ($\emptyset, \{\lambda\}, \{a\}, \cup$, concatenation and star) and then show, that X is the same as L.

   Hint: to show, that $X = L$, show $X \subseteq L$ and $L \subseteq X$ separately.

   b) Show that L^R is regular.

8. (†) Let X be a language over an alphabet A. We say that X is transitive if $XX \subseteq X$ and reflexive if $\lambda \in X$. Prove that for any language L, L^* is the smallest reflexive and transitive set containing L. (That is, show that L^* is reflexive and transitive and that if X is any other reflexive and transitive set satisfying $L \subseteq X$, then $L^* \subseteq X$.)

Lecture 2

Regular Expressions and Deterministic Finite Automata

In this second lecture, we introduce regular expressions as an alternative formalism to denote regular languages. We then take a more operational view on languages and define deterministic finite automata.

2.1 Regular Expressions

The notation above for regular languages is quite verbose (e.g. the language with only one letter requires 3 symbols). In order to have a more user friendly syntax to denote regular languages, regular expressions were introduced. Regular languages play an important role in formal languages, pattern recognition and the theory of finite state systems. Most of you have come across regular expressions in different programming languages or in text editors when searching for a certain pattern in a text.

2.1.1 Definition (Regular expressions over A). Let A be an alphabet. The *regular expressions* over A are defined inductively as follows

1. 0, 1 and a (for every $a \in A$) are regular sets over A.
2. Let r and s be regular expressions over A. Then,

   $$(r+s), \qquad rs, \qquad \text{and} \qquad (r)^*$$

   are regular expressions over A. ♣

Examples of regular expressions include $(a+b)$, ab, $(a+b)ab$ and $(ab)^*$.

Remark (Binding preference). In the definition above we implicitly assume that the alphabet A does not contain the symbols

$$0, 1, (,). +, ^*$$

Then, concretely, the set of regular expressions form a language over the alphabet containing A and these extra 6 symbols.
To make things more readable, we drop as many parentheses as possible by adopting the following convention: * bind more tightly than concatenation which binds more tightly than $+$.
For example, $r+st^*$ means $(r+s(t)^*)$ and not $(r+s)(t)^*$ or $(r+(st)^*)$.

Regular expressions are used to define languages – they are often used to match words. Each regular expression r denotes a regular language $\mathcal{L}(r)$ which we define next.

2.1.3 Definition (Language denoted by a regular expression). The map $\mathcal{L}$ associating with each regular expression the language it denotes is defined by

$$\begin{array}{lcl} \mathcal{L}(0) & = & \emptyset \\ \mathcal{L}(1) & = & \{\lambda\} \\ \mathcal{L}(a) & = & \{a\} \\ \mathcal{L}(r+s) & = & \mathcal{L}(r) \cup \mathcal{L}(s) \\ \mathcal{L}(rs) & = & \mathcal{L}(r)\mathcal{L}(s) \\ \mathcal{L}(r^*) & = & (\mathcal{L}(r))^* \end{array}$$

♣

As for regular languages we will use r^+ as a shorthand for rr^*.
Regular expressions are *syntax* to denote regular languages. It is important to keep in mind that there are these two separate worlds: the syntax, given by a regular expression r, and the *semantics*, provided by the language $\mathcal{L}(r)$ that the expression denotes.

2.1.4 Example (Languages denoted by expressions). Let $A = \{a, b\}$ be a two-letter alphabet.

syntax – regular expression r	semantics – language $\mathcal{L}(r)$
$a+b$	any symbol in $A = \{a, b\}$
$(a+b)^*$	all words over A
$a(a+b)^*$	all words that begin with an a
$((a+b)(a+b))^*$	all words of even length
$01+0$	no words (the empty language)
a^*ba^*	all words over A with exactly one b
$(a+b)^*(a+b)^*$	all words over A

In the example above you see that the expressions $(a+b)^*$ and $(a+b)^*(a+b)^*$ denote the same language. Different expressions, as they are syntax, can denote the same language. When that happens, we say that the expressions are equivalent.

2.1.5 Definition (Equivalent regular expressions). Two regular expressions r and s are *equivalent* if and only if $\mathcal{L}(r) = \mathcal{L}(s)$. When two expressions are equivalent we write $r = s$ or $r \equiv s$. ♣

2.1.6 Example. The following equivalences hold.

$$r+s \equiv s+r \qquad s0 \equiv 0 \qquad 0+s \equiv s$$

Let us show the last two:

$$\mathcal{L}(s0) = \mathcal{L}(s)\mathcal{L}(0) = \mathcal{L}(s)\emptyset = \emptyset = \mathcal{L}(0).$$
$$\mathcal{L}(0+s) = \mathcal{L}(0) \cup \mathcal{L}(s) = \emptyset \cup \mathcal{L}(s) = \mathcal{L}(s).$$

The expression rs and sr are not equivalent in general (why?).
Some equivalences are (relatively) obvious but sometimes it gets trickier to show that two expressions are equivalent. Hence, automatizing the equivalence process is convenient. We do not explore this in this course but there is a lot done in the area of algorithms to decide whether two regular sets/expressions are equivalent.
Here are two examples of equivalences that require a bit more ingenuity in proving their correctness.

$$(a^*b)^*a^* \equiv (a+b)^* \qquad\qquad a(ba)^* \equiv (ab)^*a$$

Let us show the last one. First note that

$$\mathcal{L}(a(ba)^*) = \{a\} \cdot \left(\bigcup_{n\in\mathbb{N}} \mathcal{L}(ba)^n \right) = \bigcup_{n\in\mathbb{N}} \{a\} \cdot (\{ba\})^n$$

Now observe that

$$\begin{aligned} w \in \{a\} \cdot (\{ba\})^n \quad &\iff \quad w = a\underbrace{(ba)(ba)\cdots(ba)}_{n \text{ times}} \\ &\iff \quad w = \underbrace{(ab)(ab)\cdots(ab)}_{n \text{ times}}a \\ &\iff \quad w \in (\{ab\})^n \cdot \{a\}). \end{aligned}$$

Hence,

$$\mathcal{L}(a(ba)^*) = \bigcup_{n\in\mathbb{N}} \{a\} \cdot (\{ba\})^n = \bigcup_{n\in\mathbb{N}} (\{ab\})^n \cdot \{a\} = \mathcal{L}((ab)^*a).$$

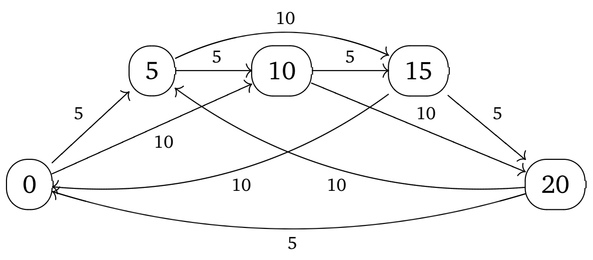
♠

2.2 Deterministic Finite Automata

We will now study languages from a more operational perspective, using finite-state machines. The goal is to define a notion of *machine* that accepts languages. The motivation behind finite state machines is that they are easier to run (from an algorithmic perspective) that regular expressions and many times defining a machine is intuitively easier than defining a regular expression. We will introduce several notions of *language acceptors*. In this lecture, we will restrict ourselves to a deterministic version of finite state machines – deterministic finite automata.

Finite state machines are devices that have (internal) control states and transitions between states are based on certain inputs from the environment. Intuitively, we can explain states and transitions as follows. Take the two-letter alphabet $A = \{5, 10\}$ and imagine that the letters in A denote inserting 5 and 10 cents into a machine where you can get a newspaper. This machine is very simplistic: the news paper costs 25 cents and when that amount (or greater) is inserted it delivers the newspaper; the

machine does not give change but leftover credit can be used by the next person. The control states of this machine will represent the amount of credit at a given moment. The transitions allow to move state by inserting more money.
We could symbolically represent this newspaper machine as follows, where the numbers inside the circles indicate how much credit is left:



Note that in this example, every state has the possibility of accepting both 5 and 10 cents and moreover there is exactly one transition from each state labelled by 5 and 10: this is what makes this machine deterministic.

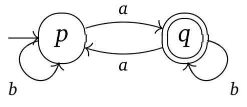
2.2.1 Definition (Deterministic Finite Automata). A deterministic finite automaton (DFA) over an alphabet A is a tuple (Q, δ, q_0, F) where

- Q is a *finite* set of states;
- $\delta : Q \times A \to Q$ is the transition function;
- $q_0 \in Q$ is the initial state;
- $F \subseteq Q$ is the set of final/accepting states. ♣

Notation: We will be using the following notation:

- $\rightarrow q_0$ – an incoming arrow will point to the initial state.
- $\boxed{q}$ – a double circle indicates that the state q is final.
- $p \xrightarrow{a} q$ – indicates $\delta(p, a) = q$.
- $p \xrightarrow{a,b} q$ is a shorthand for $\begin{tikzcd} p \arrow[r, bend left, "a"] \arrow[r, bend right, "b"'] & q \end{tikzcd}$

Let $A = \{a, b\}$. The following diagram



denotes a DFA (Q, δ, q_0, F) where

- $Q = \{p, q\}$;
- $q_0 = p$;
- $\delta(p,a) = q, \delta(p,b) = p, \delta(q,a) = p, \delta(q,b) = q$. We can represent the transition function as a matrix:

δ	a	b
p	q	p
q	p	q

- $F = \{q\}$.

Remark: Note that the transition function of the automaton is total: every state has to have exactly one transition for each letter in $a \in A$. Also note that $\delta : Q \times A \to Q$ can alternatively and equivalently be represented as a function $\delta : Q \to Q^A$, where Q^A denotes the set of functions $A \to Q$. The switch between these two representations is known in functional programming as *currying/uncurrying*.

A DFA will *accept* a word if starting from the initial state and moving through the automaton' states by reading *all* the letters of the word from left-to-right the state reached is final.
Take the word *babaa* and let us *run* the automaton in the example above:

$$\begin{tikzcd}
{} \arrow[r] & p \arrow[r, "b"] & p \arrow[r, "a"] & q \arrow[r, "b"] & q \arrow[r, "a"] & p \arrow[r, "a"] & q
\end{tikzcd}$$

babaa ~~*b*~~*abaa* ~~*ba*~~*baa* ~~*bab*~~*aa* ~~*baba*~~*a* ~~*babaa*~~

When we are finished reading the word the automaton is in state q which is final and therefore the word is accepted.
Take the word *baba* and let us again *run* the automaton above:

$$\begin{tikzcd}
{} \arrow[r] & p \arrow[r, "b"] & p \arrow[r, "a"] & q \arrow[r, "b"] & q \arrow[r, "a"] & p
\end{tikzcd}$$

baba ~~*b*~~*aba* ~~*ba*~~*ba* ~~*bab*~~*a* ~~*baba*~~

When we are finished reading the word the automaton is now in state p which is not final and therefore the word is not accepted.
How do we formally describe the language accepted by an automaton? We need to formalize the notion of moving through the automaton with a given word. Given a state $q \in Q$ and a word $w \in A^*$ we define $\delta^\star : Q \times A^* \to Q$ such that $\delta^\star(q, w)$ returns the state reached in the automaton after reading the word w. The function δ^* is the inductive extension of δ to words (intuitively, think of δ as providing you the one-step information and δ^* can do multiple steps by using δ for the singular steps).

2.2.2 Definition (δ^* of a DFA). The function $\delta^\star\colon Q \times A^* \to Q$ is given by

$$\delta^*(q,\lambda) = q \qquad \underbrace{\delta^*(q,aw) = \delta^*(\underbrace{\delta(a,q)}_{\text{one } a\text{-step}}, w)}_{\text{running the word } \cancel{a}\text{w}}$$

♣

We are now ready to define the language accepted by a DFA.

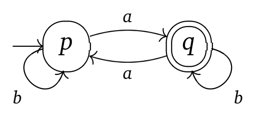
2.2.3 Definition (Language accepted by a DFA). Given a DFA $\mathcal{A} = (Q, q_0, \delta, F)$ we define, for every $q \in Q$,

$$\mathcal{L}(q) = \{w \in A^* \mid \delta^*(q,w) \in F\}.$$

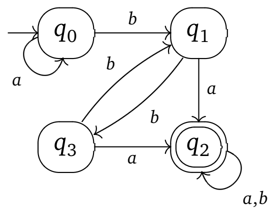
The language accepted by $\mathcal{A}$ is the language $\mathcal{L}(q_0)$ accepted by its initial state. ♣

One thing to note in this definition is that the accepted language function $\mathcal{L}$ is defined for every state $q \in Q$ and then the language of a given automaton is just the language of its initial state. This means that one can compare states p and q of the same automaton by asking whether $\mathcal{L}(q)$ and $\mathcal{L}(p)$ are the same.

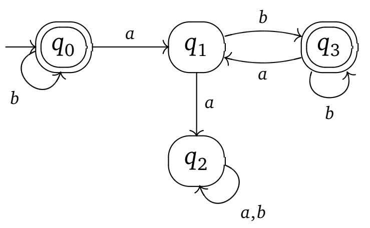
2.2.4 Example (Language accepted by a DFA). Fix $A = \{a, b\}$.



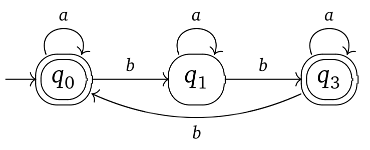
$\mathcal{L}(p) = \{w \in A^* \mid |w|_a \text{ is odd}\}$.



$\mathcal{L}(q_0) = \{w \in A^* \mid w \text{ has the subword } ba\}$.



$\mathcal{L}(q_0) = \{w \in A^* \mid \text{every } a \text{ in } w \text{ is followed by a } b\}$.



$\mathcal{L}(q_0) = \{w \in A^* \mid |w|_b = 3n+2 \text{ or } |w|_b = 3n, n \in \mathbb{N}\}$.

♠

2.3 Exercises

1. Describe the following languages over the alphabet $A = \{a, b\}$ using regular expressions.

   a) $\{x \mid x \in A^*, |x| > 3,$ and x contains at least one $a\}$.

   b) $\{x \mid x \in A^*, |x| > 3,$ and x contains an even number of b's$\}$.

   c) $\{x \mid x \in A^*, |x| > 3,$ and x contains an even number of a's and such that all a's occur before any occurrence of $b\}$.

   d) $\{x \mid x \in A^*, |x|$ is even and x contained exactly two a's$\}$.

2. Describe the languages defined by the following regular expressions over $A = \{a, b\}$:

   a) $(a + b)(a + b)(a + b)^*$

   b) $a^*ba + b^*ab$

3. Let $A = \{a, b\}$. Construct a regular expression e, such that its generated language $\mathcal{L}(e)$ is

   $$L = \left\{ w \in A^* \,\middle|\, w \text{ contains } aba \text{ \underline{at least} once} \right\}.$$

   Argue, why $\mathcal{L}(e) = L$ holds.

4. More generally, let A be an alphabet and $v \in A^*$ a word. Define a regular expression has(A, v) which recognizes the language

   $$\text{Has}(A, v) = \left\{ w \in A^* \,\middle|\, w \text{ contains } v \right\}$$

   and argue again, why $\mathcal{L}(\text{has}(A, v)) = \text{Has}(A, v)$.

5. (†) Show that for any regular language L, the reversed language L^R is again regular.

   Hint: For a regular expression e define e^R.

6. Use the regular expression identities provided below to show

   a) $(ba)^+(a^*b^* + a^*) = (ba)^*ba^+(b^* + \lambda)$

   b) $(a + b)^* = (a + b)^*b^*$

   c) (†) $(ab + a)^*a = a(ba + a)^*$.

For regular expressions e, e_1, e_2, e_3, the following identities hold:

$$
\begin{array}{lclcll}
e_1 + (e_2 + e_3) & = & (e_1 + e_2) + e_3 & & & \text{(associativity of } +\text{)} \\
e_1 + e_2 & = & e_2 + e_1 & & & \text{(commutativity of } +\text{)} \\
e + e & = & e & & & \text{(idempotency of } +\text{)} \\
e + 0 & = & e & & & \text{(0 is an identity of } +\text{)} \\
 & & & & & \\
e_1(e_2 e_3) & = & (e_1 e_2) e_3 & & & \text{(associativity of } \cdot\text{)} \\
e1 & = & e & = & 1e & \text{(1 is an identity of } \cdot\text{)} \\
e0 & = & 0 & = & 0e & \text{(0 is an annihilator of } \cdot\text{)} \\
 & & & & & \\
(e_2 + e_3) e_1 & = & e_2 e_1 + e_3 e_1 & & & \text{(right distributivity)} \\
e_1(e_2 + e_3) & = & e_1 e_2 + e_1 e_3 & & & \text{(left distributivity)} \\
e^* e + \lambda & = & e^* & & & \\
e e^* + \lambda & = & e^* & & & \\
 & & & & & \\
1^* & = & 0^* = 1 & & & \\
e^* & = & (e^*)^* & & & \\
(e_1 e_2)^* e_1 & = & e_1 (e_2 e_1)^* & & & \text{(sliding rule)} \\
(e_1 + e_2)^* & = & (e_1^* e_2)^* e_1^* & & & \text{(denesting)} \\
(e_1 + e_2)^* & = & e_1^* (e_1 + e_2)^* = (e_1^* e_2^*)^* & & & \\
 & & & & & \\
e^+ & = & e e^* = e^* e & & &
\end{array}
$$

7. (†) Give a regular expression over $\{a, b\}$ characterizing the language

   $\{x \mid x$ does not contain the subword $aba\}$

8. (†) Let A be an alphabet and $w \in A^*$. We say that $v \in A^*$ is a *subword* of w, if all letters of v occur in w in the same order. For example λ is a subword of any word, whereas aba has subwords $\{\lambda, a, b, ab, aa, ba, aba\}$. We denote the set of all subwords of w by $\mathrm{Sub}(w)$.

   Define by induction a map $\mathrm{sub}\colon A^* \to \mathit{RegExp}$ assigning to each word w a regular expression that recognises the language $\mathrm{Sub}(w)$ of all subwords of w. Argue, why $\mathcal{L}(\mathrm{sub}(w)) = \mathrm{Sub}(w)$ holds.

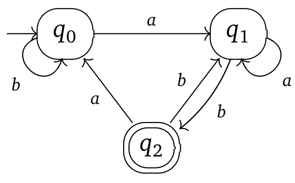
9. (†) Let $P = \{p_1, \dots, p_n\}$ be an alphabet and which does not contain the symbol "|", set $A = P \cup \{|\}$ and let

   $$L = \{t \in A^* \mid t = v_1|v_2|\dots|v_\ell|, \text{ in every } v_i \text{ each } p_k \in P \\ \text{occurs up to one time and } p_k \text{ occurs before } p_{k'} \text{ for } k < k'\}$$

   An example of a word in L is $p_1||p_1p_2|p_2||p_3|$, while *non*-examples are $p_1p_2|p_2$ since the last vertical bar is missing, $p_1p_1|p_2|$ since p_1 occurs twice in the first step and $p_2p_1|p_2|$ since p_2 occurs before p_1. Define a regular expression e such that $\mathcal{L}(e) = L$ and argue why the equality holds.

Hint: you can solve this by using the word $w_0 = p_1 \dots p_n$.

10. Let $\mathcal{A}$ be the deterministic automaton with state diagram



Which of the words $baba$, $baab$, $abaaab$ are accepted by $\mathcal{A}$?

11. Give a regular expression that denotes the language accepted by the automaton $\mathcal{A}$ of exercise 1.

12. Build a deterministic finite automaton that accepts the set of words over $\{a, b\}$ in which the subword aa occurs at least twice.

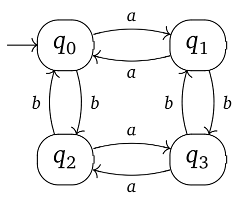
13. Build a deterministic finite automaton that accepts the set of words over $\{a, b\}$ in which the number of a's is divisible by 3.

14. Let $A = \{a, b\}$ a two letter alphabet. Construct a deterministic automaton $\mathcal{A}$ which accepts the language

$$L = \left\{ w \in A^* \,\middle|\, w \text{ contains } aba \text{ \underline{exactly} once} \right\}.$$

You don't need to prove that the languages are equivalent. Note that $ababa$ contains aba twice.

15. We define a family of DFA over the alphabet $A = \{a, b\}$ by letting the states Q, the transition map δ and the initial state q_0 be as in the following picture.



Give explicit descriptions of the languages $L_1, \dots, L_4$ accepted by the automata $D_i = (Q, \delta, q_0, F_i)$ with accepting states $F_1 = \emptyset$, $F_2 = \{q_0\}$, $F_3 = \{q_3\}$ and $F_4 = \{q_1, q_2\}$. Argue in each case briefly, why those are the accepted languages.

16. Given two DFAs $\mathcal{A}_1, \mathcal{A}_2$ with the same alphabet, construct a third such DFA $\mathcal{A}$ with the property that u is accepted by $\mathcal{A}$ iff it is accepted by both $\mathcal{A}_1$ and $\mathcal{A}_2$.

Hint: take the states of $\mathcal{A}$ to be ordered pairs (q_1, q_2) of states of $\mathcal{A}_1$ and $\mathcal{A}_2$, respectively.

Lecture 3

Non-deterministic Finite Automata

In the third lecture, we introduce two other types of automata, less restrictive than DFA, which are more compact (in terms of number of states and transitions) language acceptors ad enable easy composition of automata.

3.1 Non-deterministic Finite Automata

Consider the language

$$L = \{w \in \{a,b\}^* \mid \text{the fifth symbol from the right is an } a\}$$

We have

$$abbbb, aaaaaa, bbbabbba \in L \quad \text{and} \quad babba, aabbaaa, bbbbbb \notin L$$

We would like to show that L is regular. Thinking up a regular expression r such that $\mathcal{L}(r) = L$ is relatively easy:

$$(a+b)^*a(a+b)(a+b)(a+b)(a+b)$$

But how about a DFA? Is this possible? We will see later that a language is regular if and only if it is accepted by a DFA, so it must be possible. But the obvious automaton one would draw has a slight problem

$$\begin{tikzcd}
{} \arrow[r] & x \arrow[loop above, "{a,b}"] \arrow[r, "a"] & 5 \arrow[r, "{a,b}"] & 4 \arrow[r, "{a,b}"] & 3 \arrow[r, "{a,b}"] & 2 \arrow[r, "{a,b}"] & \boxed{1}
\end{tikzcd}$$

State x has two outgoing a transitions and the final state 1 does not have transitions defined. Still, this is a convenient automaton to accept the above language and in fact a deterministic automaton accepting L will have a minimum of 32 (!) states.
This new type of automata, non-deterministic, offer a more flexible and compact way of accepting languages. As we will see, they still accept exactly the same languages as DFA, namely regular languages.
What is the type of the new transition structure?
In the above example, we need $\delta(x,a) = x$ or 5 and we need $\delta(1,a) = \delta(1,b) =$ *no states*. That can be formalized by considering *subsets of states*, which also includes the empty subset allowing us to return *no transitions*. Define

$$\mathcal{P}(Q) = \{Y \mid Y \subseteq Q\}$$

Now the transition function of the above automaton has type $\delta : Q \times A \to \mathcal{P}(Q)$ and would be defined as

δ	a	b
x	$\{x, 5\}$	$\{5\}$
5	$\{4\}$	$\{4\}$
4	$\{3\}$	$\{3\}$
3	$\{2\}$	$\{2\}$
2	$\{1\}$	$\{1\}$
1	$\emptyset$	$\emptyset$

Formally a non-deterministic finite automaton is defined as follows.

3.1.1 Definition (Non-deterministic Finite Automata). A non-deterministic finite automaton (NFA) over an alphabet A is a tuple (Q, δ, q_0, F) where

- Q is a *finite* set of states;
- $\delta : Q \times A \to \mathcal{P}(Q)$ is the transition function;
- $q_0 \in Q$ is the initial state;
- $F \subseteq Q$ is the set of final/accepting states. ♣

We now need to define when a word is accepted by an NFA. Note that now a word w can label more than one path in the automaton, whereas for DFA every word labelled exactly one path. In this new setting, a word is accepted by an NFA if *there exists* one path from the initial state, labelled by w, that leads to a final state.
In order to compute which states can be reached by reading w we again extend the transition function to words (similar to Definition 2.2.2).

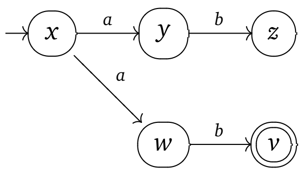
3.1.2 Definition (δ^* of an NFA). Given an NFA (Q, δ, q_0, F), we define $\delta^* : Q \times A^* \to \mathcal{P}(Q)$ such that $\delta^*(q, w)$ returns the states reached by reading w starting from q.

$$\delta^*(q, \lambda) = \{q\} \qquad\qquad \delta^*(q, aw) = \bigcup_{p \in \delta(q,a)} \delta^*(p, w)$$
♣

The union in the second clause above processes all states p reached after one a-step – $p \in \delta^*(q, a)$ – and collects all states that can be reached from such p's after reading the rest of the word w.
For a word w to be accepted we want that there exists at least one path, starting from the initial state q_0, labelled by w and leading to a final state. That means that among the states reached after reading w – $\delta^*(q_0, w)$ – there exists at least one final state. That is:

$$\delta^*(q_0, w) \cap F \neq \emptyset$$

Take for instance the NFA



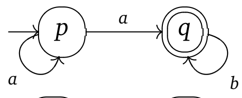
We then have $\delta^*(x, ab) = \{v, z\}$, $F = \{v\}$ and $\{v, z\} \cap \{v\} = \{v\} \neq \emptyset$. Hence ab is accepted. We have $\delta^*(x, a) = \{y, w\}$ and $\{y, w\} \cap \{v\} = \emptyset$. Hence a is not accepted. We are now ready to define the language accepted by a DFA.

3.1.3 Definition (Language accepted by an NFA). Given an NFA $\mathcal{A} = (Q, q_0, \delta, F)$ we define the language accepted by a state $q \in Q$ as

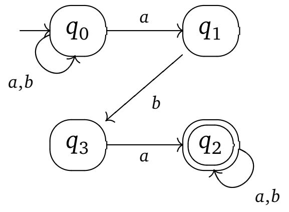
$$\mathcal{L}(q) = \{w \in A^* \mid \delta^*(q, w) \cap F \neq \emptyset\}.$$

The language accepted by $\mathcal{A}$ is the language $\mathcal{L}(q_0)$. ♣

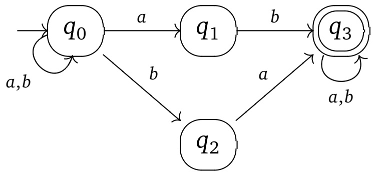
3.1.4 Example (Language accepted by an NFA). Fix $A = \{a, b\}$.



$\mathcal{L}(p) = \{a^n a b^m \mid n, m \in \mathbb{N}\}$.



$\mathcal{L}(q_0) = \{w \in A^* \mid w$ has the subword $aba\}$.

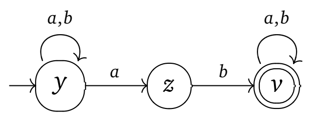


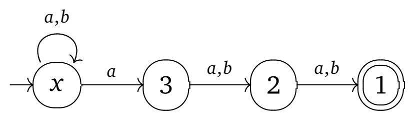
$\mathcal{L}(q_0) = \{w \in A^* \mid$ w has the subword ab **or** w has the subword $ba\}$.

3.2 Non-deterministic automata with λ-transitions

The last example above seems to hint that NFA are also suitable to build an automaton that accepts the union of languages. But let us look at another example. Can we easily build an NFA that accepts *words that contain the subword ab* ***or*** *that have a as third symbol from the right*?

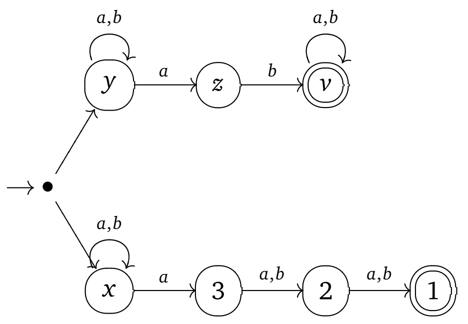
We know how to build an automaton for each of the above languages:





♠

How to combine them in a compositional way? Diagrammatically, we would like to be able to do the following:



But how do we handle the unlabeled transitions? We will actually label them with λ, the empty word, and define NFA-λ a class of non-deterministic automata that allows *empty* transitions.

3.2.1 Definition (Non-deterministic Finite Automata with λ transitions). A non-deterministic finite automaton with λ transitions (NFA-λ) over an alphabet A is a tuple (Q, δ, q_0, F) where

– Q is a *finite* set of states;

- $\delta : Q \times (A \cup \{\lambda\}) \to \mathcal{P}(Q)$ is the transition function;
- $q_0 \in Q$ is the initial state;
- $F \subseteq Q$ is the set of final/accepting states. ♣

We now need to define when a word is accepted by an NFA-λ. At every step, when reading a word w we can opt by reading a letter a and moving in the automaton with an a transition or first move with a λ to a new state q and then read the word w from the new state q.
We define $\lambda-\text{closure}: Q \to \mathcal{P}(Q)$ as the function that given $q \in Q$ returns all states reached by only taking λ-transitions starting from q.

3.2.2 Definition (λ-closure). Given an NFA-λ $\mathcal{A} = (Q, q_0, \delta, F)$ and $q \in Q$, we define $\lambda-\text{closure}(q)$ inductively:

1. $q \in \lambda-\text{closure}(q)$ and
2. If $p \in \lambda-\text{closure}(q)$ and $p' \in \delta(p, \lambda)$ then $p' \in \lambda-\text{closure}(q)$ ♣

Option 1. describes that the automaton remains in the current state whereas in option 2. the state changes by using a λ-transition. For instance, in the above example, $\lambda-\text{closure}(\bullet) = \{\bullet, x, y\}$ and $\lambda-\text{closure}(v) = \{v\}$.
In order to compute which states can be reached by reading w we again extend the transition function to words, but now taking into account that intermediate λ steps can occur.

3.2.3 Definition (δ^* of an NFA-λ). Given an NFA-λ, we define $\delta^* : Q \times A^* \to \mathcal{P}(Q)$ as follows.

$$\delta^*(q, \lambda) = \lambda-\text{closure}(q) \qquad \delta^*(q, aw) = \bigcup_{q' \in \lambda-\text{closure}(q)} \bigcup_{p \in \delta(q', a)} \delta^*(p, w)$$ ♣

In the above example, $\delta^*(\bullet)(a) = \delta(x, a) \cup \delta(y, a) = \{x, 3, y, z\}$.
The language accepted by an NFA-λ is now defined in the same way as for NFA's but now with the new δ^*.

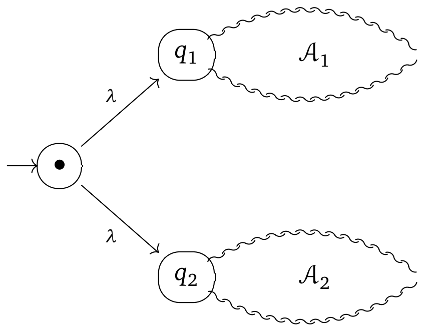
3.2.4 Definition (Language accepted by an NFA-λ). Given an NFA-λ $\mathcal{A} = (Q, q_0, \delta, F)$ we define the language accepted by a state $q \in Q$ as

$$\mathcal{L}(q) = \{w \in A^* \mid \delta^*(q, w) \cap F \neq \emptyset\}.$$

The language accepted by $\mathcal{A}$ is the language $\mathcal{L}(q_0)$. ♣

NFA-λ are very convenient to compose simpler automata. For instance, suppose we are given NFA's (possibly with λ's) $\mathcal{A}$, $\mathcal{A}_1$ and $\mathcal{A}_2$ that accept languages L, L_1 and L_2, respectively. We can use NFA-λ to build automata that accept $L_1 \cup L_2$, $L_1 L_2$ and L^*.

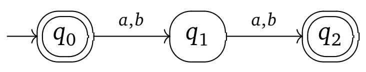
For the union we add an extra initial state, which becomes the new initial state and connect it to the initial states of $\mathcal{A}_1$ and $\mathcal{A}_2$ using λ transitions.



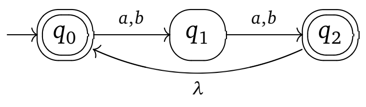
How about for the concatenation and star?
Here is a concrete example on how λ-transitions can be used to (more) easily build an automaton. Take the language

$$L = \{w \in \{a,b\}^* \mid |w| \text{ is even } \}$$

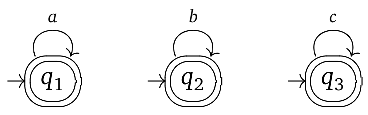
We know how to build the automaton that recognizes the words of length 0 and 2.



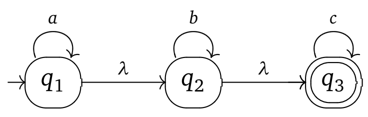
Now the automaton accepting L can easily be constructed:



For another example, imagine you are asked to build an automaton that accepts the language denoted by the regular expression $a^*b^*c^*$. Again, we can easily build automata recognizing the languages denoted by a^*, b^* and c^*.

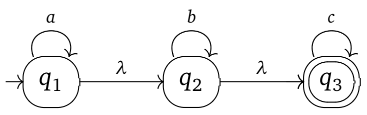


And now the automaton for $\mathcal{L}(a^*b^*c^*)$ is compositionally built from the above three automata using λ-transitions.



3.3 Removing λ-transitions

Adding non-determinism and λ-transitions increases the versatility of automata and allows us to have compacter acceptors of languages. However, the class of languages that DFA, NFA and NFA-λ accept is the same: the class of regular languages.
We will prove this result, known as Kleene's theorem, in the next lecture. For now, let us just show how to remove λ-transitions. That is, how to build an NFA starting from an NFA-λ that recognizes the same language. Intuitively, what happens is the following. Consider the automaton above accepting $\mathcal{L}(a^*b^*c^*)$:



For each state we remove the λ transitions as follows. Take a state $q \in \{q_1, q_2, q_3\}$. We need to do two things.

1. The state q is final if is was already final in the given automaton or if it has a chain of λ transitions to a final state. That is, λ−closure$(q) \cap F \neq \emptyset$.

2. The state q will have an a transition to another state q' if this transition was already there in the given automaton or if there is a path

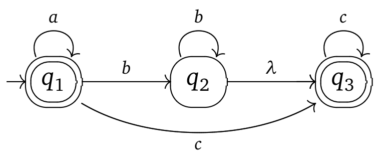
$$q \xrightarrow{\lambda} \bullet \xrightarrow{\lambda} \cdots \xrightarrow{\lambda} \bullet \xrightarrow{a} q'$$

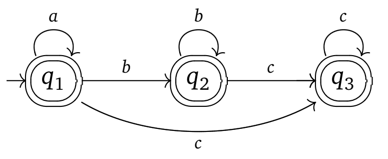
   in the automaton. In some books you will find that the definition considers λ's on both sides of the a transition:

$$q \xrightarrow{\lambda} \bullet \xrightarrow{\lambda} \cdots \xrightarrow{\lambda} \bullet \xrightarrow{a} \bullet \xrightarrow{\lambda} \cdots \xrightarrow{\lambda} q'$$

   but these are equivalent and for simplicity I take the first.

Below you can see the elimination of λ-transitions for states q_1 and q_2, respectively (q_3).





3.3.1 Definition (NFA-λ to NFA). Given an NFA-λ $\mathcal{A} = (Q, \delta, q_0, F)$ we build an NFA $\bar{\mathcal{A}} = (Q, \bar{\delta}, q_0, \bar{F})$ where

- $\bar{F} = \{q \in Q \mid \lambda\text{−closure}(q) \cap F \neq \emptyset\}$ and
- $\bar{\delta}(q,a) = \bigcup_{p \in \lambda\text{−closure}(q)} \delta(p,a)$. ♣

The new automaton $\bar{\mathcal{A}}$ recognizes the same language.

3.3.2 Theorem. *Given an NFA-λ $\mathcal{A} = (Q, \delta, q_0, F)$, the corresponding automaton $\bar{\mathcal{A}} = (Q, \bar{\delta}, q_0, \bar{F})$ after elimination of λ-transitions accepts the same language.*

Proof. We want to show that, for all $w \in A^*$,

$$w \text{ is accepted by } \mathcal{A} \iff w \text{ is accepted by } \bar{\mathcal{A}}$$

We show a slightly more general result, by induction on words,

$$\delta^*(q,w) \cap F \neq \emptyset \iff \bar{\delta}^*(q,w) \cap \bar{F} \neq \emptyset$$

The intended result follows by instantiating this last equivalence for $q = q_0$. If $w = \lambda$ then

$$\begin{aligned}
\delta^*(q,\lambda) \cap F \neq \emptyset &\iff \lambda\text{−closure}(q) \cap F \neq \emptyset \\
&\iff q \in \bar{F} \\
&\iff \{q\} \cap \bar{F} \neq \emptyset \\
&\iff \bar{\delta}^*(q,\lambda) \cap \bar{F} \neq \emptyset
\end{aligned}$$

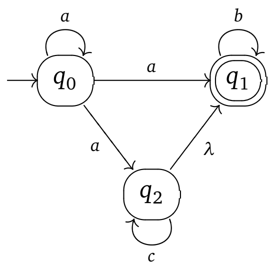
If $w = au$ then

$$\begin{aligned}
\delta^*(q,au) \cap F \neq \emptyset &\iff \left(\bigcup_{q' \in \lambda\text{−closure}(q)} \bigcup_{p \in \delta(q',a)} \delta^*(p,u) \right) \cap F \neq \emptyset \\
&\iff \left(\bigcup_{q' \in \lambda\text{−closure}(q)} \bigcup_{p \in \delta(q',a)} \delta^*(p,u) \cap F \right) \neq \emptyset \\
&\iff \left(\bigcup_{q' \in \lambda\text{−closure}(q)} \bigcup_{p \in \delta(q',a)} \delta^*(p,u) \right) \cap F \neq \emptyset \\
&\overset{\text{IH}}{\iff} \left(\bigcup_{q' \in \lambda\text{−closure}(q)} \bigcup_{p \in \delta(q',a)} \bar{\delta}^*(p,u) \right) \cap F \neq \emptyset \\
&\iff \left(\bigcup_{q' \in \lambda\text{−closure}(q)} \bigcup_{p \in \delta(q',a)} \bar{\delta}^*(p,u) \cap F \right) \neq \emptyset \\
&\iff \left(\bigcup_{p \in \bar{\delta}(q,a)} \bar{\delta}^*(p,u) \right) \cap F \neq \emptyset \\
&\iff \bar{\delta}^*(q,au) \cap F \neq \emptyset
\end{aligned}$$

□

3.3.3 Example (Elimination of λ-transitions).

Let $\mathcal{A}$ be the NFA-λ over $A = \{a, b, c\}$



We have

$$\lambda\text{−closure}(q_0) = \{q_0\} \qquad \lambda\text{−closure}(q_1) = \{q_1\} \qquad \lambda\text{−closure}(q_2) = \{q_2, q_1\}$$

and therefore

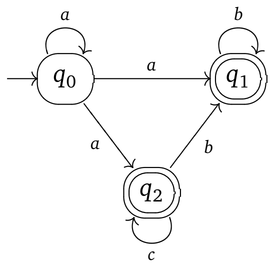
$$\bar{F} = \{q \in Q \mid \lambda\text{−closure}(q) \cap F \neq \emptyset\} = \{q_1, q_2\}$$

and

$$\bar{\delta}(q_2, b) = \bigcup_{q \in \lambda\text{−closure}(q_2)} \delta(q, b) = \delta(q_2, b) \cup \delta(q_1, b) = \{q_1\}$$

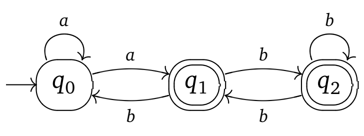
♠

The other $\bar{\delta}(q, a)$ are direct since only q_2 has a non-singleton λ−closure set. The automaton $\bar{\mathcal{A}}$ is:



3.4 Exercises

1. Let $\mathcal{A}$ be the non-deterministic automaton with state diagram



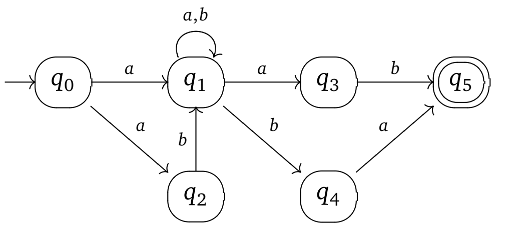
a) Compute $\delta^*(q_0)(aabb)$.

   b) Is $aabb$ accepted by $\mathcal{A}$? Justify your answer using the answer of a).
   c) Give a regular expression that denotes the language accepted by the above automaton $\mathcal{A}$.

2. Construct a deterministic automaton that accepts the language accepted by the non-deterministic automaton of exercise 1.

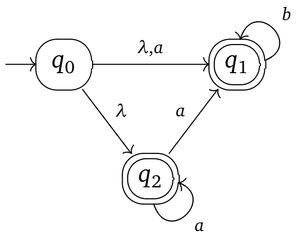
3. Build a non-deterministic finite automaton that accepts the set of words over $\{a, b\}$ that have a subword of length 3 with the same first and last letter.

4. Let $\mathcal{A}$ the non-deterministic automaton over the alphabet $A = \{a, b\}$ given by



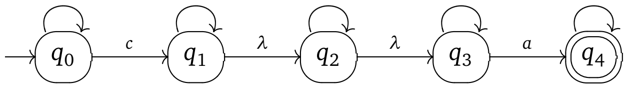
   a) Give an example of a word of length 3 which is accepted.
   b) Compute $\delta^*(q_1)(ab)$, giving the steps explicitly.
   c) Use this information to compute $\delta^*(q_0)(abab)$, again writing down all steps necessary steps.
   d) Is $abab$ accepted by $\mathcal{A}$? Use the above exercise to answer this.
   e) What is the language accepted by $\mathcal{A}$?
   f) Give a regular expression e with $\mathcal{L}(\mathcal{A}) = \mathcal{L}(e)$.

5. Consider the following automaton



   Compute the λ-closure of q_0 and give a regular expression denoting the language accepted by the automaton.

6. Let $\mathcal{A}$ be the NFA over the alphabet $A = \{a, b, c\}$ given by the following picture



a) What is the length of the shortest accepted word? Give an example.

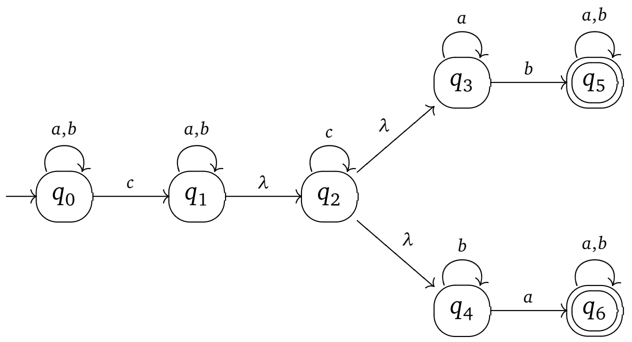
b) Is the word *cacaca* accepted? Why or why not?

c) Compute λ−closure(q_2) and λ−closure(q_1).

d) Construct an NFA which accepts the same language as $\mathcal{A}$ using λ−closure.

e) Construct a DFA which accepts the same language as $\mathcal{A}$ using the determinisation procedure (subset construction). Annotate the states with the set of states of $\mathcal{A}$ they are representing. The resulting automaton should have 7 states.

7. Let $\mathcal{A}$ be the NFA over the alphabet $A = \{a, b, c\}$ given by the following picture



a) What is the length of the shortest accepted word? Give an example.

b) Compute λ−closure(q_2) and λ−closure(q_1).

c) Construct a DFA which accepts the same language as $\mathcal{A}$ using the determinisation procedure from the lecture. Annotate the states with the set of states of $\mathcal{A}$ they are representing. The resulting automaton should have 10 states.

8. Show that any finite set of strings is a regular language.

9. (†) Give an NFA over $A = \{a\}$ that rejects some word, but the length of the shortest rejected word is strictly greater than the number of states.

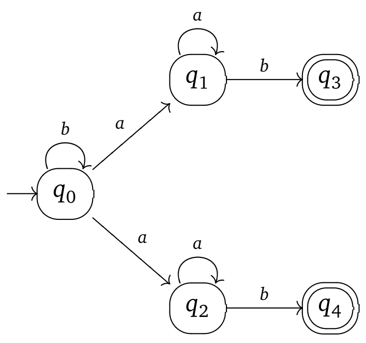
Lecture 4

Kleene's theorem and Properties of Regular Languages

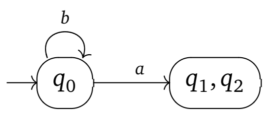
We continue the material of the previous lecture and show that every NFA can be equivalently represented by a DFA. We then present and prove Kleene's theorem that states the equivalence between regular expressions and finite automata. We conclude the lecture by discussing a few closure properties of regular languages.

4.1 Determinization of NFA

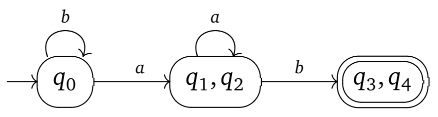
In the previous lecture we showed how to eliminate λ-transitions. In this section, we will show how to eliminate non-determinism and build a DFA that recognizes the same language as a given NFA. This process is known as *determinization*. The intuitive idea is to build an automaton where the non-determinism is hidden/collected in the states. For instance, if we have



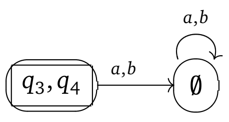
Then we build a new automaton where the first non-deterministic branching is replaced by



The we continue building the automaton by keeping in mind that a state labelled by p, q can move using *both* the transitions of q and q. Moreover, such a state will be final if one of the states is final. In the example above,



The only thing missing to make the above automaton deterministic is who to define the a and b transitions of the state q_3, q_4? Since in the original automaton there were no transitions the behavior is empty. And once empty the behavior remains empty. Hence, we complete the automaton above as follows.



4.1.1 Definition (subset construction). Given an NFA $\mathcal{A} = (Q, \delta, q_0, F)$ we build a DFA $\det(\mathcal{A}) = (Q', \delta', \{q_0\}, F')$ as follows.
Start with $Q' = \{q_0\}$. Build Q' by repeating the following procedure:

- For all $a \in A$, let $X_a = \bigcup_{q \in X} \delta(q, a)$. If $X_a \notin Q'$ then add it: $Q' \leftarrow Q' \cup \{X_a\}$.

Define

- $\delta' : Q' \times A \to Q'$ by $\delta'(X, a) = X_a$.
- $F' = \{X \in Q' \mid X \cap F \neq \emptyset\}$. ♣

We now show that the DFA obtained in the subset construction accepts the same language as the original NFA.

4.1.2 Theorem. *Given an NFA $\mathcal{A} = (Q, \delta, q_0, F)$, the corresponding DFA obtained in the subset construction* $\det(\mathcal{A}) = (Q', \delta', \{q_0\}, F')$ *accepts the same language.*

Proof. We want to show that, for all $w \in A^*$,

$$w \text{ is accepted by } \mathcal{A} \iff w \text{ is accepted by } \det(\mathcal{A})$$

We show a slightly more general result: for all $X \in Q'$

$$\bigcup_{x \in X} \delta^*(x, w) \cap F \neq \emptyset \iff (\delta')^*(X, w) \in F'$$

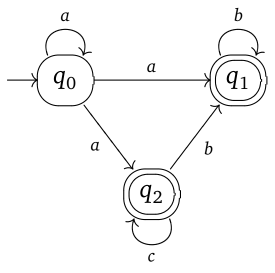
The intended result follows by instantiating this last equivalence for $X = \{q_0\}$. By induction on words. If $w = \lambda$ then

$$\begin{aligned} \bigcup_{x \in X} \delta^*(x, \lambda) \cap F \neq \emptyset &\iff X \cap F \neq \emptyset \\ &\iff X \in F' \\ &\iff (\delta')^*(X, \lambda) \in F' \end{aligned}$$

If $w = au$ then

$$\begin{aligned}
\bigcup_{x\in X} \delta^*(x,au) \cap F \neq \emptyset &\iff \bigcup_{x\in X} \bigcup_{q\in\delta(x,a)} \delta^*(q,u) \cap F \neq \emptyset \\
&\iff \bigcup_{q\in\delta'(X,a)} \delta^*(q,u) \cap F \neq \emptyset \\
&\overset{\text{IH}}{\iff} (\delta')^*(\delta'(X,a),u) \in F' \\
&\iff (\delta')^*(X,au) \in F' \qquad \square
\end{aligned}$$

4.1.3 Example (Subset construction). Consider the NFA $\mathcal{A}$ over $\{a,b,c\}$:



We first build Q', starting from $Q' = \{\{q_0\}\}$.

$$\delta(q_0)(a) = \{q_0,q_1,q_2\} \qquad \delta(q_0)(b) = \delta(q_0)(c) = \emptyset$$

We have two new states and therefore $Q' = \{\{q_0\},\emptyset,\{q_0,q_1,q_2\}\}$. We now inspect the transitions of $\{q_0,q_1,q_2\}$:

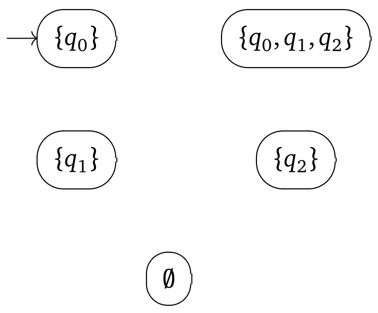
$$\begin{aligned}
\delta(q_0)(a) \cup \delta(q_1)(a) \cup \delta(q_2)(a) = \{q_0,q_1,q_2\} \\
\delta(q_0)(b) \cup \delta(q_1)(b) \cup \delta(q_2)(b) = \{q_1\} \\
\delta(q_0)(c) \cup \delta(q_1)(c) \cup \delta(q_2)(c) = \{q_2\}
\end{aligned}$$

In this step, we encountered two new states, $\{q_1\}$ and $\{q_2\}$, and therefore we extend Q' to $\{\{q_0\},\emptyset,\{q_0,q_1,q_2\},\{q_1\},\{q_2\}\}$. We now inspect the transitions of $\{q_1\}$ and $\{q_2\}$:

$$\delta(q_1)(a) = \delta(q_1)(c) = \emptyset \quad \delta(q_1)(b) = \{q_1\}$$

$$\delta(q_2)(a) = \emptyset \quad \delta(q_2)(b) = \{q_1\} \quad \delta(q_2)(c) = \{q_2\}$$

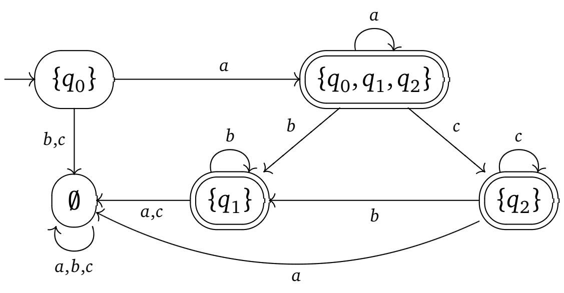
At this stage we have not found any new states and hence we have built Q'. We have a starting point for our automaton:



Now we need to compute final states:

$$F' = \{X \in Q' \mid X \cap F \neq \emptyset\} = \{\{q_0, q_1, q_2\}, \{q_1\}, \{q_2\}\}.$$

Putting everything together, we build the DFA set($\mathcal{A}$):



♠

4.2 Kleene's theorem

We have defined a language to be regular if it is denoted by a regular expression. In this section, we show that the class of languages accepted by DFA is precisely the class of regular languages. Since we also showed in the previous sections that NFA-λ accept the same languages as NFA and these accept exactly the same languages as DFA the following theorem states that all acceptors we introduced so far and regular expressions have the same expressivity in terms of languages – they all accept/denote regular languages.

4.2.1 Theorem. *Let $L \subseteq A^*$ be a language over A. The following are equivalent.*

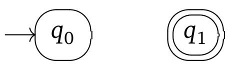
1. *L is regular.*
2. *L is accepted by a deterministic finite automaton.*

The proof of this theorem has two parts, corresponding to each implication. For the first, regular $\Rightarrow$ DFA, we need to build a DFA which accepts the same language as a given regular expression. We do this in Section 4.2.1. For the second, regular $\Leftarrow$ DFA, we need to build a regular expression which denotes the same language as the accepted a given DFA. We do this in Section 4.2.2.

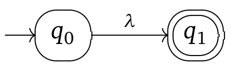
4.2.1 From regular expressions to automata

We have already explained in the previous lecture how to use λ-transitions to combine two automata in order to build an automaton that recognizes the union of the two languages. We now do this systematically for all regular operators. For simplicity, we will use λ-transitions to combine automata that have precisely one final state (this is not a restriction, how can you child an automaton that only has a final state and recognizes the same language as a given NFA(-λ)?).

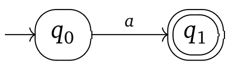
We first define automata for the basic regular constructs:



accepts $\emptyset$



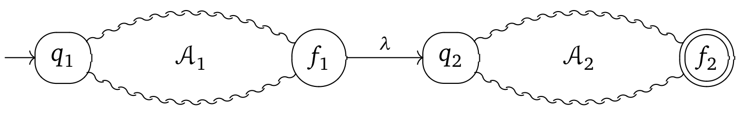
accepts $\{\lambda\}$



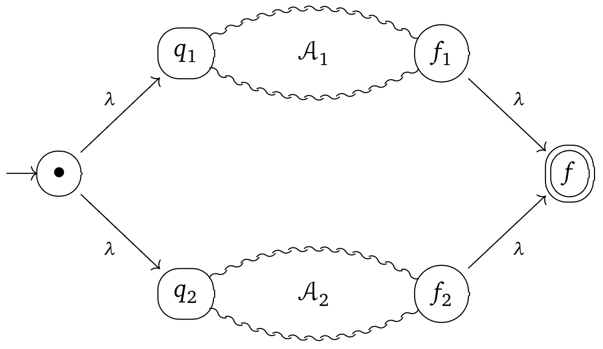
accepts $\{a\}$

Given two NFA-λ $\mathcal{A}_1 = (Q_1, \delta_1, q_1, \{f_1\})$ and $\mathcal{A}_2 = (Q_2, \delta_2, q_2, \{f_2\})$ (with $Q_1 \cap Q_2 = \emptyset$) we build automata for the concatenation, union and star as follows.

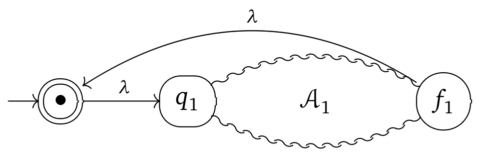
Concatenation. Let $\mathcal{C} = (Q_1 \cup Q_2, \delta, q_1, \{f_2\})$ where $\delta(f_1, \lambda) = \{q_2\}$, $\delta(q, a) = \delta_1(q, a)$, for all $q \in Q_1$ and $\delta(q, a) = \delta_2(q, a)$, for all $q \in Q_2$. The language accepted by $\mathcal{C}$ is precisely $\mathcal{L}(q_1)\mathcal{L}(q_2)$. Diagramatically, $\mathcal{C}$ can be represented as follows:



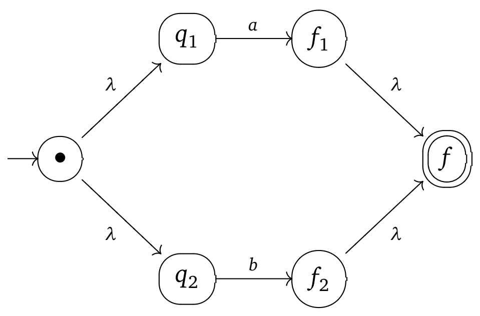
Union. Let $\mathcal{U} = (Q_1 \cup Q_2 \cup \{\bullet, f\}, \delta, \bullet, \{f\})$ where $\delta(f_1, \lambda) = \delta(f_2, \lambda) = \{f\}$, $\delta(\bullet, \lambda) = \{q_1, q_2\}$, $\delta(q, a) = \delta_1(q, a)$, for all $q \in Q_1$ and $\delta(q, a) = \delta_2(q, a)$, for all $q \in Q_2$. The language accepted by $\mathcal{U}$ is precisely $\mathcal{L}(q_1) \cup \mathcal{L}(q_2)$. Diagramatically, $\mathcal{U}$ can be represented as follows:



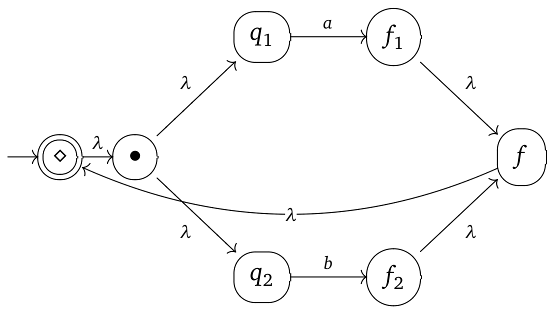
Star. Let $\mathcal{S} = (Q_1 \cup \{\bullet\}, \delta, \bullet, \{\bullet\})$ where $\delta(\bullet, \lambda) = \{q_1\}$, $\delta(f_1, \lambda) = \{\bullet\}$, $\delta(q, a) = \delta_1(q, a)$, for all $q \in Q_1$. The language accepted by $\mathcal{S}$ is precisely $\mathcal{L}(q_1)^*$. Diagramatically, $\mathcal{S}$ can be represented as follows:



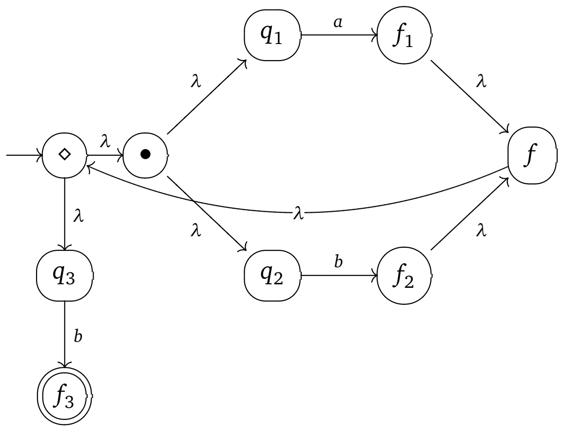
4.2.2 EXAMPLE (From expressions to automata). Consider the expression $(a+b)^*b$. We construct the automaton for $a + b$:



and from this the automaton for $(a + b)^*$:



We can now construct the automaton for $(a+b)^*b$:

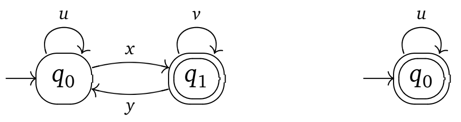


♠

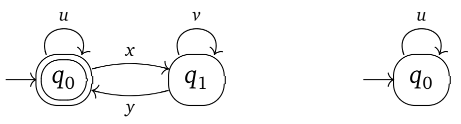
So, we have now showed how to obtain an NFA-λ which accepts the same language as denoted by a given regular expression. In order to build a DFA, we just need to apply the procedures of λ-elimination and determinization presented above.

4.2.2 From automata to regular expressions

We now explain how to build a regular expression denoting the language accepted by an automaton. There are several ways to do this but here we follow the method presented in Sudkamp.
Consider the following two automata:

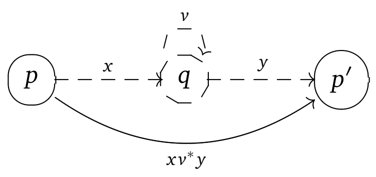


The regular expressions $u^*x(v+yu^*x)^*$ and u^* denote the languages accepted by the above automata (check!). Also, note that if we have

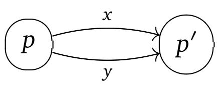


then the regular expressions $(u+xv^*y)^*$ and 0 denote the languages accepted by the above automata (check!)
What we do next is to reduce the problem of finding a regular expression denoting the language of any automaton by gradually reducing the automaton to one of the cases above.
We again assume that the input automaton only has one final state. As above, this is not a restriction.

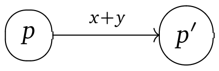
4.2.3 Definition (Two state NFA-λ). Given an NFA-λ $(Q, \delta, q_0, \{f\})$ we transform the automaton into $(\{q_0, f\}, \delta', q_0, \{f\})$ by defining δ' as follows.
For every $q \in Q$ such that $q \neq f$ and $q \neq q_0$ we delete the state q after replacing, for every pair of states $p, p' \in Q$, the dashed transitions below by a new one displayed below:



Note that if the state q does not have a self-loop, that is $v = \lambda$, then $xv^*y = xy$. To ensure that we always only have one transition between each two states we replace multiple transitions



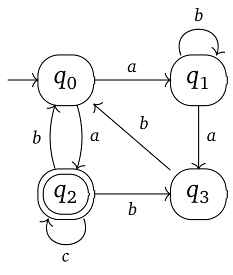
by



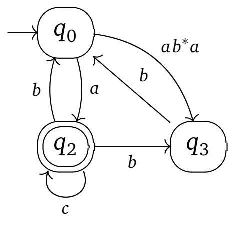
♣

Once we have the new NFA with only two states we can apply the formulae above to derive the regular expression.

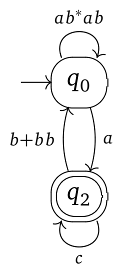
4.2.4 Example. Consider the following NFA:



We first delete q_1:



We then delete q_3:



Hence, the regular expression denoted by the automaton is:

$$(ab^*ab)^*a(c+(b+bb)(ab^*ab)^*a)^*.$$

4.3 Properties of regular languages

Regular languages have some closure properties. For instance, if languages L, M are regular then the reverse L^R, the intersection $L \cap M$ and the complement $\bar{L}$ are also regular. Let us show the last two closure properties.
Let L be a regular language. By Kleene's theorem, we know that there is a DFA (Q, δ, q_0, F) accepting L. Define a new DFA $(Q, \delta, q_0, Q \setminus F)$ and let us show that this new automaton accepts $\bar{L} = \{w \in A^* \mid w \notin L\}$, that is, $w \in \bar{L} \iff \delta^*(q_0, w) \in Q \setminus F$. It is an easy calculation:

$$w \in \bar{L} \iff w \notin L \iff \delta^*(q_0, w) \notin F \iff \delta^*(q_0, w) \in Q \setminus F$$

Now the fact that regular languages are closed under intersection follow by using de Morgan's laws and observing that:

$$L \cap M = \overline{\bar{L} \cup \overline{M}}.$$

Closure properties help in establishing that a certain language is regular. For instance, take the language L over $\{a, b\}$ of all words that contain the substring aba but do not contain the substring bb. To show that this language is regular we observe that

1. $r = (a+b)^*aba(a+b)^*$ is a regular expression for the language over $\{a, b\}$ of all words that contain the substring aba.

2. $s = (a+b)^*bb(a+b)^*$ is a regular expression for the language over $\{a, b\}$ of all words that contain the substring bb.

3. $L = \mathcal{L}(r) \cap \overline{\mathcal{L}(s)}$.

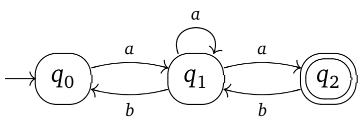
4.4 Exercises

1. a) Build two NFAs that accept b^*a and $(ab^*b)^*$, respectively.

   b) Using λ-transitions combine them into an NFA that accepts $b^*a(ab^*b)^*$.

   c) Using the subset construction build an equivalent DFA.

2. Consider the following automaton



   Compute the regular expression denoting the language accepted by this automaton using the (state-elimination) algorithm we discussed in the lecture.

3. Let L_1 be a non-regular language and L_2 a regular language. Give a concrete instance of L_1 and L_2 such that

   a) $L_1 \cup L_2$ is regular.

   b) $L_1 \cup L_2$ is not regular.

4. Let $r = (a+b)^*ab(a+b)^*$. Find a complement for r over the alphabet $A = \{a, b\}$, that is a regular expressions $\bar{r}$ over the alphabet A satisfying $\mathcal{L}(\bar{r}) = \{w \in A^* \mid w \notin \mathcal{L}(r)\}$.

5. (†)

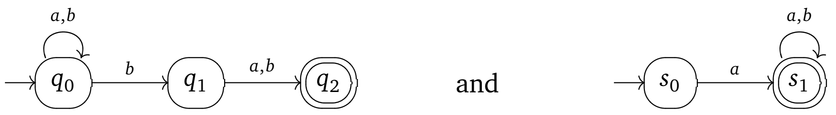
   a) Let L be a regular language. Is $\{ww | w \in L\}$ regular?

   b) Let A and B be alphabets. A string homomorphism is a total function $h\colon A^* \to B^*$ that preserves concatenation. More precisely it satisfies:

$$\begin{aligned} h(\lambda) &= \lambda \\ h(uv) &= h(u)h(v) \end{aligned}$$

   Let $L \subseteq A^*$ be regular. Show that $\{h(w) \mid w \in L\}$ is also regular.

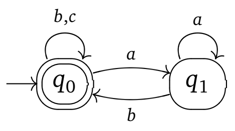
6. a) Let $\mathcal{A}_1, \mathcal{A}_2$ be NFA over the alphabet $A = \{a, b\}$ given by



   Use the construction given in the lecture to obtain an λ-NFA $\mathcal{A}$ which accepts $\mathcal{L}(\mathcal{A}_1) \cup \mathcal{L}(\mathcal{A}_2)$. Describe the resulting language.

b) Let $L_1 = \mathcal{L}(\mathcal{A}_1)$ and $L_2 = \mathcal{L}(\mathcal{A}_2)$ for the above NFA. Construct a λ-NFA $\mathcal{A}$ which accepts exactly the concatenation L_1L_2. Describe again the resulting language.

7. Let A be the alphabet $\{a, b, c\}$ and $\mathcal{A}$ the NFA



a) Which of the words cac, $cabc$ and $caaabc$ are accepted? What is the language the automaton accepts?

b) Construct a regular expression e using the algorithm from the lecture which generates the language $\mathcal{L}(\mathcal{A})$.

8. (†)

a) Using that regular languages are closed under complement and intersection, show that for regular languages L_1, L_2 their difference $L_1 \backslash L_2$ is regular as well.

b) Give an algorithm which decides whether for a regular expression e its language $\mathcal{L}(e)$ is empty. Hint: use deterministic finite automata.

c) Give an algorithm which checks for given regular expressions e_1, e_2 whether their languages are equal: $\mathcal{L}(e_1) = \mathcal{L}(e_2)$. Hint: use the above exercises.

9. (†) Let A be an alphabet and $L_1, L_2 \subseteq A^*$ two regular languages given by NFAs $\mathcal{A}_1, \mathcal{A}_2$ with $\mathcal{L}(\mathcal{A}_i) = L_i$. Show that the concatenation L_1L_2 is regular by constructing a nondeterministic automaton $\mathcal{A}$ from $\mathcal{A}_1, \mathcal{A}_2$ and showing that $\mathcal{L}(\mathcal{A}) = L_1L_2$.

   Hint: use the set $Q = (\{1\} \times Q_1) \cup (\{2\} \times Q_2)$ as states.

10. The algorithm to construct a regular expression from an NFA requires that the automaton has exactly one accepting state. Show how to construct for a given NFA $\mathcal{A}$ another NFA $\mathcal{B}$ with exactly one acceptance state and $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{B})$.

Lecture 5

Beyond regular languages.

In the previous lectures, we have studied regular languages both from a denotational perspective (regular expressions) and an operational perspective (automata). In the coming lectures, we are going to move up in the Chomsky hierarchy of formal languages and we are going to study another, more expressive, class: context-free languages. But before we do, we will show a method to prove that a given language is not regular.

5.1 How to prove that a language is not regular?

Consider the language over alphabet $\{a, b\}$

$$L = \{a^n b^n \mid n \in \mathbb{N}\}.$$

Using Kleene's theorem, we know that if L is regular then there exists a finite automaton accepting L. Hence, a method to disprove that L is not regular is to show that there does not exist a finite automaton accepting it.
Suppose there would exist a finite automaton (Q, q_0, δ, F), with $Q = \{q_0, \cdots, q_m\}$, accepting L. Then, for any $i, j \in \mathbb{N}$ such that $i \neq j$ we know that

$$a^i b^i \in L \text{ and } a^j b^i \notin L$$

That is, $\delta^*(q_0, a^i b^i) \neq \delta^*(q_0, a^j b^i)$. But we have

$$\delta^*(q_0, a^i b^i) = \delta^*(\delta^*(q_0, a^i), b^i) \text{ and } \delta^*(q_0, a^j b^i) = \delta^*(\delta^*(q_0, a^j), b^i)$$

Hence, we must have that $\delta^*(q_0, a^i) \neq \delta^*(q_0, a^j)$ for all $i, j \in \mathbb{N}$ which implies that Q is infinite. We can therefore conclude that L is not regular.
What one can observe in the above proof is that in order to build an automaton to accept L we need states q_i that *memorize* the number of *a*ś that have been read so far. In the next lecture, we will introduce a new type of automaton which has the capability to *memorize*, thereby increasing the expressivity in relation to DFA/NFA.

Another way to prove that a language is not regular is given by the *pumping lemma* for regular languages, which requires words in a regular language to satisfy certain decomposition properties.
Pumping a word refers to build new words by repeating sub-words in the original word. In a regular language, accepted by a finite automaton, if one increases the

length of an accepted word enough, the path going through the automaton will have to repeatedly go through the same states (since we have a finite number of them). For instance, consider the automaton

$$\begin{tikzcd} {} \arrow[r] & q_0 \arrow[r, "a"] \arrow[loop above, "c"] & q_1 \arrow[r, "a"] \arrow[loop above, "c"] & q_2 \arrow[loop above, "c"] \arrow[ll, bend left, "a"] \end{tikzcd}$$

and the word $w = ccaaac$. We can divide w into $u = cc$, $v = aaa$ and $z = c$. *Pumping* subword aaa means building words uv^iz – $ccaaac, ccaaaaaac, \ldots$. All these words are accepted by the above automaton.

5.1.1 Lemma (Pumping lemma for regular languages). *Let L be a regular language accepted by a DFA with k states. Let w be any word in L with $|w| \geq k$. Then, w can be written as uvz with $|uv| \leq k$, $|v| > 0$ and $uv^iz \in L$ for all $i \geq 0$.*

The pumping lemma is a powerful took to prove that a language is not regular. Every string of length k must have an appropriate decomposition. To show that a language is not regular, it is enough to find one word that does not satisfy the conditions in the lemma. That is, find a word w and show that there is no decomposition $w = uvz$ for which every $uv^iz \in L$, for all $i \geq 0$.

5.1.2 Example (Pumping Lemma).

1. Let us consider the example of the beginning of the lecture $\{a^n b^n \mid n \in \mathbb{N}\}$. Consider a word $a^i b^i \in L$. We can decompose it in three different ways

$$a^j \cdot a^m \cdot a^{i-m-j} b^i \qquad a^i b^k \cdot b^m \cdot b^{i-m-k} \qquad a^j \cdot a^{i-j} b^k \cdot b^{i-k}$$

   In the last case if we pump substring $a^{i-j}b^k$ we end up in a word that has a's and b's mixed: hence not in L. The other two cases are symmetric, we argue the first. Pump substring a^m twice. This generates word

$$a^j \cdot a^{2m} \cdot a^{i-m-j} b^i$$

   which has $j + 2m + i - m - j = i + m$ a's and i b's. Since $i \neq i + m$ the word is not in L. Hence, for all decompositions $w = uvz$ we have that $uv^2z \notin L$ and therefore, using the Pumping Lemma, we conclude that the language is not regular. (There was a simpler string we could have considered, namely $a^k b^k$ for k the number of states of the DFA accepting L. why? how many decompositions are then?).

2. Consider the language $\{w \in \{a\}^* \mid |w| = k^2 \text{ for some } k \in \mathbb{N}\}$. Assume that this language is regular and let k be the number of states in the DFA that accepts it. Now, take the word $w = a^{k^2} \in L$. We can divide this string as $w = uvz = a^p a^q a^r$ with $q > 0$ and $p + q + r = k^2$. But now observe that uv^2w has the length:

$$|uv^2w| = |uvw| + |v| = k^2 + q \leq k^2 + k < k^2 + 2k + 1 = (k+1)^2$$

so we have $k^2 < |uv^2w| < (k+1)^2$ which shows that it cannot be a perfect square and hence $uv^2w \notin L$. We can then conclude that L is not regular. ♠

5.2 Context-free languages

The language $\{a^n b^n \mid n \in \mathbb{N}\}$ which we considered above is not regular but it belongs to another class of well-studied languages: context-free languages.
We will now introduce a formal system, context-free grammars, which is used to generate strings of a language.

5.2.1 Definition (Context-free grammar). A context-free grammar is a quadruple (V, Σ, P, S) where

- V is a finite set of non-terminal symbols.
- Σ is the alphabet (finite set of terminal symbols).
- P is a finite set of rules. A rule is written $A \to w$, where $A \in V$ and $w \in (V \cup \Sigma)^*$.
- $S \in V$ is the start symbol. ♣

We use capital letters $A, B, C, \ldots$ to denote non-terminal symbols and lower-case $a, b, c, \ldots$ to denote elements of Σ.

5.2.2 Example (Context-free Grammar). Let $V = \{S, B\}$, let $\Sigma = \{a, b\}$, and let P be the set with the following two rules.

$$S \to aSB \mid \lambda \qquad B \to b$$

Note that λ is always a word in $(V \cup \Sigma)^*$ and hence can always be used on the right side of a production rule. ♠

Grammars are used to generate/derive words as follows. Given a rule $A \to w$ and a string uAv we can apply the rule to this string to generate uwv and we write

$$uAv \to uwv$$

The grammars we study in this course are called context-free because rules $A \to w$ can be applied in any context to replace the non-terminal symbol A.
Given $u, v \in (V \cup \Sigma)^*$, we say that u can be derived from v if there is a sequence of applications of the production rules

$$v \to v' \to v'' \to \cdots \to u$$

and we write $v \Rightarrow u$.

5.2.3 DEFINITION (Language generated/derived from a context-free grammar). Let $G = (V, \Sigma, P, S)$ be a context-free grammar. The language generated (or derived) by the context-free grammar G is given by

$$\mathcal{L}(G) = \{w \in \Sigma^* \mid S \Rightarrow w\}$$ ♣

We use capital letters $A, B, C, \ldots$ to denote non-terminal symbols and lower-case $a, b, c, \ldots$ to denote elements of Σ.

5.2.4 EXAMPLE (Language generated). Recall the grammar above

$$S \to aSB \mid \lambda \qquad B \to b$$

The word $aabb$ is in the language generated by the grammar:

$$S \to aSB \to aaSBB \to aaBB \to aabB \to aabb$$

In fact, the grammar above generates precisely the language $\{a^n b^n \mid n \in \mathbb{N}\}$ which we have been using as running example. ♠

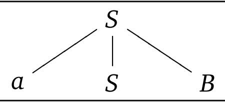
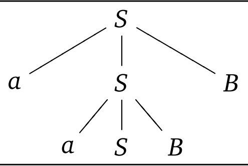
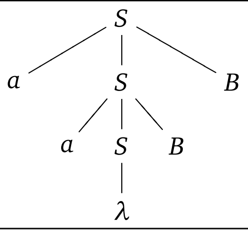
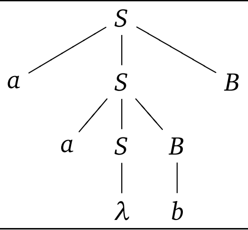
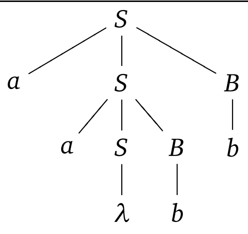
The application of rules, when showing that a word is generated by a grammar, does not need to follow a particular order. For instance, the word $aabb$ of the previous example could have been generated applying the rules in a different order. Here are three alternative examples of derivations, including the one above:

$$S \to aSB \to aaSBB \to aaBB \to aabB \to aabb$$

$$S \to aSB \to aSb \to aaSBb \to aaSbb \to aabb$$

$$S \to aSB \to aSb \to aaSBb \to aaBb \to aabb$$

The first derivation above was obtained by always replacing the non-terminal symbol appearing first in a left-to-right reading of the word. Such derivations are called *leftmost*. The second derivation is *rightmost* because we always replaced the rightmost symbol first.
Derivations can be depicted as trees by initializing the tree with S as root and then for each application of rule $A \to x_1 x_2 \ldots x_n$ add $x_1, \ldots, x_n$ as children of the leaf A. We illustrate this with the example above.

Derivation steps	Corresponding tree
S	S
$\rightarrow aSB$	
$\rightarrow aaSBB$	
$\rightarrow aaBB$	
$\rightarrow aabB$	
$\rightarrow aabb$	

Leftmost derivations correspond to replacing the leftmost leaf of the tree. The word derived is simply obtained by reading the leaves of the tree in a left-to-right order. When a word can be derived using two leftmost derivations in the same grammar we say that the grammar is *ambiguous*. For instance, consider the grammar

$$S \rightarrow aS \mid Sa \mid a$$

that generates a^+. The grammar is ambiguous because aa can be obtained using two different leftmost derivations

$$S \to aS \to aa \qquad\qquad S \to Sa \to aa$$

The language generated by the above grammar can also be generated by an unambiguous grammar, namely

$$S \to aS \mid a$$

This example show that ambiguity is a property of grammars not languages. When a grammar is shown to be ambiguous it is often possible to build an equivalent unambiguous grammar. However, this is not always possible. Context-free languages that are only generated by an ambiguous grammar are called *inherently ambiguous*.

5.3 Exercises

1. Using the pumping lemma, show that the following languages are not regular
   a) $\{a^n b^m \mid n = 2m\}$
   b) $\{x \in \{a,b,c\}^* \mid x$ is a palindrome (that is, $x = x^R$)$\}$
   c) $\{x \in \{a,b\}^* \mid$ the number of a's is a cube$\}$
   d) $\{xx \mid x \in \{a,b\}^*\}$
   e) $\{x \in \{a,b\}^* \mid \; |x|_a = |x|_b\}$
2. Let L_1 be a non-regular language and L_2 a finite language.
   a) Show that $L_1 \cup L_2$ and $L_1 - L_2$ are not regular.
   b) If L_2 is not finite, are $L_1 \cup L_2$ and $L_1 - L_2$ not regular?
3. Let A be the alphabet $A = \{a,b\}$. Show that the language $L_1 = \{w \in A^* \mid |w|_a = 2|w|_b\}$ is context-free by giving a context free grammar that generates it. Give the derivation of the word $abbaa$.
4. Consider the following grammar

   $$S \to aS \mid Sb \mid ab \mid SS$$

   a) Give a regular expression for the language denoted by the above grammar.
   b) Construct the two leftmost derivations for the string $aabb$ and build the respective derivation trees.
5. Show that the above grammar is ambiguous and give an equivalent unambigous grammar.
6. Let G be the CFG given by the derivation rules

   $$S \longrightarrow aS \mid SS \mid b$$

   and the non-terminal S being the start symbol.

a) Show that $ababb \in \mathcal{L}(G)$ by giving a *left-most* derivation.
b) Turn the above derivation into a derivation tree.
c) Find another left-most derivation that leads to another derivation tree.

7. Give a context free grammar generating the language of "balanced parenthesis" D. Show that contains the words λ, (), ()(), (()()).

8. Let $L = \{a^n b^k a^m \mid k = n + m\} \subseteq A^*$.

   a) Construct a CFG G such that $\mathcal{L}(G) = L$.
   b) Give a derivation for the word $aabbba \in L$.

9. (†) The operation of shuffle is important in the context of concurrency theory. If $u, v \in A^*$, we write $u \mid v$ for the set of all strings that can be obtained from u and v by shuffling them like a deck of cards; for instance

$$ab \mid cd = \{abcd, cdab, acdb, cadb, cdab\}$$

The set $u \mid v$ can formally be defined by induction:

$$\lambda \mid v = \{y\} \qquad u \mid \lambda = \{u\} \qquad ua \mid vb = (u \mid vb) \cdot \{a\} \cup (ua \mid v) \cdot \{b\}$$

The operation $\cdot$ is language concatenation. The shuffle f two languages is then defined as

$$X \mid Y = \{(x \mid y) \mid x \in X, y \in Y\}$$

For example,

$$\{ab\} \mid \{cd, e\} = \{abe, acb, eab, abcd, acbd, acdb, cabd, cadb, cdab\}$$

   a) What is $(01)^* \mid (10)^*$?
   b) Show that if X and Y are regular, so is $X \mid Y$.

Lecture 6

Automata with memory.

In this lecture, we introduce pushdown automata, an automaton model with *memory* that accepts context-free languages.

6.1 Pushdown automata.

Recall from the previous lecture that the language

$$\{a^n b^n \mid n \in \mathbb{N}\}$$

was not accepted by a finite automaton because, intuitively, the automaton had no way to remember how many a's it had already read using only finite states.
In this lecture we study an automaton model that combines finite automata with an external memory. The memory we consider is a *stack memory*, a last-in first-out memory. Writing the string $DCBA$ to an empty stack results in the following:

$$\ldots DCBA \rightarrow \begin{array}{|c|}\hline D\\ \hline C\\ \hline B\\ \hline A\\ \hline\end{array} \rightarrow ABCD$$

When reading from the stack the first element will be D, followed by C, B and A, in the reverse order in which they were written into the stack.
The stack has 2 operations: *push* and *pop*. Push takes an element and puts it in the top of the stack.

$$push(Z, \begin{array}{|c|}\hline D\\ \hline C\\ \hline B\\ \hline A\\ \hline\end{array}) = \begin{array}{|c|}\hline Z\\ \hline D\\ \hline C\\ \hline B\\ \hline A\\ \hline\end{array}$$

Pop removes the top element of the stack.

$$pop(\begin{array}{|c|}\hline D\\ \hline C\\ \hline B\\ \hline A\\ \hline\end{array}) = \begin{array}{|c|}\hline C\\ \hline B\\ \hline A\\ \hline\end{array}$$

The operations pop and push only affect the top element of the stack.
We are now ready to define pushdown automata.

6.1.1 DEFINITION (Pushdown automaton). A pushdown automaton (PDA) is a tuple $(Q, \Sigma, \Gamma, \delta, q_0, F)$ where

- Q is a set of states (finite);
- Σ is the input alphabet;
- Γ is the stack alphabet;
- q_0 is the initial state;
- $F \subseteq Q$ is the set of final states;
- $\delta : Q \to \mathcal{P}(Q \times \Gamma_\lambda)^{\Sigma_\lambda \times \Gamma_\lambda}$ is the transition function.

Here, we use Γ_λ and Σ_λ as shorthands for $\Gamma \cup \{\lambda\}$ and $\Sigma \cup \{\lambda\}$, respectively. ♣

We will use the notation

$$\begin{tikzcd} \boxed{p} \arrow[r, "{a,\ A/B}"] & \boxed{q} \end{tikzcd} \iff (q,B) \in \delta(p)(a,A)$$

The differences of PDA with the automata we defined in previous lectures:

- two alphabets: one for the stack, one for input.
- to make a transition a PDA uses: the current control state $q \in Q$, the input symbol, and the symbol on top of the stack.
- $\delta(p)(a,A) = \{(q,B),(r,C)\}$ indicates two possible transitions and two possible changes in the stack.
  $$(q,B) \in \delta(p)(a,A)$$
  causes
  - change of state from p to q;
  - process the input symbol a;
  - pop symbol A from the stack (so: if the stack top is different from A this transition cannot be taken);
  - push symbol B into the stack.
- If $(q,B) \in \delta(p)(a,\lambda)$ this means that the transition can be taken no matter what the content of the stack is and only three actions occur:
  - change of state from p to q;
  - process the input symbol a;

- push symbol B into the stack.

When depicting computations in a PDA we will represent the stack as a string in Γ^*. The empty stack will be denoted by the empty word λ; the word $A\alpha$ denotes a stack with A on top.
The computation of a PDA begins at the initial state with the empty stack. Then the input word is read causing changes to the stack as described above.

6.1.2 EXAMPLE (Computation in a PDA). Consider the following PDA over $\Sigma = \{a, b\}$ and $\Gamma = \{A, B\}$

$$\begin{tikzcd}
\boxed{q_0} \arrow[loop above, "{a,\lambda/A}"] \arrow[r, "{b,A/\lambda}"] & \boxed{\boxed{q_1}} \arrow[loop above, "{b,A/\lambda}"]
\end{tikzcd}$$

In input word $aabb$ the computation steps will be as follows:

$$q_0, aabb, \lambda \longrightarrow q_0, \cancel{a}abb, A \longrightarrow q_0, \cancel{aa}bb, AA \longrightarrow q_0, \cancel{aab}b, A \longrightarrow q_0, \cancel{aabb}, \lambda \qquad \spadesuit$$

We will use the notation $p, uv, \alpha \Longrightarrow q, v, \beta$ to denote that the PDA moved from state p to q, through several steps of computation, by reading word u and with the stack α at the start of the computation (which is then changed to β).

6.1.3 DEFINITION (Language accepted by a pushdown automaton). Given a pushdown automaton $(Q, \Sigma, \Gamma, \delta, q_0, F)$ the language accepted by a state $q \in Q$ is given by

$$\mathcal{L}(q) = \{w \in \Sigma^* \mid q, w, \lambda \Longrightarrow q', \lambda, \lambda \text{ and } q' \in F\} \qquad \clubsuit$$

This definition is known as acceptance by empty stack and final states. In textbooks, you will find alternative definitions where PDA are allowed to accept a word based on only final states or only empty stack. All these definitions are equivalent and in this course we consider acceptance by both empty stack and final states.

6.1.4 EXAMPLE (Language accepted by a PDA). Consider the language $\{wcw^r \mid w \in \{a, b\}^*\}$ over alphabet $\Sigma = \{a, b, c\}$.
Take $\Gamma = \{A, B\}$ and F and δ as depicted:

$$\begin{tikzcd}
\boxed{q_0} \arrow[loop above, "{b,\lambda/B}"] \arrow[loop below, "{a,\lambda/A}"] \arrow[r, "{c,\lambda/\lambda}"] & \boxed{\boxed{q_1}} \arrow[loop above, "{b,B/\lambda}"] \arrow[loop below, "{a,A/\lambda}"]
\end{tikzcd}$$

A successful computation in this PDA: memorizes w, then reads c. At this point the stack contains precisely (an encoding of) the word w^R and the automaton just reads the input string until the stack is empty. For example, the word $abcba$ would generate computations

$$q_0, abcba, \lambda \longrightarrow q_0, bcba, A \longrightarrow q_0, cba, BA \longrightarrow q_1, ba, BA \longrightarrow q_1, a, A \longrightarrow q_1, \lambda, \lambda$$

State q_1 is final and hence $abcba$ is accepted.
Take now word $abcab$:

$$q_0, abcab, \lambda \longrightarrow q_0, bcab, A \longrightarrow q_0, cab, BA \longrightarrow q_1, ab, BA \longrightarrow XX$$

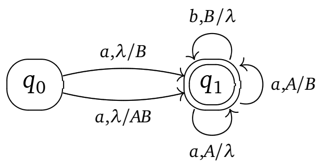
Now, the automaton is stuck because there is a B on top of the stack but the next input letter is a and the only transition in the automaton for a requires an A on top of the stack. Hence, $abcab$ is not accepted. ♠

6.2 From grammars to automata.

In this section, we show how to build a PDA from a given grammar. For simplicity, we consider grammars in *Greibach normal form*. A grammar is in Greibach normal form if every production rule is of the form $A \rightarrow \lambda$ or $A \rightarrow aA_1 \cdots A_n$, with $a \in \Sigma$ and $A_1, \cdots, A_n \in V$. Every grammar can be converted into an equivalent grammar in Greibach normal form (how?) so this is not a restriction.
Take the grammar

$$\begin{array}{l} S \rightarrow aAB \mid aB \\ A \rightarrow aB \mid a \\ B \rightarrow b \end{array}$$

The following automaton accepts the same language:



Note that we are using a shorthand: we have a push of a string *AB* instead of two transitions. Why is this not a problem?
The above automaton was constructed using the following definition.

6.2.1 Definition (Construction of a PDA from a grammar). Given a grammar (V, Σ, P, S), we build a PDA with $(\{q_0, q_1\}, \Sigma, \Gamma, \delta, \{q_1\})$ with

$$\begin{array}{l} \delta(q_0, a, \lambda) = \{(q_1, w) \mid S \rightarrow aw \in P\} \\ \delta(q_0, \lambda, \lambda) = \{(q_1, \lambda) \mid S \rightarrow \lambda \in P\} \\ \delta(q_1, a, A) = \{(q_1, w) \mid A \rightarrow aw \in P\} \end{array}$$ ♣

Note that the last rule can be instantiated for λ: $\delta(q_1, \lambda, A) = \{(q_1, \lambda) \mid A \rightarrow \lambda \in P\}$.

The converse construction, of a CFG from a PDA, exists but is more verbose and we will skip it in this course (for those who are curious, check pages 235-237 of the Sudkamp book).

6.3 Closure properties of CFL

Context free languages are closed under union, concatenation and Kleene star. Let us show the first, the other two are left as exercises.
Suppose that L and M are context free languages. Then there are CFG's $G_1 = (V_1, \Sigma_1, P_1, S_1)$ and $G_2 = (V_2, \Sigma_2, P_2, S_2)$ generating L_1 and L_2, respectively. Define $G = (V_1 \cup V_2 \cup \{S\}, \Sigma_1 \cup \Sigma_2, P_1 \cup P_2 \cup \{S \to S_1 \mid S_2\}, S)$, assuming S is a symbol not present in the given grammars. It is now easy to see that $\mathcal{L}(G) = \mathcal{L}(G_1) \cup \mathcal{L}(G_2)$.
Context free languages are not closed under intersection. For instance,

$$L_1 = \{a^i b^i c^j \mid i, j \geq 0\} \qquad \text{and} \qquad L_2 = \{a^j b^i c^i \mid i, j \geq 0\}$$

are context free (what are the grammars?) but

$$L_1 \cap L_2 = \{a^i b^i c^i \mid i, j \geq 0\}$$

is not (try!).
Context free languages are also not closed under complement. If they were then, given context free languages L_1 and L_2 we would be able to prove that $L_1 \cap L_2 = \overline{\overline{L_1} \cup \overline{L_2}}$ also is context free, but we just saw a counter-example of this fact.

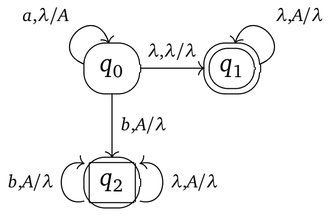
6.4 Wrap-up

In this course we studied two major classes of languages from a denotational (syntactic) and operational perspective (automata). The table below summarizes the material.

	Regular Languages	**Context-Free Languages**
Syntax	Regular expressions	Context-Free Grammars
Operational	DFA, NFA, NFA-λ	PDA
Closure properties	intersection, complement, ...	union, concatenation and star
Connecting syntax and automata	Kleene's theorem	CFG $\to$ PDA

6.5 Exercises

1. Consider the following PDA:



   a) Describe the language accepted by the above PDA.
   b) Describe all computations of the input strings aab and abb.

2. Construct a PDA accepting the language $L_2 = \{w \in \{a,b\}^* \mid |w|_a = |w|_b\}$. Show that the word $abba$ is accepted.

3. Construct a PDA accepting the language $L_1 = \{w \in A^* \mid |w|_a = 2|w|_b\}$. Show that $abbaaa$ is accepted.

4. Construct PDA's that accept the following languages
   a) $\{a^n b^m \mid m = 2n\}$
   b) $\{x \in \{a,b\}^* \mid x$ is a palindrome (that is, $x = x^R$)$\}$
   c) $\{x \in \{a,b\}^* \mid x$ has at least twice as many a's as b's$\}$

5. Let L be the language $\{w \in a, b^* \mid w$ has a prefix containing more b's than a's$\}$. For example, $baa, abba, abbaaa \in L$ but $aab, aabbab \notin L$. Construct a PDA that accepts L.

6. (†) Construct a PDA where the stack alphabet only has two symbols that recognizes the language over $\{a,b,c,d\}$:

$$\{wdw^R \mid w \in \{a,b,c\}^*\}$$

7. Construct (two-state) PDA's for the following CFG
   a)
$$\begin{aligned} S &\to aABA \mid aBB \\ A &\to bA \mid b \\ B &\to cB \mid c \end{aligned}$$
   b)
   c)
$$\begin{aligned} S &\to AB \\ A &\to aAC \mid \lambda \\ B &\to cB \mid \lambda \\ C &\to b \end{aligned}$$

8. (†) Show that for a given context free grammar G, there is a grammar G' with $\mathcal{L}(G') = \mathcal{L}(G)^*$.

9. (†) Let M be the regular language given by $(a+b)^*a(a+b)$ over $A = \{a,b\}$. Show that $L_2 \cap M$ is context-free. Hint: given the NFA for M and the PDA for L_2, one can construct a PDA accepting the intersection, just like in the case for NFA/DFA.

Lecture A

Extra exercises on induction

A.1 Exercise I

The purpose of this exercise is to explain in detail the basics of induction on sets. If you do not know how to answer the questions below right away, we strongly recommend to do this exercise. Induction is a very basic yet powerful technique. You will see it over and over again in this course and everywhere in Computer Science (and of course in Mathematics for that matter). So it is worthwhile to familiarise yourself with it. This can be only achieved by practising, which is the purpose of this exercise. So let us begin with the general scheme of defining a set by induction. Assume we want to define a set X by some inductive property. Then its definition has to contain a base case, which we will refer to by B_X, an induction step S_X and a *closure clause* C_X which states that nothing besides what is given by B_X and S_X is in X. Often this closure clause is left implicit. We will make it explicit in this exercise for the sake of clarity.
For this first exercise, we include solutions. Try to solve the exercise before reading the solutions. After solving and understanding the first exercise you should be able to solve the second exercise provided in this appendix (for which we do not provide solutions).

A.1.1 EXAMPLE (Natural numbers). Natural numbers are the prime example for doing induction and often are mistaken to be the *only* set on which one can do induction. Even though this is not true, we still begin by giving a definition of natural numbers using the above described principle. A natural number will in this case be either 0 or sn where n is already a natural number (pronounce s as successor). Both 0 and s are just letters in the alphabet $\{0, s\}$. So here is the official definition of a set N which we might call natural numbers. The set $N \subseteq \{0, s\}^*$ is defined by

B_N) the symbol 0 is an element of N.

S_N) for any n in N the word sn is in N.

C_N) nothing else is an element of N. ♠

A.1.2 EXAMPLE (Non-empty words). The first example is close to what you already have seen in the lecture: we define the set A^+ of non-empty words for an alphabet A. The set A^+ is defined by

B_{A^+}) all elements of A are in A^+: $A \subseteq A^+$.

S_{A^+}) for any $w \in A^+$ we have that $wa \in A^+$ for all $a \in A$.

C_{A^+}) nothing else is an element of A^+. ♠

Let A be the alphabet $A = \{x, y\}$.

a) What is the minimal length of the words in A^+? Give an example.

Solution: The minimal length of a word $w \in A^+$ is 1, for example $x \in A^+$.

b) Is there a maximum length of words in A^+?.

Solution: Of course not, the induction step always increases the word length. So there is no upper bound.

Given an inductively defined set X, the next step we want to take, is to check whether a given element x is in X. This is done by constructing a finite number (possibly zero) of applications of S_X to an element defined in the base case.

A.1.3 Example. We may wonder, whether the number 2 represented by $ss0$ is a natural number in N. That it is, one can be see by starting from $0 \in N$ and then applying the induction step S_N twice: $0 \xrightarrow{S_N} s0 \xrightarrow{S_N} ss0$. ♠

c) Show how xx is an element of A^+.

Solution: By the base case $x \in A \subseteq A^+$ and by applying the induction step once, we have $xx \in A^+$.

On the contrary one sometimes also would like to show that something is *not* contained in X. To do this, we need to give an argument why the element is not given by the base case and cannot be constructed by the induction step.

A.1.4 Example. Take for example the word $s0s \in \{0, s\}^*$. It should not be an element of N. First it is not covered by the base case, since $s0s \neq 0$. To be constructed in the induction step, it must be of the form, sw with $w \in N$. But then $w = 0s$, for which we prove by a second induction that it cannot be in N. To this end we observe that again $0s \neq 0$ and also that $0s \neq sw$ for any $w \in N \subseteq \{0, s\}^*$. Thus $w \notin N$ and hence $s0s = sw \notin N$. ♠

d) Show that $\lambda \notin A^+$.

Solution: λ is not covered by the base case, since $\lambda \notin A$. In the induction step we find that $\lambda \neq wa$ for any $w \in A^+$, hence $\lambda \notin A^+$.

More generally, a property on an inductively defined set can be proven by the induction principle which comes with every inductively defined set. Assume we want to prove that a property $P(x)$ holds for all $x \in X$. Then the induction principle reduces this problem to showing that

B_X) $P(x)$ holds for all $x \in X$ defined in the base case and

S_X) $P(x')$ holds for all x' constructed in the induction step, using the assumption that $P(x)$ holds for all $x \in X$.

Please note here, that this is different from assuming that $P(x)$ holds for all $x \in X$. Let us write $x \in B_X$ if x is defined by the base case and $x' \in S_X(x)$ if x' is constructed from some $x \in X$ using the induction step (e.g. $0 \in B_N$ and $ss0 \in S_N(s0)$). Then it might be helpful to see the induction principle as a formula:

$$\big((\forall y \in B_X{:}P(y)) \wedge (\forall z \in X{:}P(z) \Rightarrow (\forall z' \in S_X(z){:}P(z'))\big) \Rightarrow \forall x \in X{:}P(x).$$

This says: if $P(y)$ holds for all y in the base case and if $P(z')$ holds for all z' constructed in the induction step from any $z \in X$ with the assumption that $P(z)$ holds, then $P(x)$ holds for all $x \in X$. This is a scary formula indeed, but since natural language has quite some imprecision, this can be clearer. But let us look at a concrete example.

A.1.5 Example. In the case of N we recover the usual principle of induction on the natural numbers (sometimes called “mathematical induction”). A property P on N is thus proven by

B_N) showing that $P(0)$ holds and

S_N) showing that $P(sn)$ holds, provided $P(n)$ holds for all $n \in N$ (this assumption is called *induction hypothesis*).

The logical formula then becomes

$$(P(0) \wedge (\forall n \in N{:}P(n) \Rightarrow P(sn))) \Rightarrow \forall n \in N{:}P(n)$$

which should look a little less scary.
Note that above we have proven that the property $P(n) \Leftrightarrow n \neq s0s$ holds for all $n \in N$. ♠

e) Define the property P you have shown in item d).

Solution: Since $\lambda \notin A^+$ means, that every $w \in A^+$ is not equal to λ, we can define P by $P(w) \Leftrightarrow w \neq \lambda$.

Remark. Sometimes it is easier to see a property as a set P where we say, that the property P holds for X if $P \subseteq X$. In this context P is often called a *predicate*.

A.2 Exercise II

We come now to a more realistic example of defining a set and proving properties about it using induction. In the following we fix alphabets $A = \{a, b, c\}$ and $B = A \cup \{\_\}$ (yes, an underscore). We will define two languages over B by induction and your task will be to show that they are equal. One of the descriptions is in a way which is easy to capture by a regular expression or a DFA.
We start with the language $L \subseteq B^*$ defined by

B_L) the letters a, b, c are in L, i.e. $A \subseteq L$.

S_L) if $w \in L$ then

- i) for all $x \in A$ the word wx is in L, i.e. $wA \subseteq L$.
- ii) for all $x \in A$ the word w_x is in L, i.e. $w\_A \subseteq L$.

C_L) Nothing else is in L.

a) Is $\lambda \in L$? Give a reason.

b) What is the length of the shortest word in L? Give an example.

c) Decide which of the following words is in L. Give a construction or show why it is not in L.

- i) abc
- ii) ab_bc
- iii) $a\_b\_$.

Now it is up to you to define another language K and investigate properties of it.

d) Give a formal definition (by giving the base case, induction step and closure clause) of the following informally defined language K. The language $K \subseteq B^*$ consists of non-empty words over A and for any word w in K of words w_v for all non-empty words v over A, i.e. a non-empty word followed by a sequence of $\_v$ with v being non-empty.

e) Decide which of the following words is in K. Give construction or show why it is not in K.

- i) The empty word λ.
- ii) abc
- iii) ab_bc
- iv) $a\_b\_$.

Now you are going to show that $L = K$ by showing that $L \subseteq K$ and $K \subseteq L$. This will of course be carried out by induction and is more interesting than what we have done so far.

f) Show that $L \subseteq K$. Use induction on L to show that every word $w \in L$ is contained in K which in turn uses the inductive definition of K.

g) Show that $K \subseteq L$ by induction. For this you need to construct a word w from K in L using the inductive definition of L.

Now this representation is actually very close to a regular expression and a deterministic finite automaton. So we close here with the following exercises.

a) Give a regular expression e with $\mathcal{L}(e) = K$.

b) Give a DFA which accepts the language K.

Tests/Exams of previous years

Talen en Automata

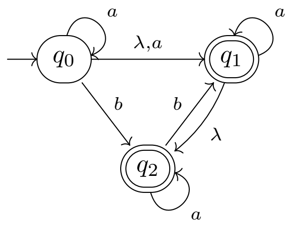
Test 1, Wednesday, Dec. 19, 2012
15h45 - 17h00

This test consists of **five** exercises. It is advised to explain your approach and to check your answers carefully. You can score a maximum of 100 points. Each question indicates how many points it is worth. The test is closed book. You are NOT allowed to use a calculator, a computer or a mobile phone. You may answer in Dutch or in English. Please write clearly, and do not forget to put on each page: your name, your student number, and your werkcollege group. Put your student-card clearly visible at the corner of your table (for inspection).

1. **(20 points)** Prove that the following languages are regular.
   (a) $\{w \in \{a,b\}^* \mid w$ begins with b and has an even number of a's $\}$
   (b) $\{w \in \{a,b\}^* \mid w$ contains the substring ba and does not contain the substring bb $\}$

2. **(20 points)** Let L_1 be a non-regular language and L_2 a regular language.
   (a) Is the language L_1L_2 always regular? If not, give a counter-example (that is, concrete L_1 and L_2 for which L_1L_2 is non-regular).
   (b) Give L_1 and L_2 such that L_1L_2 is regular.

3. **(20 points)** Consider the following non-deterministic automaton with λ transitions:



   (a) Construct an equivalent deterministic automaton (using the power set construction). In your answer, show the intermediate steps needed to reach the final result (so that we can evaluate that you understand the algorithm).

4. **(20 points)** Prove that the following language is not regular.
   (a) $\{w \in \{a,b\}^* \mid \ |w|_a = 2|w|_b\}$, where $|w|_a$ and $|w|_b$ denote the number of a and b in w, respectively.

5. **(20 points)** Given a language $L \subseteq A^*$, define two words $u, v \in A^*$ to be equivalent with respect to L, written $u \sim_L v$, as follows.

$$u \sim_L v \overset{\text{def}}{\iff} \text{for all } z \in A^* \quad (uz \in L \iff vz \in L)$$

   That is, two words are L equivalent if no matter how we extend them (by appending z) they either both belong to L or they both do not.
   (a) Suppose $L = \{a^n \mid n \text{ is even}\}$. Which words $v \in A^*$ are L-equivalent to λ?
   (b) Prove that if L is regular, then there exist $w_1, \ldots, w_n$, such that for each $w \in A^*$ there is a i such that $w \sim_L w_i$.

Talen en Automata

Test 2, Wednesday, Jan. 22, 2013

15h45 - 17h30

This test consists of **six** exercises. It is advised to explain your approach and to check your answers carefully. You can score a maximum of 100 points. Each question indicates how many points it is worth. The test is closed book. You are NOT allowed to use a calculator, a computer or a mobile phone. You may answer in Dutch or in English. Please write clearly, and do not forget to put on each page: your name, your student number, and your werkcollege group. Put your student-card clearly visible at the corner of your table (for inspection).

1. **(10 points)** Consider the following context-free grammar, with starting symbol S.

$$\begin{aligned} S &\to VV \mid V \\ V &\to (S) \mid a \mid b \mid c \end{aligned}$$

For each of the following sentences say whether it is described by this grammar or not. If it is, give either a syntax tree with S at its root or a derivation from S.

i. $a(b)$

ii. abc

iii. $a(bc)$

2. **(10 points)** Write a context-free grammar describing the language containing all strings of a's and b's for which the number of a's is even.

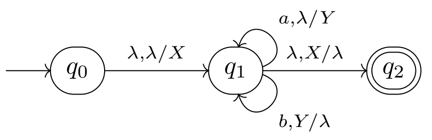
3. **(20 points)** Consider the language

$L = \{w \in \{a, b\}^* \mid$ the first, middle, and last characters of w are the same and $|w|$ is odd.$\}$.

(a) Provide a context-free grammar that generates L.

(b) Provide a pushdown automaton that recognizes L.

4. **(20 points)** Consider the following pushdown automaton with starting state q_0 and final state q_2.



(a) Which of the strings $aaabbbba$, $aaabbabb$ and $aaabbbaababb$ are accepted?

(b) Describe the language accepted by this pushdown automaton.

5. **(20 points)** Consider the grammar below over alphabet $\Sigma = \{(,), a, +, *\}$:

$$\begin{aligned} S &\to T \mid S + T \\ T &\to F \mid T * F \\ F &\to a \mid (S) \end{aligned}$$

The grammar accepts sentences such as $a+(a*a)$. Construct a PDA recognizing the language accepted by this grammar.

6. **(20 points)** The language

$$L = \{ww \mid w \in \{a, b\}^*\}$$

is not a context free language. However, its complement $\bar{L}$ is context free. Construct a grammar for $\bar{L}$.

Talen en Automata

Re-take exam, Friday, April 5, 2013
15h45 - 17h30

This exam consists of **three** parts. Each part is optional and will replace the mark you had for, respectively, homework, first and second tests.

It is advised to explain your approach and to check your answers carefully. You can score a maximum of 100 points for each part. Each question indicates how many points it is worth. The exam is closed book. You are NOT allowed to use a calculator, a computer or a mobile phone. You may answer in Dutch or in English. Please write clearly, and do not forget to put on each page: your name, your student number, and your werkcollege group. Put your student-card clearly visible at the corner of your table (for inspection).

Part 1 (Homework)

1. **(30 points)** Give a regular expression r over the alphabet $A = \{a, b, c\}$ such that the language determined by r consists of all strings that contain at least one occurrence of each symbol in A. Briefly explain your answer.
2. **(30 points)** Give a deterministic finite automaton over the alphabet $\{a, b\}$ which accepts all strings containing no more than two consecutive occurrences of the same input letter. (For example, *abba* should be accepted but not *abaaab*.)
3. **(40 points)** Give a pushdown automaton that recognizes the language $\{a^n b^n \mid n \in \mathbb{N}\}$.

Part 2 (Regular languages)

1. Let L be the language accepted by the following non-deterministic finite automaton with λ-transitions.

$$\begin{tikzcd}
{} \arrow[r] & q_0 \arrow[rr, "a"] \arrow[dr, "a"] & & q_1 \arrow[rr, "a"] & & q_2 \arrow[rr, "a"] \arrow[dlll, "a"'] \arrow[dl, "\lambda"] & & q_3 \\
& & q_4 \arrow[rr, bend left, "a"] & & q_5 \arrow[ll, bend left, "a"] & & &
\end{tikzcd}$$

   (a) **(20 points)** Construct an equivalent deterministic automaton (using the power set construction). In your answer, show (some of) the intermediate steps needed to reach the final result (so that we can evaluate that you understand the algorithm).

   (b) **(20 points)** Give a regular expression that denotes L.

2. **(30 points)** Is the language $\{a^k b^l a^m \mid k \geq n + l\}$ regular? Provide a proof of your answer.
3. **(30 points)** Show that regular languages are closed under doubling: If a language L is regular, then so is the language $\{\mathbf{two}(x) \mid x \in L\}$, where string doubling is defined by $\mathbf{two}(\lambda) = \lambda$ and $\mathbf{two}(xa) = (\mathbf{two}(x))aa$ (for instance, $\mathbf{two}(abb) = aabbbb$).

Part 3 (Context-free languages)

1. Consider the language $L = \{xy \in \{a, b\}^+ \mid y = x^R\}$, where x^R denotes the reverse string of x.

   (a) **(30 points)** Give a context-free grammar for L.

   (b) **(30 points)** Give a pushdown automaton that accepts L.

2. **(40 points)** Construct a pushdown automaton, with two states, that accepts the grammar over alphabet $\Sigma = \{(,), a, +, *\}$:

$$\begin{aligned}
S &\to T \mid S + T \\
T &\to F \mid T * F \\
F &\to a \mid (S)
\end{aligned}$$