Automata: a short introduction

Paolo Turrini

ILIAS, University of Luxembourg

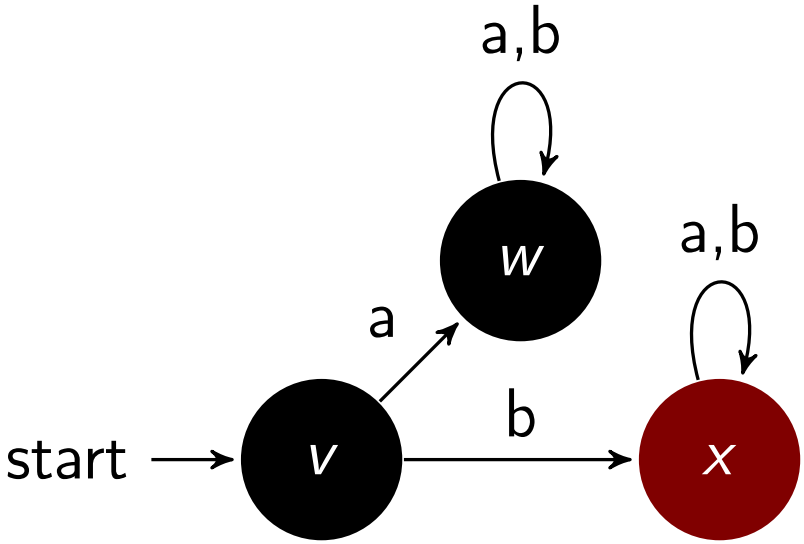
Discrete Mathematics II

May 2012

What is a computer?

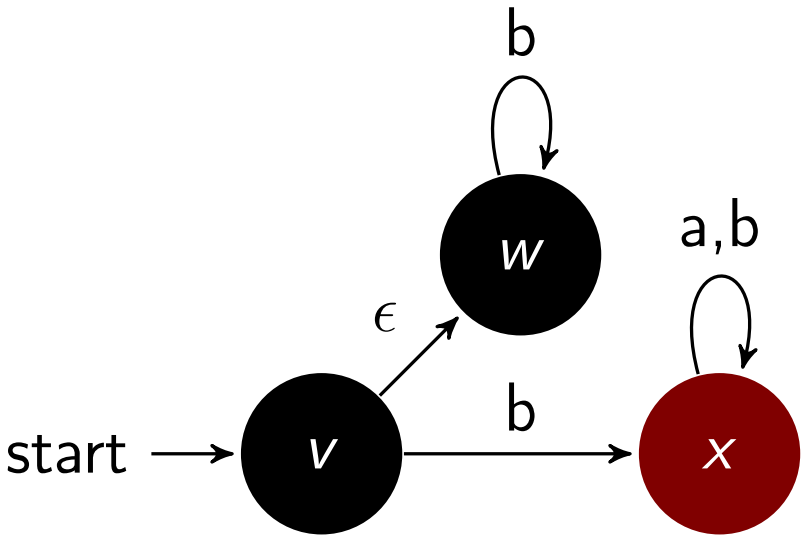
- Real computers are complicated;
- We abstract up to an essential *model of computation*;
- We begin with the simplest possible model, a *finite automaton*.

Outline: Deterministic Finite Automata



- A formal description of abstract computers, with a finite number of *states*, with distinguished *accepting* ones, and with labelled *transitions*, where each *action* labels exactly one outgoing arrow.

Outline: Non-deterministic Finite Automata



- A formal description of abstract computers, with a finite number of *states*, with distinguished *accepting* ones, and with labelled *transitions*, where each *action* **needs not** label exactly one outgoing arrow. Special characters are possible.

Outline: Languages

- Abstract languages, i.e. sets of words obtained from a given alphabet.
  - $\mathcal{L} = \{a, b, ba, ab, aaa, aab, \dots\}$, i.e. all possible combinations of words formed using a and b;
  - $\mathcal{L}' = \{\varepsilon\}$, the language containing only the empty word;
  - $\mathcal{L}' = \{a^n b^n |$ for each natural number $n\}$, i.e. the language of words with n times a followed by n times b, for instance having $aaabbb$ inside.

Outline: Regular Expressions

- Operations that *generate* languages with properties that are interesting from a computational point of view.
  - $(a \cup b)^* = \{a, b, ba, ab, aaa, aab, \dots\}$, i.e. all possible combinations of words formed using a and b;
  - $(a \circ b) = \{ab\}$ i.e. the concatenation of the words a and b
  - $(a)^* = \{\varepsilon, a, aa, aaa, aaaa, \dots\}$, i.e. the word a repeated zero, one or many times.

What we will talk about

- We will study the *mathematical properties* of these structures.

What we will observe

- These structures *are one and the same thing*.

The book

Michael Sipser

Introduction to the Theory of Computation

2nd edition, 2006

Formal definition of an automaton

- An automaton has several parts:
  - a set of states;
  - rules for going from one state to another, depending on the input symbol;
  - a starting state and a set of accepting states.
- The formal definition:
  - makes this precise;
  - it can be used to study automata *as mathematical structures*.

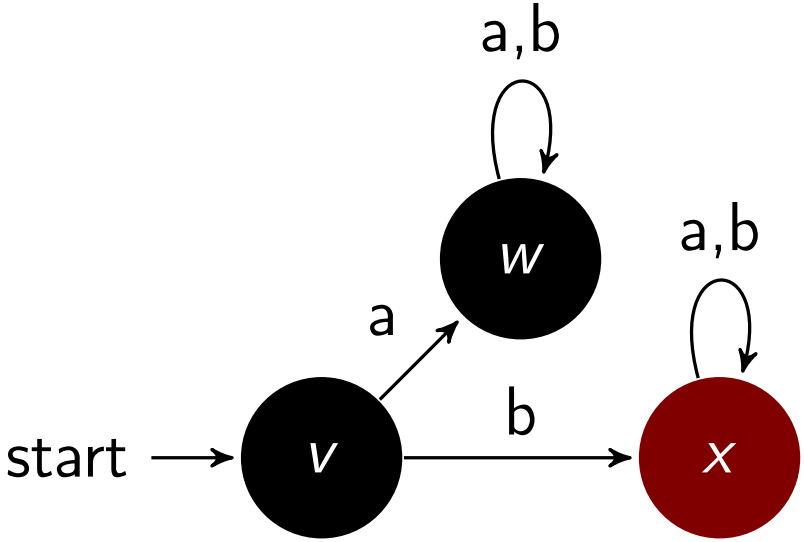
The formal definition

Definition (Deterministic Finite Automata)

A *deterministic finite automaton* is a tuple $(Q, \Sigma, \delta, q_0, F)$, where:

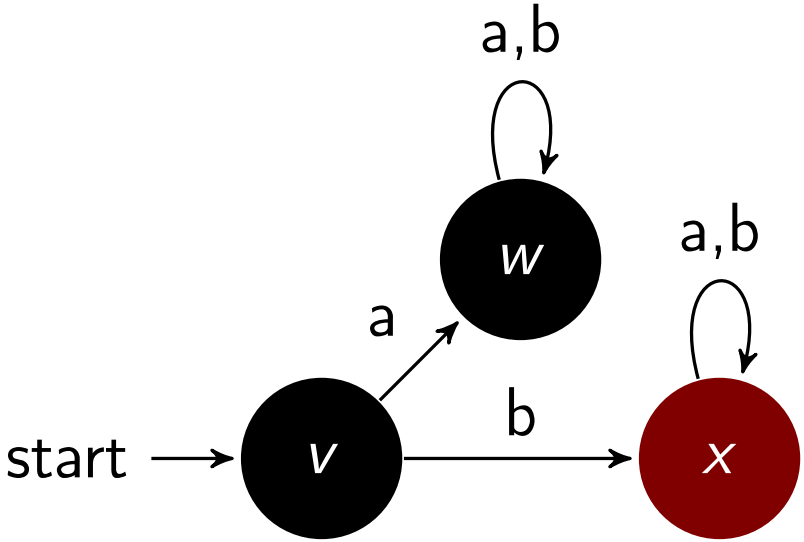
1. Q is a finite set called the **states**;
2. Σ is a finite set called the **alphabet**;
3. $\delta : Q \times \Sigma \to Q$ is the **transition function**;
4. $q_0 \in Q$ is the **start state**;
5. $F \subseteq Q$ is the **set of accepting (or terminal) states**.

Example



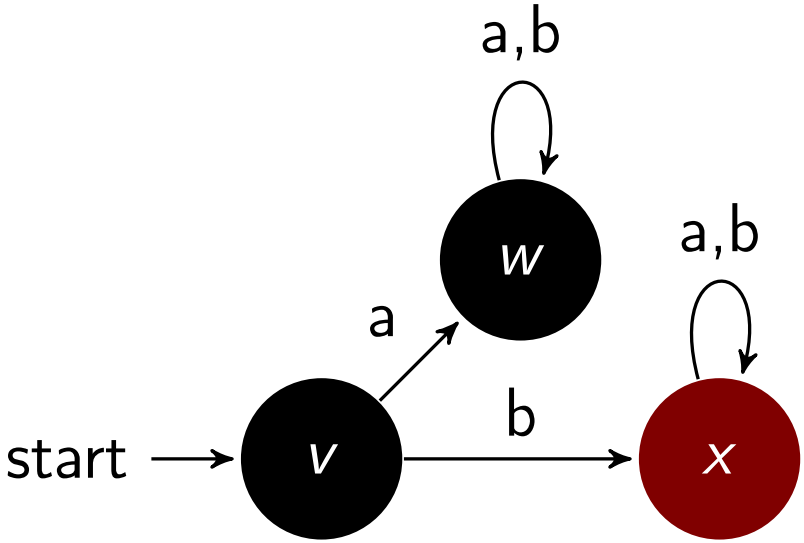
- The set of states Q is $\{v, w, x\}$.

Example



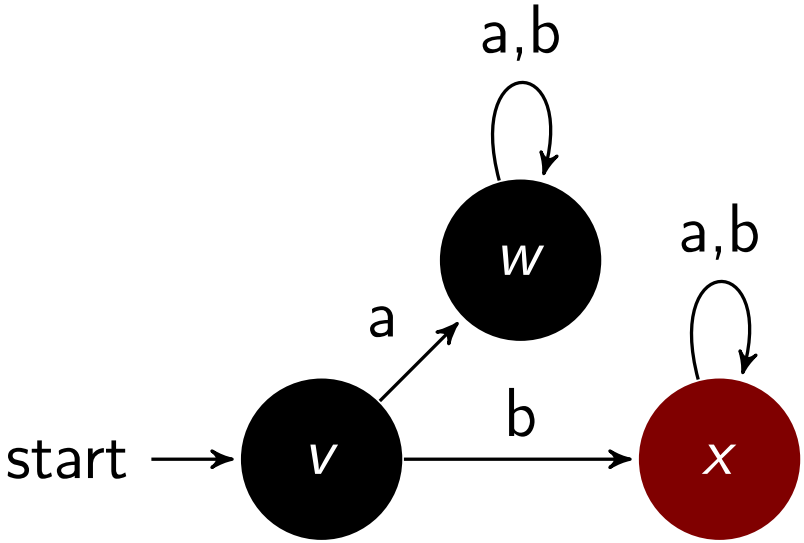
- The alphabet Σ is $\{a, b\}$, the set of basic characters that we have used.

Example



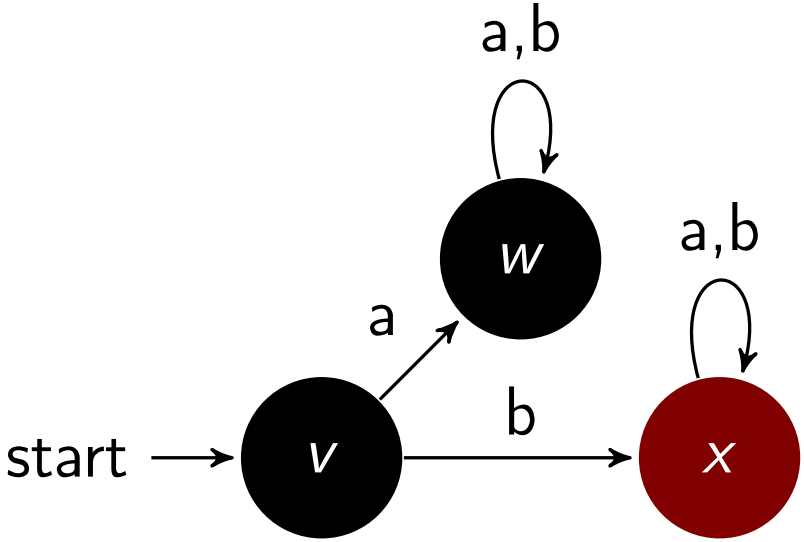
- The transition function δ is $\delta(v, a) = w, \delta(v, b) = x, \delta(w, a) = \delta(w, b) = w, \delta(x, a) = \delta(x, b) = x$.

Example



- The initial state q_0 is v.

Example



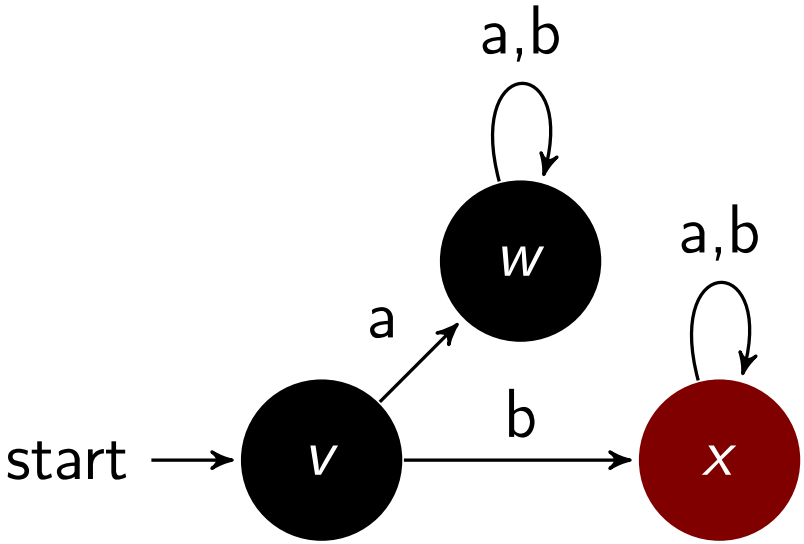
- The set F of accepting states is $\{x\}$.

Definition (Word acceptance)

Let Σ be our alphabet and let $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ be our automaton. A finite sequence $w_1, w_2, \ldots, w_n$, where each w_i is an element of Σ, is *accepted* by $\mathcal{A}$ if and only if there exists a sequence of states $r_0, r_1, \ldots, r_n$ in Q such that:

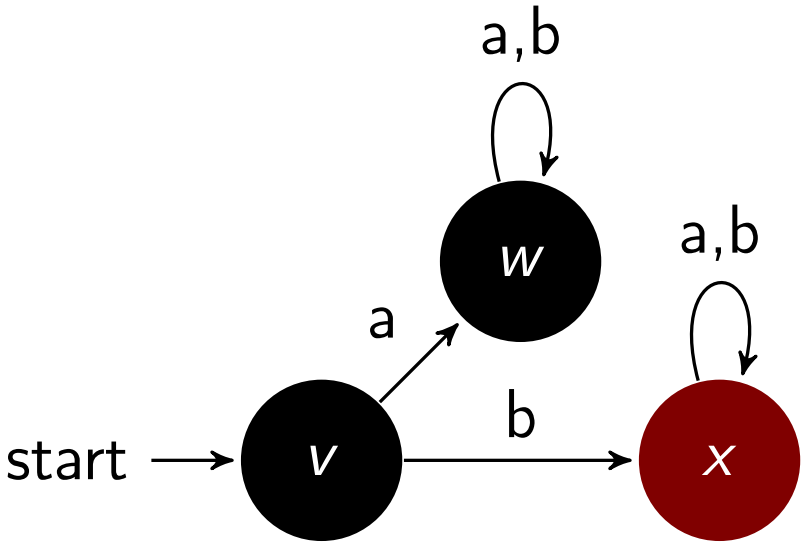
1. $r_0 = q_0$, i.e. we start from the starting state;
2. for each i between 0 and $n-1$, $\delta(r_i, w_{i+1}) = r_{i+1}$, i.e. the computation follows exactly the word;
3. $r_n \in F$, i.e. we end up in an accepting state.

Example



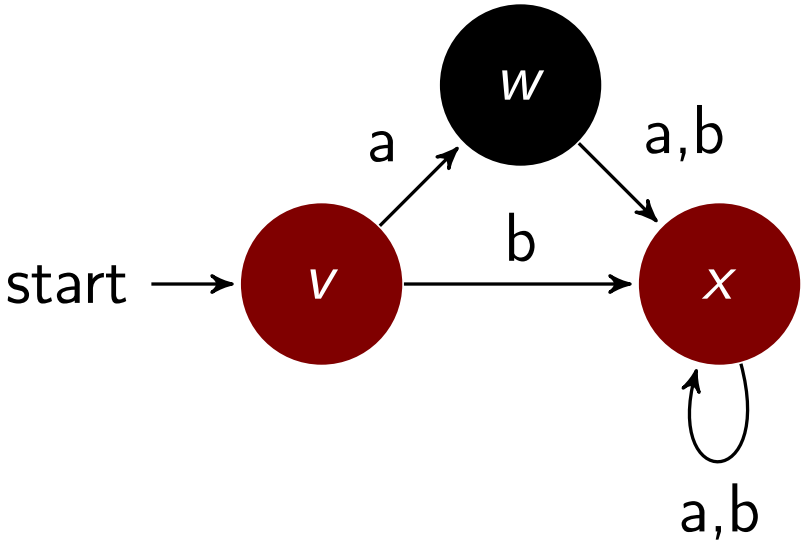
- The word (b, b, a, b, a, b), briefly *bbabab*, is accepted by this automaton, because there is a sequence of states, namely (v, x, x, x, x, x, x), briefly *vxxxxxx*, such that:
  1. $v = q_0$, i.e. we start from the starting state;
  2. for each i between 0 and 6, $\delta(r_i, w_{i+1}) = r_{i+1}$, i.e. the computation follows exactly the word;
  3. $x \in F$, i.e. we end up in an accepting state.

Example



- The empty word, ε, is *not* accepted by this automaton because... the starting state is not an accepting state!

Example



- Can you find a word, longer than 20 characters, that is accepted by this automaton?
- Can you find a word that is *not* accepted by this automaton?
- Is ε, the empty word, accepted by this automaton?

Regular Languages

Definition (Language Recognition)

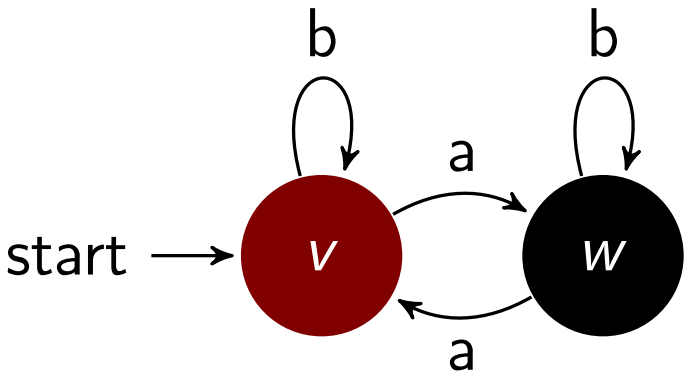
We say that a deterministic finite automaton $\mathcal{A}$ *recognizes* a language $\mathcal{L}$ if and only if $\mathcal{L} = \{w \mid \mathcal{A} \text{ accepts } w\}$

Regular Languages

Definition (Language Recognition)

A language is called *regular* if and only if some deterministic finite automaton recognizes it.

Regular Languages



- The language $\mathcal{L}$, made by all words with an even number of a, is regular;
- Is the language $\mathcal{L}'$, made by all words with an *odd* number of a, regular?

Exercise

- Show that:
  - The language made only by ε, is regular;
  - The empty language, i.e. the language with no words, is regular;
  - The language made by all words with no b is regular;
  - The language made by all words with more than two a is regular;
  - The language made by all words with exactly two b is regular;
  - The language $\{babba, cab, \varepsilon\}$ is regular.

Regular Operations on Languages

- So far we have described languages that automata can recognize;
- Each time we have constructed an automaton;
- However we can apply certain *regular* operations on languages, being sure to preserve regularity: if we start with a regular language, no matter how many times we will apply these operations, we will still have a regular language.

Regular Operations on Languages

Definition

Let $\mathcal{L}, \mathcal{L}'$ be two languages. We define the regular operations of **union**, **concatenation**, **star**, as follows:

Union: $\mathcal{L} \cup \mathcal{L}' = \{x \mid x \in A \text{ or } x \in B\}$, i.e. words in the first language with words in the second language;

Concatenation: $\mathcal{L} \circ \mathcal{L}' = \{xy \mid x \in A \text{ and } y \in B\}$, i.e. words in the first language followed by words in the second language;

Star: $\mathcal{L}^* = \{x_1, x_2, \ldots, x_n \mid n \geq 0 \text{ and each } x_i \in \mathcal{L}\}$, i.e. words made by repeting words in the starting language, zero, one, or many times.

Regular Operations on Languages

Definition

Let $\mathcal{L} = \{a, babab, bbb\}, \mathcal{L}' = \{bb\}$. We have:

Union: $\mathcal{L} \cup \mathcal{L}' = \{a, babab, bb, bbb\}$, i.e. words in the first language with words in the second language;

Concatenation: $\mathcal{L} \circ \mathcal{L}' = \{abb, bababbb, bbbbb\}$, i.e. words in the first language followed by words in the second language. Notice that $\mathcal{L} \circ \mathcal{L}'$ is not the same as $\mathcal{L}' \circ \mathcal{L}$;

Star: $\mathcal{L}^* = \{\varepsilon, a, babab, ababab, abbb, bbbbbb, abbba, \dots\}$, i.e. words made by repeting words in the starting language, zero, one, or many times.

Exercise

Let $\mathcal{L}$ be the language of words with exactly two a and let $\mathcal{L}'$ be the language of words with exactly two b.

- What is $\mathcal{L} \cup \mathcal{L}'$?
- What is $\mathcal{L} \circ \mathcal{L}'$?
- What is $\mathcal{L}^*$?

A fundamental result

Theorem (Closure)

Let $\mathcal{L}, \mathcal{L}'$ be two languages. If $\mathcal{L}, \mathcal{L}'$ are regular then:

- *$\mathcal{L} \cup \mathcal{L}'$ is regular;*
- *$\mathcal{L} \circ \mathcal{L}'$ is regular;*
- *Both $\mathcal{L}^*$ and $\mathcal{L}'^*$ are regular.*

Exercise

Definition

Let $\mathcal{L} = \{a, babab, bbb\}, \mathcal{L}' = \{bb\}$. We have:

Union: $\mathcal{L} \cup \mathcal{L}' = \{a, babab, bb, bbb\}$;

Concatenation $\mathcal{L} \circ \mathcal{L}' = \{abb, bababbb, bbbbb\}$;

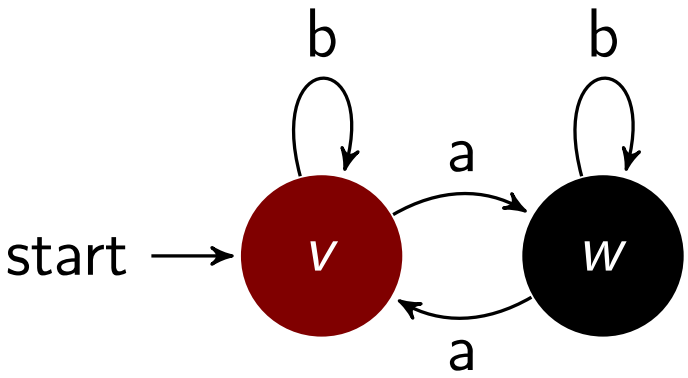
Star: $\mathcal{L}^* = \{\varepsilon, a, babab, ababab, abbb, bbbbbb, abbba, \dots\}$.

- Can you construct the automata recognizing the new languages?
- Think of how they are obtained from the original ones.

What we have seen so far

- Deterministic finite automata are abstract models of computation;
- They recognize languages;
- The language that they recognise are closed under the operations of union, concatenation, and star.

The constraints of determinism

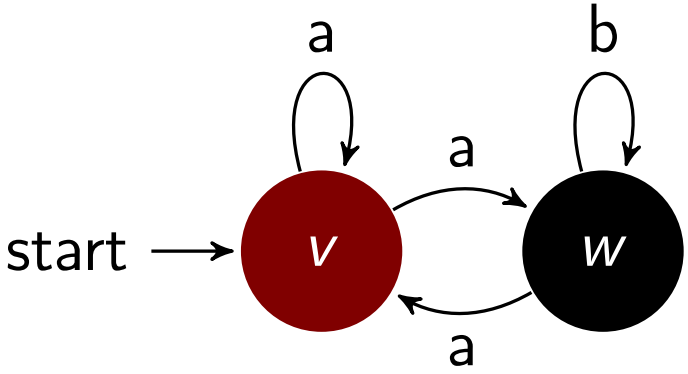


- In deterministic finite automata, when the machine is at a state and reads a symbol, we know exactly what the next state will be;
- Now we want to generalize these models, and allow a machine to be *at different states at the same moment*.

A generalization

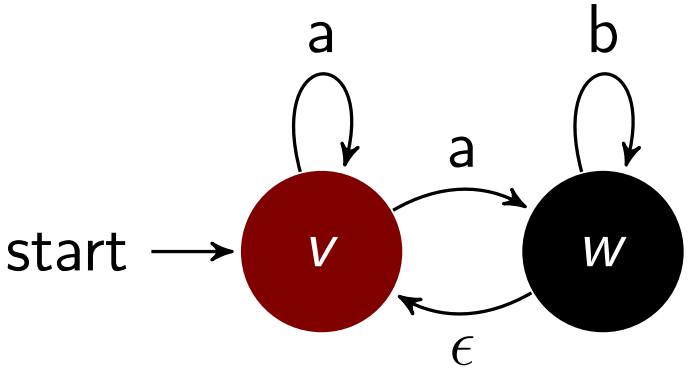
- Non-deterministic finite automata:
  - might have ϵ actions, that jump to different states istantaneously;
  - from each letter from the alphabet, there might be zero, one, or more outgoing arrows labelled with that letter.
- In fact every deterministic finite automaton is *just a special case* of non-deterministic finite automata: can you see why?

Different states at the same moment



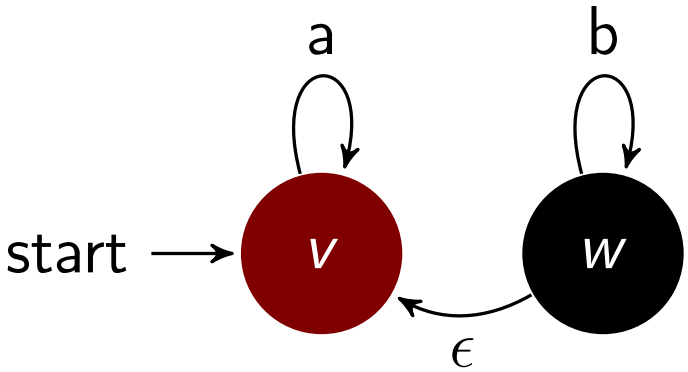
- At state v, after reading a, the machine is in *both* v and w.

Different states at the same moment



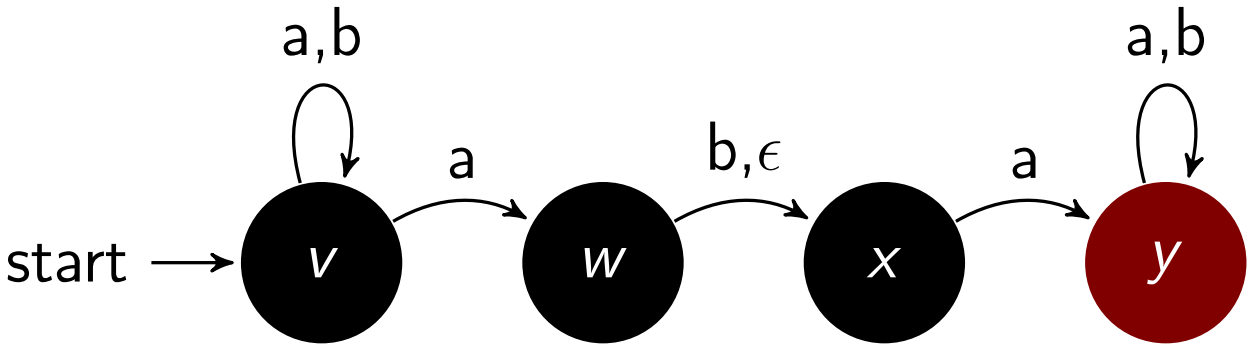
- At state w, after reading b, the machine is in *both* v and w.

Different states at the same moment



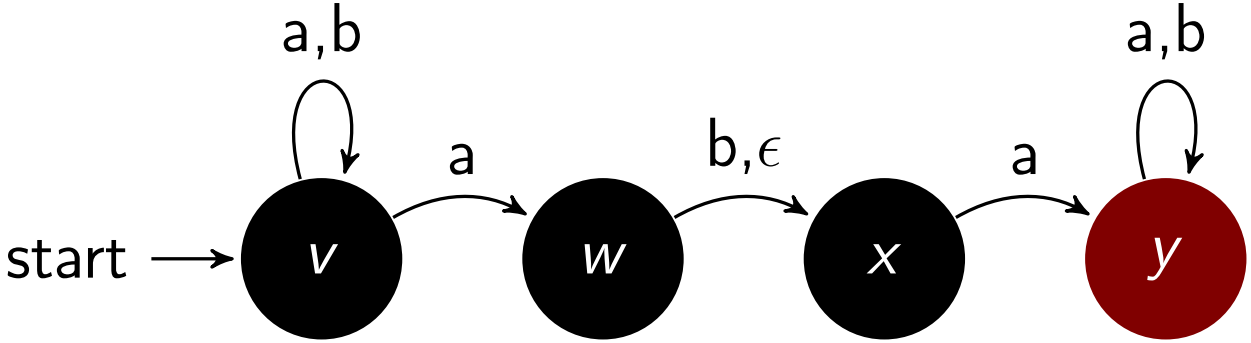
- At what state(s) goes the machine if it reads a at state v?
- And if it reads b at state w?
- And if it reads a at state w?

An example: babaab



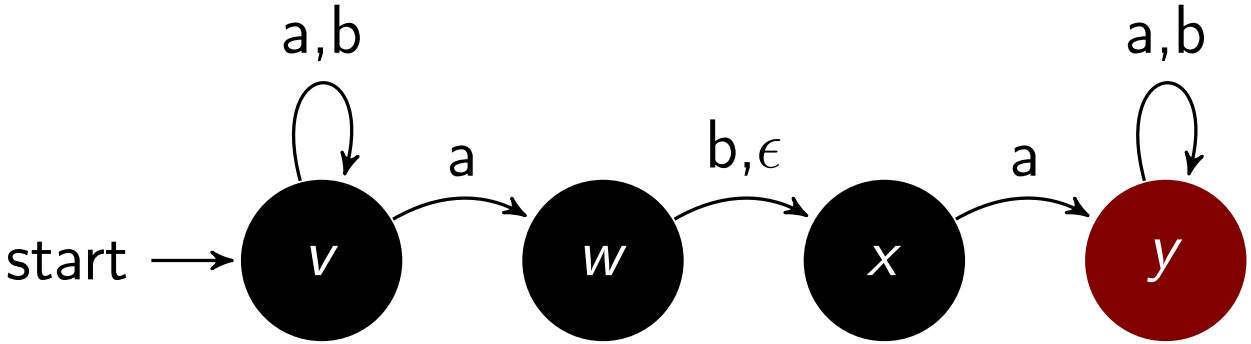
- Suppose we are at the start state and the machine receives the word *babaab*;
- We start from the state v; we read b;
- We go to v and we now read a.
- We go to v and w and also x (notice the ϵ), following all the possibilities.

An example: babaab



- We are in v, w and x and the third letter is b. From v we go to v, while from w we go to x, finally from x we break the computation (notice: with no arrow we go nowhere!);
- We are in v and x and fourth letter is a; now we are in v, w, x, y. (notice the ϵ!);
- Now we read again a: we go to v, w, x, y;
- Finally we read b and we go to v, x, y.

An example: babaab



- We ended up in v, x, y;
- Acceptance rule: if we end up in at least one accepting state, the word is accepted.

The formal definition

- Remember first that, for every set A, its powerset (which we indicate with 2^A) is the set of all its subsets.

The formal definition

Definition (Non-deterministic Finite Automata)

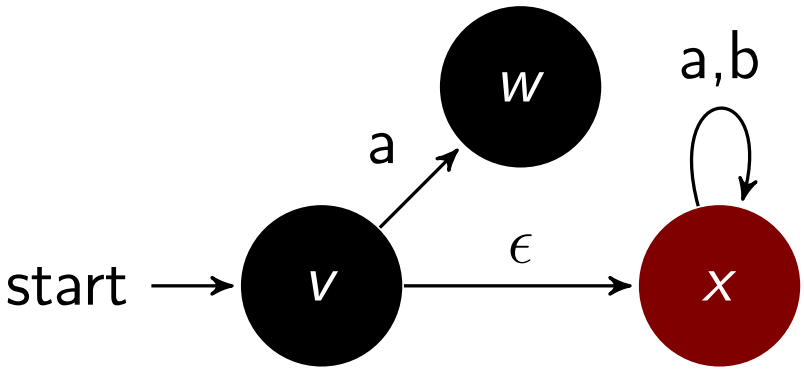
A *non-deterministic finite automaton* is a tuple $(Q, \Sigma, \delta, q_0, F)$, where:

1. Q is a finite set called the **states**;
2. Σ is a finite set called the **alphabet**;
3. $\delta : Q \times \Sigma \cup \epsilon \rightarrow 2^Q$ is the **transition function**;
4. $q_0 \in Q$ is the **start state**;
5. $F \subseteq Q$ is the **set of accepting (or terminal) states**.

Differences

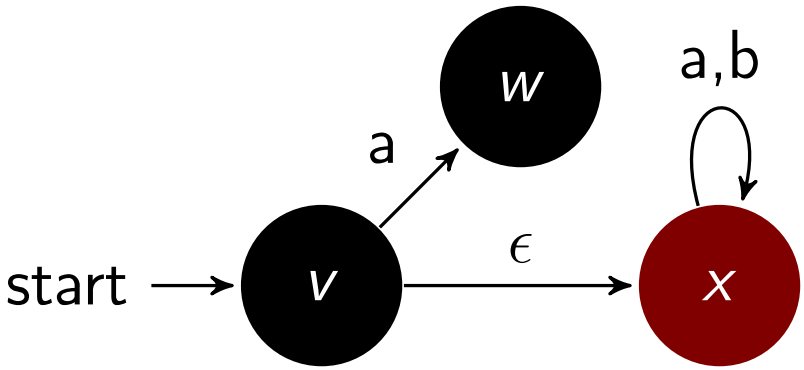
- The only special feature is the transition function $\delta : Q \times \Sigma \cup \epsilon \rightarrow 2^Q$, which associates to each state and each input symbol *enriched with the* ϵ action a possibly empty set of successor states;

Example



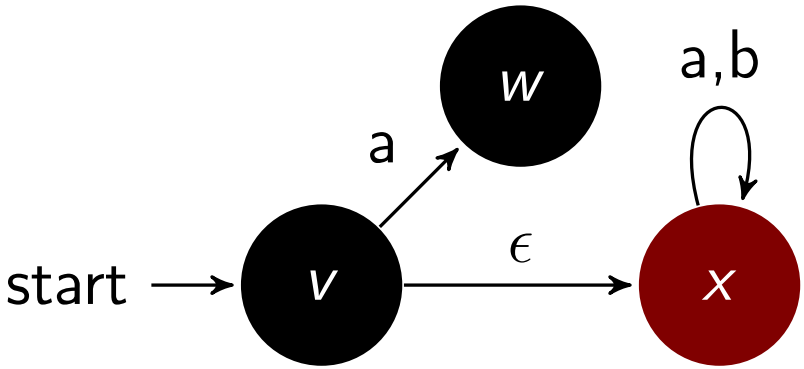
- The set of states Q is $\{v, w, x\}$.

Example



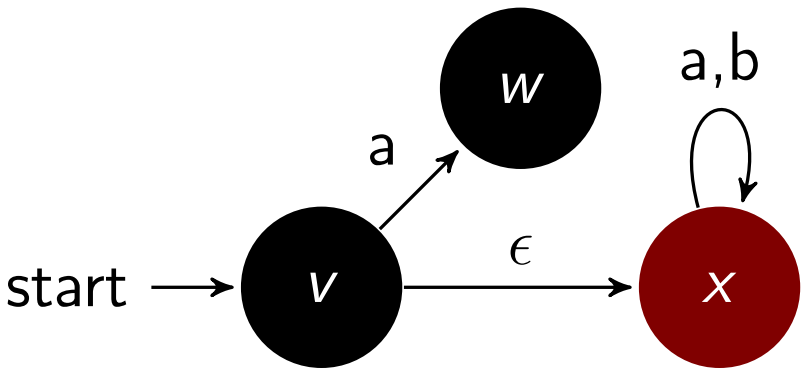
- The alphabet Σ is $\{a, b\}$, the set of basic characters that we have used.

Example



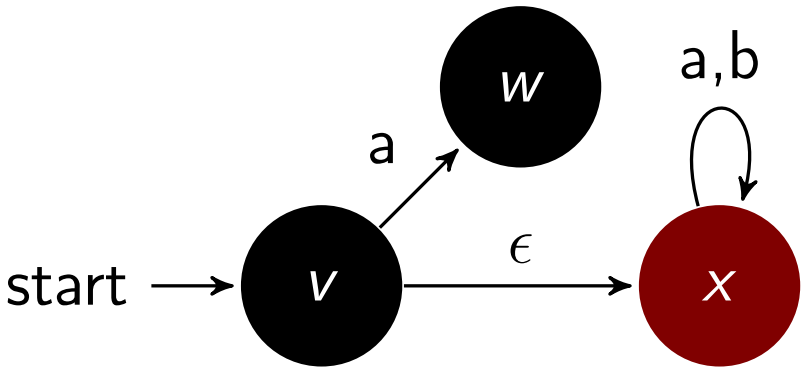
- The transition function δ is
  $\delta(v, \epsilon) = \{x\}, \delta(v, a) = \{w\}, \delta(v, b) = \emptyset, \delta(w, a) = \delta(w, b) = \emptyset, \delta(x, a) = \delta(x, b) = x, \delta(w, \epsilon) = \delta(x, \epsilon) = \emptyset.$

Example



- The initial state q_0 is v.

Example



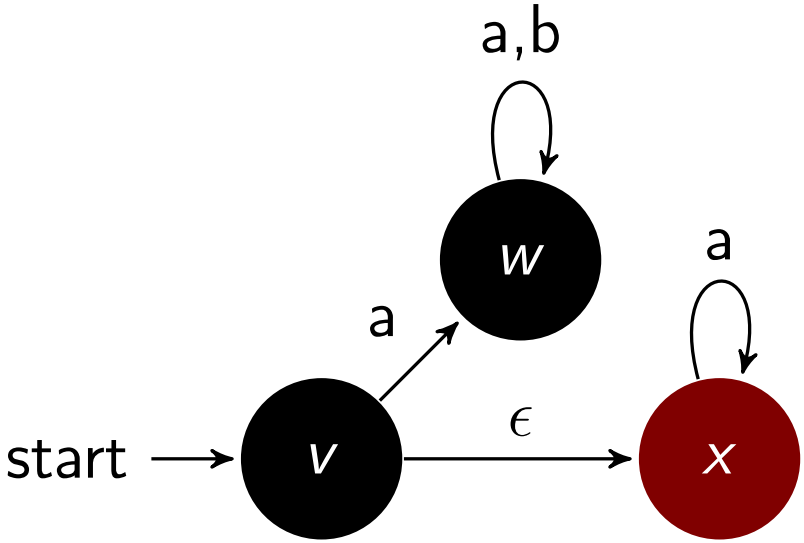
- The set F of accepting states is $\{x\}$.

Definition (Word acceptance)

Let Σ be our alphabet and let $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ be our automaton. A finite sequence $w_1, w_2, \ldots, w_n$, where each w_i is an element of $\Sigma \cup \epsilon$, is *accepted* by $\mathcal{A}$ if and only if there exists a sequence of states $r_0, r_1, \ldots, r_n$ in Q such that:

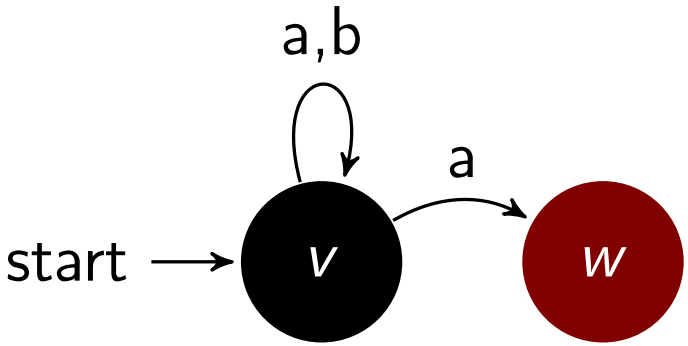
1. $r_0 = q_0$, i.e. we start from the starting state;
2. for each i between 0 and $n-1$, $r_{i+1} \in \delta(r_i, w_{i+1})$, i.e. the computation follows exactly the word;
3. $r_n \in F$, i.e. we end up in an accepting state.

Example



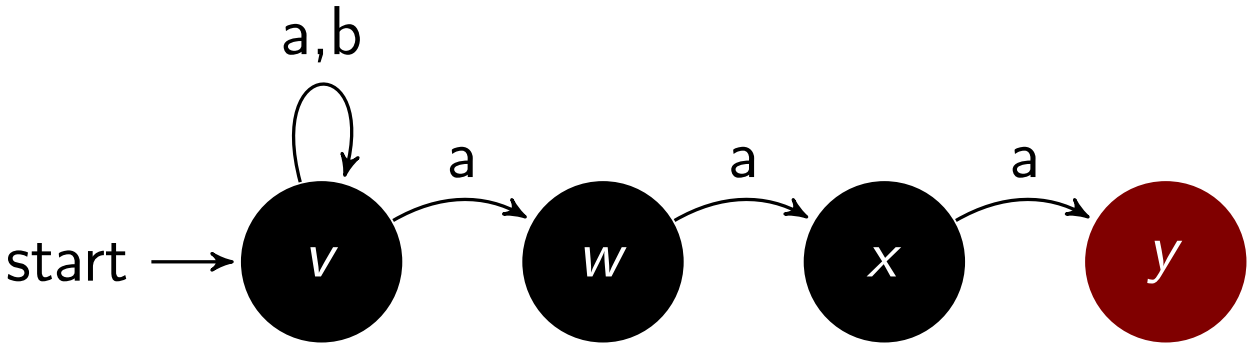
- The empty word ϵ is accepted;
- The word a is accepted;
- The word b is not accepted.
- Can you tell some more words that are accepted or not?

An exercise



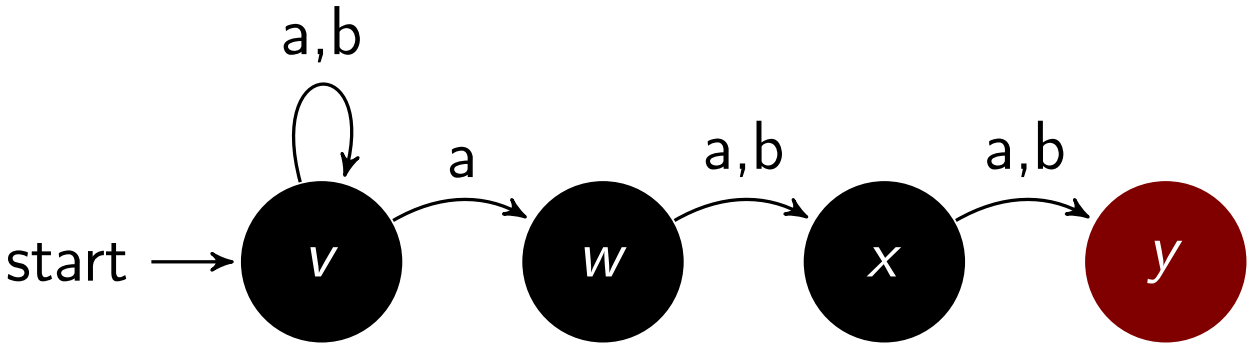
- What language does this automaton recognize?

An exercise



- What language does this automaton recognize?

An exercise



- What language does this automaton recognize?

It is not a *proper* generalization

Theorem

Every nondeterministic finite automaton has an equivalent deterministic finite automaton

Proof idea 1

We know that every DFA is an NFA. But what about the contrary? If a language is recognized by an NFA, then we must show the existence of a DFA that also recognizes it. The idea is to convert the NFA into an equivalent DFA that simulates the NFA.
But we can use the idea that when we encounter the same simbol on an NFA the machine splits in multiple copies!

Proof idea 2

If k is the number of states of the NFA, it has 2^k subsets of states. Each subset corresponds to one of the possibilities that the DFA must remember, so the DFA simulating the NFA will have 2^k states. If one state w is connected to one state z in the NFA, then all the sets in the DFA containing w will be connected to states containing z. And if some state accepts in the NFA then all sets containing that state will be accepting in the corresponding DFA.

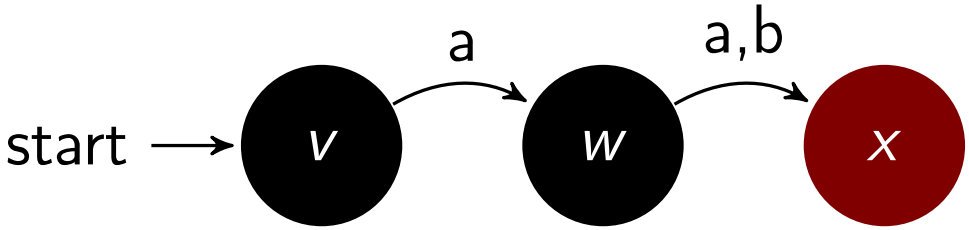
A consequence

- We have called *regular* a language recognized by some deterministic finite automaton. But now we can say more.

Proposition (Equivalence)

A language is regular if and only if it is recognized by some nondeterministic finite automaton.

Ideas



- What language does this automaton recognize?
- How does a DFA look like recognizing the same language?

What we have seen so far

- Nondeterministic finite automata generalize deterministic ones, by allowing:
  - zero, one, more outgoing arrows labelled with the same alphabet symbol;
  - special ϵ actions;
- However they *cannot* recognize more languages than deterministic finite automata;
- But, they are much more efficient! To construct a DFA simulating and NFA of n states we need between n and 2^n states.

Building languages

- In arithmetic, we can use operations $+$ and $\times$ to build expressions such as $(5 + 3) \times 4$
- Similarly, we can use regular operations (such as union, concatenation and star) to build up expressions describing languages;

Building languages

- Consider $(0 \cup 1)0^*$. What is the value of this expression?
  - In this case the value is the language consisting of all strings starting with a 0 or a 1 followed by any number of 0.
  - So for instance words such as $1, 0, 10000$ are in this language;
  - Words such as $01, 11, 1100000$ are not in this language.

Building languages

- How can we calculate the value of $(0 \cup 1)0^*$? We get the result by dissecting the expression in small parts:
  - The symbols 0 and 1 are shorthand for the sets $\{0\}$ and $\{1\}$;
  - So $(0 \cup 1)$ is the language $\{0, 1\}$;
  - The part 0^* means $\{0\}^*$, and its value is all the strings consisting of any number of 0;
  - $(0 \cup 1)0^*$ is a shorthand for $(0 \cup 1) \circ 0^*$;

Building languages

- What is the value of $(0 \cup 1)^*$?
- For convenience,
  - when the alphabet $\Sigma = \{0, 1\}$, we abbreviate the expression $(0 \cup 1)$ with Σ.
  - We write R^+ to mean $R \circ R^*$. Notice that $R^+ \cup \epsilon = R^*$.

Regular expressions

- They are everywhere in theoretical computer science;
- Many programming languages use them to facilitate description of long expressions;

Formal definition

Definition

R is a *regular expression* if and only if it is of the form:

1. a for some a in some alphabet Σ;
2. ϵ;
3. $\emptyset$;
4. $(R_1 \cup R_2)$, where R_1 and R_2 are two regular expressions;
5. $(R_1 \circ R_2)$, where R_1 and R_2 are two regular expressions;
6. (R_1^*), where R_1 is a regular expression.

Nothing else is a regular expression.

Example

Assume $\Sigma = \{0, 1\}$. We have the following:

- $0^*10^* = \{w \mid$ w contains a single $1\}$
- $\Sigma^*1\Sigma^* = \{w \mid w$ has at least one $1\}$
- $(\Sigma\Sigma)^* = \{w \mid w$ is a string of even length$\}$
- $(0 \cup \epsilon)1^* = 01^* \cup 1^*$
- $1^*\emptyset = \emptyset$
- $\emptyset^* = \epsilon$

Exercise

Assume $\Sigma = \{0, 1\}$. Calculate and describe the following:

- $(0 \cup \epsilon)(1 \cup \epsilon)$
- $(\Sigma\Sigma\Sigma)^*$
- $(01^+)^*$

Exercise

Let R be a regular expression. Are the following statements true?

- $R \cup \emptyset = R$
- $R \circ \epsilon = R$
- $R \cup \epsilon = R$
- $R \circ \emptyset = R$

Equivalence with finite automata

Theorem

A language is regular if and only if some regular expression describes it

Proof idea

Two lemmas:

1. *If a language is described by a regular expression, then it is regular;*
2. *If a language is regular, then it is described by a regular epxression.*

Proof idea

For the first case, for the language A described by R we can construct an NFA recognizing it. But then A is regular. For the second case, we know that the the language A described by R is regular. But then we can construct a DFA recognizing it. Then we use a standard procedure to convert the DFA into an equivalent regular epression.

What we have seen so far

- Regular expressions are building blocks of languages;
- They can build all regular languages.

Summing up...

- Deterministic finite automata are equivalent to nondeterministic finite automata;
- Determinisitic finite automata are equivalent to regular expressions;
- Regular expressions are equivalent to nondeterministic finite automata;

Nonregular languages

- Not all languages are regular!
- How do non-regular languages look like?

Nonregular languages

Proposition

All finite languages are regular

- Can you prove this?
- Construct an automaton that is big enough and accept all and only the words of that language.

Nonregular languages

- What about infinite languages?
- Can we reuse the same argument?

Nonregular languages

Nonregular languages

- Consider the following $\{a^n b^n \mid \text{ for every } n \in \mathbb{N}\}$
- We need to see n times a and then accept the word only if we see n times b , for every $n \in \mathbb{N}$!
- We would need an *infinite* finite automaton. But there is no such thing.

Nonregular languages

- Let w be a word and w^- be the same word written in reverse , e.g. $abb^- = bba$;
- Consider the following
  $\{wxw^- \mid \text{ for every word } w \text{ with } x \in \Sigma\}$
- What are these words?
- They are *palindromes*, words that are read the same in both directions

Nonregular lanuages

- The language of all the palindromes from an alphabet Σ is non-regular
- Again, for every word wx ($x \in \Sigma$) that we meet, we would need to remember it in order to accept the word if followed by w^-
- Again, we would need an infinite automaton!

A formal statement

- Notice first this: if a word of 6 characters is accepted by an automaton of 3 states then... there is a loop somewhere and it is leading to an accepting state!
- And if there is a loop somewhere leading to an accepting state then the language recognized by the automaton... is infinite!

The pumping lemma

Theorem

Let L be a regular language. Then there exists an natural number $p \geq 1$ such that every string w in L of length at least p (p is called the "pumping length") can be written as $w = xyz$ (i.e., w can be divided into three substrings), satisfying the following conditions:

- $|y| \geq 1$
- $|xy| \leq p$
- *for all* $i \geq 0, xy^i z \in L$

The pumping lemma

- If the conditions are not met, the language is not regular

Palindromes are not regular

Suppose that the set of palindromes were regular. Let p be the value from the pumping lemma. Consider the string $w = 0^p110^p$. w is clearly a palindrome and $|w| \geq p$. By the pumping lemma, there must exist strings x, y, and z satisfying the constraints of the pumping lemma. Pick any such x, y, z such that $w = xyz$, $|xy| \leq p$, and $|y| \geq 1$. Because $|xy| \leq p$, xy is entirely contained in the 0^p at the start of w. So x and y consist entirely of zeros. Now, consider xz. By the pumping lemma, xz must be in the language. But xz can't be a palindrome. This means that the set of palindromes doesn't satisfy the pumping lemma and, thus, the set of palindromes cannot be regular.

Conclusion

- Automata can recognize regular languages
- But they can't recognize every language
- Their power and limitation is summarized in the pumping lemma.