

Example. Let us prove that the language

$$L = \{0^n 10^{2n} \mid n \geq 0\}$$

is not regular. Define the homomorphism h :

$$\begin{aligned} h(a) &= 0, \\ h(b) &= 1, \\ h(c) &= 00. \end{aligned}$$

Then

$$h^{-1}(L) \cap a^*bc^* =$$

Example. Let us prove that the language

$$L = \{0^n 10^{2n} \mid n \geq 0\}$$

is not regular. Define the homomorphism h :

$$\begin{aligned} h(a) &= 0, \\ h(b) &= 1, \\ h(c) &= 00. \end{aligned}$$

Then

$$h^{-1}(L) \cap a^*bc^* = \{a^nbc^n \mid n \geq 0\}$$

Let us denote this language by L_1 . If L is regular then also L_1 is regular. Define another homomorphism g :

$$\begin{aligned} g(a) &= a, \\ g(b) &= \varepsilon, \\ g(c) &= b. \end{aligned}$$

Then

$$g(L_1) =$$

Example. Let us prove that the language

$$L = \{0^n 10^{2n} \mid n \geq 0\}$$

is not regular. Define the homomorphism h :

$$\begin{aligned} h(a) &= 0, \\ h(b) &= 1, \\ h(c) &= 00. \end{aligned}$$

Then

$$h^{-1}(L) \cap a^*bc^* = \{a^nbc^n \mid n \geq 0\}$$

Let us denote this language by L_1 . If L is regular then also L_1 is regular. Define another homomorphism g :

$$\begin{aligned} g(a) &= a, \\ g(b) &= \varepsilon, \\ g(c) &= b. \end{aligned}$$

Then

$$g(L_1) = \{a^n b^n \mid n \geq 0\}$$

But this language is not regular, so L_1 cannot be regular, which means that L cannot be regular either. We only used operations that preserve regularity (inverse homomorphism, intersection with a regular language, homomorphism).

Constructs of type

$$h^{-1}(L) \cap R$$

are very useful. They contain all words of R that are mapped to L by homomorphism h :

$$h^{-1}(L) \cap R = \{w \in R \mid h(w) \in L\}$$

Our next language operation is called **quotient**.

Let L_1 and L_2 be two languages. Their quotient L_1/L_2 is the language containing all words obtained by removing from the end of L_1 's words suffix that belongs to L_2 .

$$L_1/L_2 = \{w \mid wu \in L_1 \text{ for some } u \in L_2\}$$

Our next language operation is called **quotient**.

Let L_1 and L_2 be two languages. Their quotient L_1/L_2 is the language containing all words obtained by removing from the end of L_1 's words suffix that belongs to L_2 .

$$L_1/L_2 = \{w \mid wu \in L_1 \text{ for some } u \in L_2\}$$

Example. Let

$$L_1 = abaa + aaa,$$

$$L_2 = a + baa,$$

$$L_3 = a^*ba^*.$$

Then

$$L_1/L_2 =$$

$$L_2/L_2 =$$

$$L_3/L_1 =$$

$$L_3/L_3 =$$

$$L_2/L_1 =$$

Our next language operation is called **quotient**.

Let L_1 and L_2 be two languages. Their quotient L_1/L_2 is the language containing all words obtained by removing from the end of L_1 's words suffix that belongs to L_2 .

$$L_1/L_2 = \{w \mid wu \in L_1 \text{ for some } u \in L_2\}$$

Example. Let

$$L_1 = abaa + aaa,$$

$$L_2 = a + baa,$$

$$L_3 = a^*ba^*.$$

Then

$$L_1/L_2 = aba + aa + a$$

$$L_2/L_2 = \varepsilon + ba$$

$$L_3/L_1 = a^* + a^*ba^*$$

$$L_3/L_3 = a^*$$

$$L_2/L_1 = \emptyset$$

Theorem. The family of regular languages is closed under quotient with **arbitrary** languages.

In other words, if L_1 is a regular language, and L_2 any language (not necessarily even regular) then L_1/L_2 is regular.

Proof.

Theorem. The family of regular languages is closed under quotient with **arbitrary** languages.

In other words, if L_1 is a regular language, and L_2 any language (not necessarily even regular) then L_1/L_2 is regular.

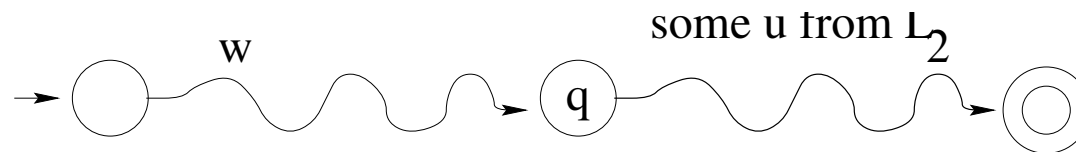
Proof. Let L_1 be a regular language, and A a DFA recognizing L_1 . Let L_2 be an arbitrary language. We show that there is a DFA A' that recognizes the quotient $L' = L_1/L_2$.

Theorem. The family of regular languages is closed under quotient with **arbitrary** languages.

In other words, if L_1 is a regular language, and L_2 any language (not necessarily even regular) then L_1/L_2 is regular.

Proof. Let L_1 be a regular language, and A a DFA recognizing L_1 . Let L_2 be an arbitrary language. We show that there is a DFA A' that recognizes the quotient $L' = L_1/L_2$.

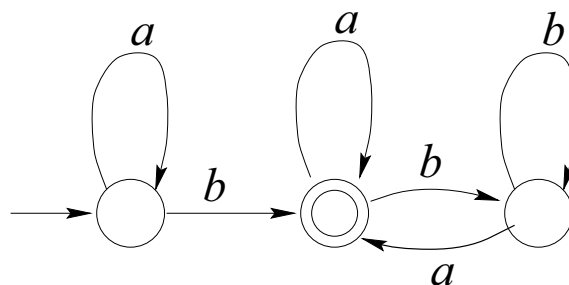
DFA A' has the same states and transitions as A . Also the initial state is the same. Only the final states are different: A state q is made final in A' if and only if some word $u \in L_2$ takes A from state q to some original final state.



Clearly word w is accepted by A' if and only if some wu is accepted by A for some $u \in L_2$. But this is equivalent to saying that w belongs to L_1/L_2 : it is obtained from word $wu \in L_1$ by deleting word $u \in L_2$ from the end. So

$$L(A') = L_1/L_2.$$

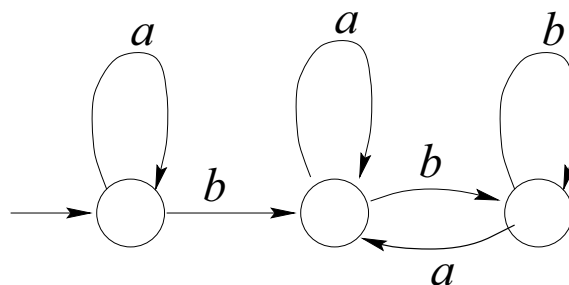
Example. Let L_1 be the regular language recognized by DFA



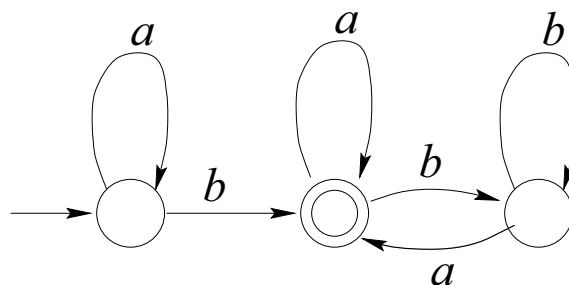
and let

$$L_2 = \{a^n b^n \mid n \geq 0\}.$$

Language L_1/L_2 is recognized by a DFA that has same transitions as A above. We only have to figure out which states are final:



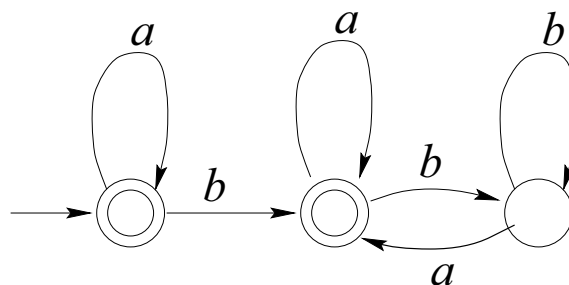
Example. Let L_1 be the regular language recognized by DFA



and let

$$L_2 = \{a^n b^n \mid n \geq 0\}.$$

Language L_1/L_2 is recognized by a DFA that has same transitions as A above. We only have to figure out which states are final:



Remark. Closure under quotient is not effective. If L_2 is some complicated language we may not have any way of determining which states to make final.

However, if both L_1 and L_2 are regular languages then L_1/L_2 can be constructed effectively.

Example. The quotient L/R can also be expressed using homomorphisms, finite substitutions, intersections and concatenations applied to L and R .

Let $L, R \subseteq \Sigma^*$. Let

$$\Sigma' = \{a' \mid a \in \Sigma\}$$

be a disjoint copy of Σ . (There is a' in place of a for each $a \in \Sigma$). For any word $u = a_1 \dots a_n$ let us denote $u' = a'_1 \dots a'_n$.

Define homomorphism h from Σ to Σ' by $h(a) = a'$ for all $a \in \Sigma$. Then for any word $u \in \Sigma^*$ we have $h(u) = u'$. (So **h marks all letters of a word.**)

Example. The quotient L/R can also be expressed using homomorphisms, finite substitutions, intersections and concatenations applied to L and R .

Let $L, R \subseteq \Sigma^*$. Let

$$\Sigma' = \{a' \mid a \in \Sigma\}$$

be a disjoint copy of Σ . (There is a' in place of a for each $a \in \Sigma$). For any word $u = a_1 \dots a_n$ let us denote $u' = a'_1 \dots a'_n$.

Define homomorphism h from Σ to Σ' by $h(a) = a'$ for all $a \in \Sigma$. Then for any word $u \in \Sigma^*$ we have $h(u) = u'$. (So **h marks all letters of a word.**)

Define substitution s from Σ to $\Sigma \cup \Sigma'$ as follows: $s(a) = \{a, a'\}$ for all $a \in \Sigma$. (The substitution **s may mark some letters of a word.**)

$$s(L) \cap \Sigma^* h(R) = \{uv' \mid uv \in L \text{ and } v \in R\}$$

Example. The quotient L/R can also be expressed using homomorphisms, finite substitutions, intersections and concatenations applied to L and R .

Let $L, R \subseteq \Sigma^*$. Let

$$\Sigma' = \{a' \mid a \in \Sigma\}$$

be a disjoint copy of Σ . (There is a' in place of a for each $a \in \Sigma$). For any word $u = a_1 \dots a_n$ let us denote $u' = a'_1 \dots a'_n$.

Define homomorphism h from Σ to Σ' by $h(a) = a'$ for all $a \in \Sigma$. Then for any word $u \in \Sigma^*$ we have $h(u) = u'$. (So h marks all letters of a word.)

Define substitution s from Σ to $\Sigma \cup \Sigma'$ as follows: $s(a) = \{a, a'\}$ for all $a \in \Sigma$. (The substitution s may mark some letters of a word.)

$$s(L) \cap \Sigma^* h(R) = \{uv' \mid uv \in L \text{ and } v \in R\}$$

Define homomorphism g from $\Sigma \cup \Sigma'$ to Σ by $g(a) = a$ and $g(a') = \varepsilon$ for all $a \in \Sigma$. (Now g erases all marked letters.)

$$g(s(L) \cap \Sigma^* h(R)) = L/R.$$

Decision algorithms

In this section we present algorithms for determining whether a given regular language is empty, finite or infinite. We also present an algorithm for determining whether two given regular languages are identical, *i.e.*, whether they contain exactly same words.

First we make the same observation as before: it does not matter in which form the input language is represented. As a regular expression, DFA, NFA or ε -NFA. All representations are effectively equivalent.

Theorem. There is an algorithm to determine if a given regular language is empty.

Theorem. There is an algorithm to determine if a given regular language is empty.

Proof 1. Given a DFA A that recognizes L , check if there exists a path from the initial state to a final state. Such a path exists if and only if $L \neq \emptyset$.

Theorem. There is an algorithm to determine if a given regular language is empty.

Proof 2. Given a DFA A with n states that recognizes L , try all input words $w \in \Sigma^*$ of length $|w| < n$ and check if any of them is accepted by A . By a homework problem, $L \neq \emptyset$ if and only if some w is accepted by A .

(This is a very, very slow algorithm in practice. I gave it here only to show how the homework problem can be used here.)

Theorem. There is an algorithm to determine if a given regular language is empty.

Proof 3. We can also check emptiness directly from a regular expression r , without converting it to a DFA first. Observe that

- If $r = \varepsilon$ or $r = a \in \Sigma$ then $L(r)$ is **not** empty.
- If $r = \emptyset$ then $L(r)$ is empty.
- If $r = r_1 + r_2$ then $L(r)$ is empty if and only if both $L(r_1)$ **and** $L(r_2)$ are empty.
- If $r = r_1 r_2$ then $L(r)$ is empty if and only if $L(r_1)$ **or** $L(r_2)$ is empty.
- If $r = (r_1)^*$ then $L(r)$ is **not** empty. (Even $\emptyset^*$ is non-empty.)

A recursive algorithm based on these facts for the emptiness of $L(r)$:

Empty(r)

Begin

 if $r = \varepsilon$ or $r = a$ for some letter a then return(False)

 if $r = \emptyset$ then return(True)

 if $r = r_1 + r_2$ then return(Empty(r_1) and Empty(r_2))

 if $r = r_1 r_2$ then return(Empty(r_1) or Empty(r_2))

 if $r = r_1^*$ then return(False)

End

Example. Is the language represented by

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

empty ?

Consider then the equivalence problem. Some regular expressions stand for the same language. For example, $a(a + ba)^*$ and $(ab + a)^*a$ both define the same language. How can we determine for any given two regular expressions correctly if their languages are identical ?

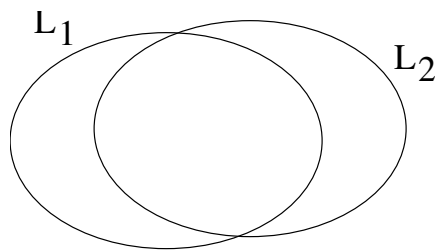
Theorem. There is an algorithm to determine if two regular languages are the same.

Consider then the equivalence problem. Some regular expressions stand for the same language. For example, $a(a + ba)^*$ and $(ab + a)^*a$ both define the same language. How can we determine for any given two regular expressions correctly if their languages are identical ?

Theorem. There is an algorithm to determine if two regular languages are the same.

Proof. Let L_1 and L_2 be two regular languages (represented by finite automata or regular expressions.) Using the effective closure properties proved earlier we can construct a regular expression for their symmetric difference

$$L = (L_1 \setminus L_2) \cup (L_2 \setminus L_1) = (L_1 \cap \bar{L}_2) \cup (L_2 \cap \bar{L}_1).$$



Since $L_1 = L_2$ if and only if L is empty, we can use the algorithm for emptiness to find out whether L_1 and L_2 are the same language.

Our last decision algorithm determines whether given regular language contains finitely or infinitely many words. We know that every finite language is regular:

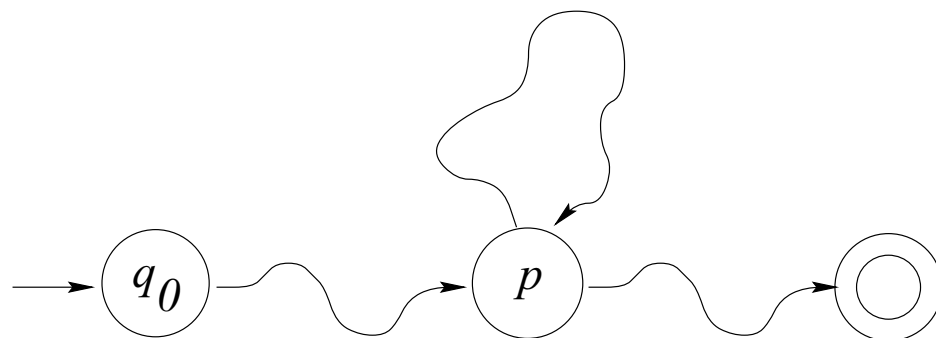
$$\{w_1, w_2, \dots, w_n\} = L(w_1 + w_2 + \dots + w_n).$$

Some regular languages are infinite, for example a^* . How do we determine whether given regular language has infinitely many words ?

Theorem. There is an algorithm to determine if a given regular language is infinite.

Theorem. There is an algorithm to determine if a given regular language is infinite.

Proof 1. Given a DFA A that recognizes L , check if for some state p there exists a path from the initial state to p , a cycle containing p and a path from p to a final state. Such an accepting path containing a cycle exists if and only if L is infinite.



Theorem. There is an algorithm to determine if a given regular language is infinite.

Proof 2. Given a DFA A with n states that recognizes L , try all input words $w \in \Sigma^*$ of length $n \leq |w| < 2n$ and check if any of them is accepted by A . By a homework problem, L is infinite if and only if some w is accepted by A .

(Again, just to show how the homework problem can be used here. Do not try this in practice.)

Theorem. There is an algorithm to determine if a given regular language is infinite.

Proof 3. Testing if a given regular expression r represents an infinite language:

- If $r = \varepsilon$, $r = a \in \Sigma$ or $r = \emptyset$ then $L(r)$ is not infinite.
- If $r = r_1 + r_2$ then $L(r)$ is infinite if and only if $L(r_1)$ or $L(r_2)$ is infinite.
- If $r = r_1 r_2$ then $L(r)$ is infinite if and only if $L(r_1)$ and $L(r_2)$ are non-empty, and at least one of them is infinite.
- If $r = (r_1)^*$ then $L(r)$ is infinite unless $L(r_1)$ is $\emptyset$ or $\{\varepsilon\}$.

A recursive algorithm based on these facts uses also as a subroutine our earlier algorithms for emptiness and equivalence.

Infinite(r)

Begin

 if $r = \emptyset$ or $r = \varepsilon$ or $r = a$ for some letter a then
 return(False)

 if $r = r_1 + r_2$ then
 return(Infinite(r_1) or Infinite(r_2))

 if $r = r_1 r_2$ then
 return((Infinite(r_1) and not Empty(r_2))
 or (Infinite(r_2) and not Empty(r_1)))

 if $r = r_1^*$ then
 return(not Empty(r_1) and not Equal(r_1, ε))

End

Example. An algorithm to test if every word accepted a given DFA A is also accepted by another given DFA B .

Example. An algorithm to test if every word accepted a given DFA A is also accepted by another given DFA B .

Example. An algorithm to test, for a given regular expression r , whether every word is a concatenation of words that match r .

Myhill-Nerode theorem

A relation R on a set S is an **equivalence** relation if it is reflexive, symmetric and transitive, i.e. if

- aRa for all $a \in S$,
- aRb implies bRa , and
- aRb and bRc imply aRc .

Myhill-Nerode theorem

A relation R on a set S is an **equivalence** relation if it is reflexive, symmetric and transitive, i.e. if

- aRa for all $a \in S$,
- aRb implies bRa , and
- aRb and bRc imply aRc .

Example. With $S = \mathbb{Z}$, the set of integers, and n is any fixed positive integer, the congruence

$$a \equiv b \pmod{n}$$

is an equivalence relation. Namely,

- $a \equiv a \pmod{n}$ for all $a \in \mathbb{Z}$,
- if $a \equiv b \pmod{n}$ then $b \equiv a \pmod{n}$, and
- if $a \equiv b \pmod{n}$ and $b \equiv c \pmod{n}$
then $a \equiv c \pmod{n}$.

An equivalence relation R partitions S into disjoint **equivalence classes** such that aRb if and only if a and b are in the same equivalence class. The equivalence class containing a is denoted by $[a]$:

$$[a] = \{b \in S \mid aRb\}.$$

An equivalence relation R partitions S into disjoint **equivalence classes** such that aRb if and only if a and b are in the same equivalence class. The equivalence class containing a is denoted by $[a]$:

$$[a] = \{b \in S \mid aRb\}.$$

Example. The equivalence class containing integer i in the congruence $(\text{mod } n)$ relation is

$$[i] = \{i + kn \mid k \in \mathbb{Z}\}.$$

There are n different equivalence classes of the integer congruence $(\text{mod } n)$:

$$\begin{aligned} [0] &= \{\dots, -2n, -n, 0, n, 2n, \dots\} \\ [1] &= \{\dots, -2n + 1, -n + 1, 1, n + 1, 2n + 1, \dots\} \\ &\dots \quad \dots \\ [n - 1] &= \{\dots, -n - 1, -1, n - 1, 2n - 1, 3n - 1, \dots\} \end{aligned}$$

The **index** of an equivalence relation is the number of equivalence classes. The index can also be infinite.

Every language $L \subseteq \Sigma^*$ defines an equivalence relation R_L on words of Σ^* :

$$w R_L u$$

$$\Updownarrow$$

$$(\forall x \in \Sigma^*) \, wx \in L \text{ if and only if } ux \in L$$

Words w and u are in the relation R_L iff exactly the same extensions x take them to L .

Whether words w and u are in the relation R_L for a given language L depends on for which extensions x words wx and ux belong to L . Let us define the language of **good extensions** x as

$$\text{Ext}(w, L) = \{x \mid wx \in L\}$$

Then $w R_L u$ if and only if

$$\text{Ext}(w, L) = \text{Ext}(u, L).$$

Example. Let $L = aa + baa$. Then

$$\text{Ext}(a, L) =$$

$$\text{Ext}(b, L) =$$

$$\text{Ext}(ba, L) =$$

$$\text{Ext}(\varepsilon, L) =$$

$$\text{Ext}(aa, L) =$$

$$\text{Ext}(baa, L) =$$

$$\text{Ext}(aba, L) =$$

Example. Let $L = aa + baa$. Then

$$\text{Ext}(a, L) = \{a\}$$

$$\text{Ext}(b, L) = \{aa\}$$

$$\text{Ext}(ba, L) = \{a\}$$

$$\text{Ext}(\varepsilon, L) = \{aa, baa\}$$

$$\text{Ext}(aa, L) = \{\varepsilon\}$$

$$\text{Ext}(baa, L) = \{\varepsilon\}$$

$$\text{Ext}(aba, L) = \emptyset$$

Are the following words in the relation R_L ?

a and b ?

a and ba ?

a and ε ?

Example. Let $L = aa + baa$. Then

$$\text{Ext}(a, L) = \{a\}$$

$$\text{Ext}(b, L) = \{aa\}$$

$$\text{Ext}(ba, L) = \{a\}$$

$$\text{Ext}(\varepsilon, L) = \{aa, baa\}$$

$$\text{Ext}(aa, L) = \{\varepsilon\}$$

$$\text{Ext}(baa, L) = \{\varepsilon\}$$

$$\text{Ext}(aba, L) = \emptyset$$

Are the following words in the relation R_L ?

a and b ? No

a and ba ? Yes

a and ε ? No

What is the set of all words that are in relation R_L with word a : $[a] = ?$

What are the equivalence classes of relation R_L ?

Example. Let $L = aa + baa$. Then

$$\text{Ext}(a, L) = \{a\}$$

$$\text{Ext}(b, L) = \{aa\}$$

$$\text{Ext}(ba, L) = \{a\}$$

$$\text{Ext}(\varepsilon, L) = \{aa, baa\}$$

$$\text{Ext}(aa, L) = \{\varepsilon\}$$

$$\text{Ext}(baa, L) = \{\varepsilon\}$$

$$\text{Ext}(aba, L) = \emptyset$$

Are the following words in the relation R_L ?

a and b ? No

a and ba ? Yes

a and ε ? No

What is the set of all words that are in relation R_L with word a : $[a] = \{a, ba\}$

What are the equivalence classes of relation R_L ?

$\{\varepsilon\}, \{a, ba\}, \{aa, baa\}, \{b\}, \{u \mid u \text{ is not a prefix of any word in } L\}$