(Not only) Regular Languages

Discrete Math, Spring 2025

Konstantin Chukharev

Regular Languages

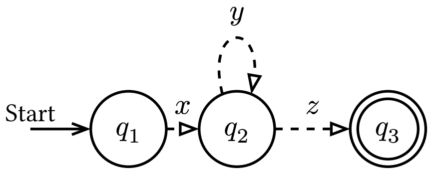
Regular Expressions

Regular languages can be composed from “smaller” regular languages.

- Atomic regular expressions:
  - $\varnothing$, an empty language
  - ε, a singleton language consisting of a single ε word
  - $\mathtt{a}$, a singleton language consisting of a single 1-letter word a, for each $a \in \Sigma$
- Compound regular expressions:
  - $R_1 R_2$, the concatenation of R_1 and R_2
  - $R_1 \mid R_2$, the union of R_1 and R_2
  - $R^* = RRR...$, the Kleene star of R
  - (R), just a bracketed expression
  - Operator precedence: `ab*c|d` $\triangleq ((\mathtt{a}\ (\mathtt{b}^*))\ \mathtt{c}) \mid \mathtt{d}$

Re-visiting States

- Let D be a DFA with n states.
- Any string w accepted by D that has length at least n must visit some state twice.
- Number of states visited is equal to $|w| + 1$.
- By the pigeonhole principle, some state is "duplicated", *i.e.* visited more than once.
- The substring of w between those *revisited states* can be removed, duplicated, tripled, *etc.* without changing the fact that D accepts w.



Informally:

- Let L be a regular language.
- If we have a string $w \in L$ that is "sufficiently long", then we can *split* the string into *three pieces* and *"pump"* the middle.
- We can write $w = xyz$ such that xy^0z, xy^1z, xy^2z, ..., xy^nz, ... are all in L.
  - Notation: y^n means "n copies of y".

Weak Pumping Lemma

Theorem 1 (Weak Pumping Lemma for Regular Languages):

- For any regular language L,
  - There exists a positive natural number n (also called *pumping length*) such that
    - For any $w \in L$ with $|w| \geq n$,
      - There exists strings x, y, z such that
        - For any natural number i,
          - $w = xyz$ (w can be broken into three pieces)
          - $y \neq \varepsilon$ (the middle part is not empty)
          - $xy^i z \in L$ (the middle part can repeated any number of times)

Example: Let $\Sigma = \{0, 1\}$ and $L = \{w \in \Sigma^* \mid w \text{ contains } 00 \text{ as a substring}\}$. Any string of length 3 or greater can be split into three parts, the second of which can be "pumped".

Example: Let $\Sigma = \{0, 1\}$ and $L = \{\varepsilon, 0, 1, 00, 01, 10, 11\}$. The weak pumping lemma still holds for finite languages, because the pumping length n can be longer than the longest word in the language!

Testing Equality

Definition 1: The *equality problem* is, given two strings x and y, to decide whether $x = y$.

Example: Let $\Sigma = \{0, 1, \#\}$. We can *encode* the equality problem as a string of the form $x \ \# \ y$.

- "Is *001* equal to *110* ?" would be $001 \ \# \ 110$.
- "Is *11* equal to *11* ?" would be $11 \ \# \ 11$.
- "Is *110* equal to *110* ?" would be $110 \ \# \ 110$.

Let $\mathrm{EQUAL} = \{w \ \# \ w \mid w \in \{0, 1\}^*\}$.

Question: Is EQUAL a *regular* language?

A typical word in EQUAL looks like this: $001 \ \# \ 001$.

- If the "middle" piece is just a symbol $\#$, then observe that $001\,001 \notin \mathrm{EQUAL}$.
- If the "middle" piece is either completely to the left or completely to the right of $\#$, then observe that any duplication or removal of this piece is not in EQUAL.
- If the "middle" piece includes $\#$ and any symbols from the left/right of it, then, again, observe that any duplication or removal of this piece is not in EQUAL.

Testing Equality [2]

Theorem 2: EQUAL is not a regular language.

Proof: By contradiction. Assume that EQUAL is a regular language.

Let n be the pumping length guaranteed by the weak pumping lemma. Let $w = 0^n \ \# \ 0^n$, which is in EQUAL and $|w| = 2n + 1 \geq n$. By the weak pumping lemma, we can write $w = xyz$ such that $y \neq \varepsilon$ and for any $i \in \mathbb{N}$, $xy^i \ \# \ z \in$ EQUAL. Then y cannot contain $\#$, since otherwise if we let $i = 0$, then $xy^0 \ \# \ z = x \ \# \ z$ does not contain $\#$ and would not be in EQUAL. So y is either completely to the left of $\#$ or completely to the right of $\#$.

Let $|y| = k$, so $k > 0$. Since y is completely to the left or right of $\#$, then $y = 0^k$.

Now, we consider two cases:

Case 1: y is to the left of $\#$. Then $xy^2z = 0^{n+k} \ \# \ 0^n \notin$ EQUAL, contradicting the weak pumping lemma.

Case 2: y is to the right of $\#$. Then $xy^2z = 0^n \ \# \ 0^{n+k} \notin$ EQUAL, contradicting the weak pumping lemma.

Testing Equality [3]

In either case we reach a contradiction, so our assumption was wrong. Thus, EQUAL *is not regular*. $\square$

Non-regular Languages

(Not only) Regular Languages

- The weak pumping lemma describes a property common to *all* regular languages.
- Any language L which does not have this property *cannot be regular*.
- What other languages can we find that are not regular?

Example: Consider the language $L = \{0^n 1^n \mid n \in \mathbb{N}\}$.

- $L = \{\varepsilon, 01, 0011, 000111, 00001111, \ldots\}$
- L is a classic example of a non-regular language.
- **Intuitively:** if you have *only finitely many states* in a DFA, you cannot *"remember"* an arbitrary number of 0s to match *the same* number of 1s.

How would we prove that L is non-regular?

Use the Pumping Lemma to show that L *cannot* be regular.

Pumping Lemma as a Game

The weak pumping lemma can be thought of as a *game* between **you** and an **adversary**.

- **You win** if you can prove that the pumping lemma *fails*.
- **The adversary wins** if the adversary can make a choice for which the pumping lemma *succeeds*.

The game goes as follows:

- The adversary chooses a pumping length n.
- You choose a string w with $|w| \geq n$ and $w \in L$.
- The adversary breaks it into x, y, and z.
- You choose an i such that $xy^iz \notin L$ *(if you can't, you lose!)*.

Pumping Lemma as a Game [2]

$$L = \{0^n 1^n \mid n \in \mathbb{N}\}$$

Adversary	**You**
Maliciously choose pumping length n	
	Cleverly choose a string $w \in L$, $\lvert w\rvert \geq n$
Maliciously split $w = xyz$, $y \neq \varepsilon$	
	Cleverly choose an i such that $xy^iz \notin L$
Lose	Win

$\{0^n 1^n\}$ is **not** regular

Formal Proof of Non-regularity

Theorem 3: $L = \{0^n 1^n \mid n \in \mathbb{N}\}$ is not regular.

Proof: By contradiction. Assume that L is regular.

Let n be the pumping length guaranteed by the weak pumping lemma (“there exists n…”). Consider the string $w = 0^n 1^n$. Then $|w| = 2n \geq n$ and $w \in L$, so we can write (split) $w = xyz$ such that $y \neq \varepsilon$ and for any $i \in \mathbb{N}$, we have $xy^i z \in L$.

We consider three cases:
Case 1: y consists solely of 0s. Then $xy^0 z = xz = 0^{n-|y|} 1^n$, and since $|y| > 0$, $xz \notin L$.
Case 2: y consists solely of 1s. Then $xy^0 z = xz = 0^n 1^{n-|y|}$, and since $|y| > 0$, $xz \notin L$.
Case 3: y consists of $k > 0$ 0s followed by $m > 0$ 1s. Then $xy^2 z = 0^n 1^m 0^k 1^n$, so $xy^2 z \notin L$.

In all three cases we reach a contradiction, so our assumption was wrong and L is not regular. $\square$

Pumping Lemma

Pumping

Consider the language L over $\Sigma = \{0, 1\}$ of strings $w \in \Sigma^*$ that contain *an equal number* of 0s and 1s.

For example:

- `01` in L
- `11011` not in L
- `110010` in L

Question: Is L a *regular* language?

Let's *use* the weak pumping lemma to show it is by *pumping all the strings* in this language.

Proof *(incorrect)*: We are going to show that L satisfies the conditions of the weak pumping lemma. Let $n = 2$. Consider any string $w \in L$ (*i.e.*, w contains the same number of 0s and 1s) with $|w| \geq 2$.

We can split $w = xyz$ such that $x = z = \varepsilon$ and $y = w$, so $y \neq \varepsilon$. Then, for any natural number $i \in \mathbb{N}$, $xy^i z = w^i$, which has the same number of 0s and 1s.

Since L passes the conditions of the weak pumping lemma, L is regular. $\square$

A word of Caution

- The weak and full pumping lemmas describe the *necessary* condition of regular languages.
  - If L is *regular*, then it *passes* the conditions of the pumping lemma.
  - If a language *fails* the pumping lemma, it is *definitely not regular*.
- The weak and full pumping lemmas are *not a sufficient* condition of regular languages.
  - If L is *not regular*, then it still *may pass* the conditions of the pumping lemma.
  - If a language *passes* the pumping lemma, we *learn nothing* about whether it is regular or not.

The Stronger Pumping Lemma

The language L *can* be proven to be *non-regular* using a *stronger version* of the pumping lemma.

For the intuition behind the "full" pumping lemma, let's revisit our original observation.

- Let D be a DFA with n states.
- Any string w accepted by D of length at least n must visit some state twice *within its first n* symbols.
  - The number of visited states is equal to $n + 1$.
  - By the pigeonhole principle, some state is *duplicated*.
- The substring of w between those *revisited states* can be removed, duplicated, tripled, *etc.* without changing the fact that D accepts w.

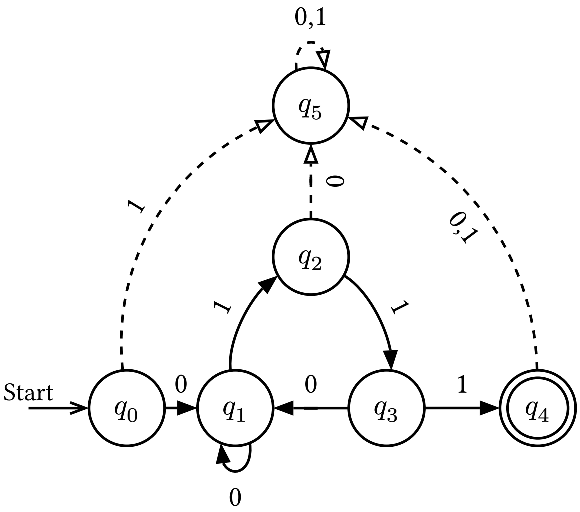
Overall, we can add the following condition to the weak pumping lemma:

$$|xy| \leq n$$

This restriction means that we can limit where the string to pump must be. If we specifically choose the first n characters of the string to pump, we can ensure y (middle part) to have a specific property.

We can then show that y cannot be pumped arbitrarily many times.

The Stronger Pumping Lemma [2]



$$q_0 \xrightarrow{0} q_1 \xrightarrow{1} q_2 \xrightarrow{1} q_3 \xrightarrow{0} q_1 \xrightarrow{1} q_2 \xrightarrow{1} q_3 \xrightarrow{0} q_1 \xrightarrow{1} q_2 \xrightarrow{1} q_3 \xrightarrow{0} q_1 \xrightarrow{1} q_2 \xrightarrow{1} q_3 \xrightarrow{1} q_4$$

Formal Proof of Non-regularity

Theorem 4: $L = \{w \in \{0,1\}^* \mid w \text{ has an equal number of 0s and 1s}\}$ is *not regular*.

Proof: By contradiction. Assume that L is regular.

Let n be the pumping length guaranteed by the weak pumping lemma. Consider the string $w = 0^n 1^n$. Then $|w| = 2n \geq n$ and $w \in L$. Therefore, there exist strings x, y, and z such that $w = xyz$, $|xy| \leq n$, $y \neq \varepsilon$, and for any $i \in \mathbb{N}$, we have $xy^i z \in L$.

Since $|xy| \leq n$, y must consist solely of 0s. But then $xy^2z = 0^{n+|y|}1^n$, and since $|y| > 0$, $xy^2z \notin L$.

We have reached a contradiction, so our assumption was wrong and L is not regular. □

Summary of the Pumping Lemma

1. Using the *pigeonhole principle*, we can prove the weak and full *pumping lemma*.
2. These lemmas describe essential properties of the *regular* languages.
3. Any language that *fails* to have these properties *can not be regular*.

Closure Properties of Regular Languages

Closure of Regular Languages

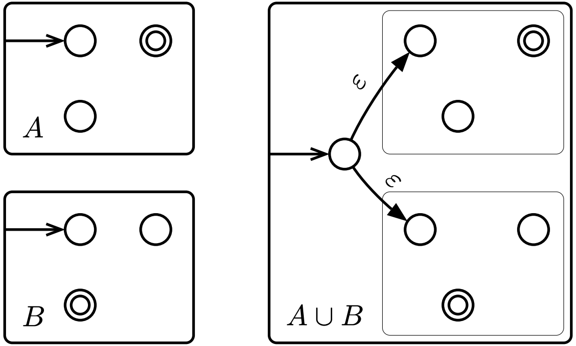
1. The *union* of two regular languages is regular.
2. The *intersection* of two regular languages is regular.
3. The *complement* of a regular language is regular.
4. The *difference* of two regular languages is regular.
5. The *reversal* of a regular language is regular.
6. The *Kleene star* of a regular language is regular.
7. The *concatenation* of regular languages is regular.
8. A *homomorphism* (substitution of strings for symbols) of a regular language is regular.
9. The *inverse homomorphism* of a regular language is regular.

Closure under Union

Theorem 5: If L and M are regular languages, then so is their union $L \cup M$.

Proof: Since L and M are regular, they have regular expressions, *i.e.* $L = \mathcal{L}(R)$ and $M = \mathcal{L}(S)$.

Then $L \cup M = \mathcal{L}(R + S)$ by the definition of the union $(+)$ operator for regular expressions. $\square$

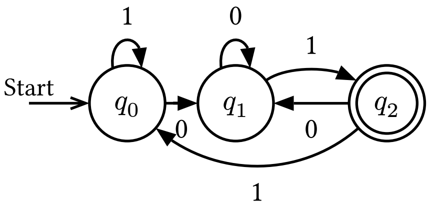


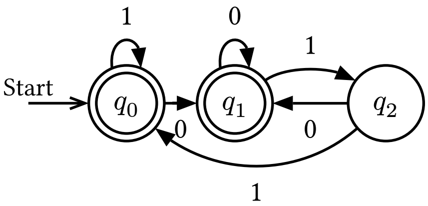
Closure under Complement

Theorem 6: If L is a regular language over the alphabet Σ, then its complement $\overline{L} = \Sigma^* - L$ is also a regular language.

Proof: Let $L = \mathcal{L}(A)$ for some DFA $A = (Q, \Sigma, \delta, q_0, F)$. Then $\overline{L} = \mathcal{L}(B)$, where B is the DFA $(Q, \Sigma, \delta, q_0, Q - F)$. That is, B is exactly like A, but with the accepting states flipped. Then w is in $\overline{L}$ if and only if $\delta(q_0, w)$ is in $Q - F$, which occurs if and only if w is not in $\mathcal{L}(A)$. $\square$

Example: The DFA A presented below on the left accepts only the strings of 0′s and 1′s that end in `01`, $\mathcal{L}(A) =$ `(0+1)*01`. The complement of $\mathcal{L}(A)$ is therefore all strings of 0′s and 1′s that *do not* end in `01`. Below on the right is the automaton for $\{0, 1\}^* - \mathcal{L}(A)$.





Closure under Intersection

Theorem 7: If L and M are regular languages, then so is their intersection $L \cap M$.

Proof *(simple)*: $L \cap M = \overline{\overline{L} \cup \overline{M}}$. $\square$

Proof: We can directly construct a "product" DFA for the intersection of two regular languages.

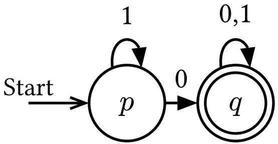
Let L and M be the languages of automata $A_L = (Q_L, \Sigma, \delta_L, q_L, F_L)$ and $A_M = (Q_M, \Sigma, \delta_M, q_M, F_M)$. Note that we assume that the alphabets of both automata are the same (or Σ is their union).

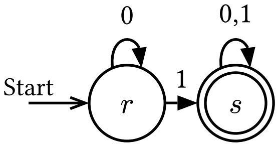
For $L \cap M$, we construct the automaton A that simulates both A_L and A_M. The states of A are the product of the states of A_L and A_M. The initial state is (q_L, q_M), and the accepting states are $F_L \times F_M$. The transitions are defined as $\delta(\langle p, q\rangle, c) = \langle \delta_L(p, c), \delta_M(q, c)\rangle$.

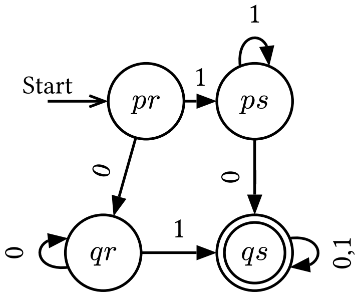
To see why $\mathcal{L}(A) = \mathcal{L}(A_L) \cap \mathcal{L}(A_M)$, first observe that $\hat{\delta}(\langle q_L, q_M\rangle, w) = \langle \hat{\delta}_L(q_L, w), \hat{\delta}_M(q_M, w)\rangle$. But A accepts w if and only if $\hat{\delta}(q_0, w)$ is in $F_L \times F_M$, which occurs if and only if $\hat{\delta}_L(q_L, w)$ is in F_L and $\hat{\delta}_M(q_M, w)$ is in F_M. Or rather, A accepts w if and only if both A_L and A_M accept w. Thus, A accepts the intersection of L and M. $\square$

Closure under Intersection [2]

Example: The first automaton on the left accepts all strings that *have a 0*. The second automaton in the middle accepts all strings that *have a 1*. On the right, we show the *product* of these two automata. Its states are labelled by the pairs of states of the two automata. It is easy to see that this automaton accepts the *intersection* of the two languages: all strings that *have both a 0 and a 1*.







Closure under Difference

Theorem 8: If L and M are regular languages, then so is their difference $L - M$.

Proof: Observe that $L - M = L \cap \overline{M}$. By previous theorems, $\overline{M}$ is regular, and $L \cap \overline{M}$ is also regular. $\square$

Closure under Reversal

Definition 2: The *reversal* of a string $w = a_1 a_2 \ldots a_n$ is the string $w^R = a_n a_{n-1} \ldots a_1$.

Example: $0010^R = 0100$ and $\varepsilon^R = \varepsilon$.

Definition 3: The *reversal* of a language L is the language $L^R = \{w^R \mid w \in L\}$.

Example: Let $L = \{001, 10, 111\}$, then $L^R = \{001^R, 10^R, 111^R\} = \{100, 01, 111\}$.

Theorem 9: If L is a regular language, then so its reversal L^R.

Proof: Assume L is defined by regular expression E. The proof is a structural induction on the size of E. We show that there is another regular expression E^R such that $\mathcal{L}(E^R) = (\mathcal{L}(E))^R$, that is, the language of E^R is the reversal of the language of E.

Basis: If E is ε, $\emptyset$, or $\mathtt{a}$ for some symbol a, then E^R is the same as E.

Closure under Reversal [2]

Induction: There are three cases, depending on the form of E.

1. $E = E_1 + E_2$. Then $E^R = E_1^R + E_2^R$.

   The justification is that the reversal of the union of two languages is obtained by computing the reversals of the two languages and taking the union of those languages.

2. $E = E_1 E_2$. Then $E^R = E_2^R E_1^R$. Note that we reverse the order of the two languages, as well as reversing the languages themselves. For example, if $\mathcal{L}(E_1) = \{0, 1111\}$ and $\mathcal{L}(E_2) = \{00, 10\}$, then $\mathcal{L}(E_1 E_2) = \{0100, 0110, 11100, 11110\}$. The reversal of the latter language is

   $$\{0010, 0110, 00111, 01111\}$$

   If we concatenate the reversals of $\mathcal{L}(E_2)$ and $\mathcal{L}(E_1)$, we get

   $$\{00, 01\}\{10, 111\} = \{0010, 00111, 0110, 01111\}$$

   which is the same language as $(\mathcal{L}(E_1 E_2))^R$. In general, if a word w in $\mathcal{L}(E)$ is the concatenation of w_1 from $\mathcal{L}(E_1)$ and w_2 from $\mathcal{L}(E_2)$, then $w^R = w_2^R w_1^R$.

Closure under Reversal [3]

3. $E = E_1^*$. Then $E^R = \left(E_1^R\right)^*$.

   The justification is that any string w in $\mathcal{L}(E)$ can be written as $w_1 w_2 \ldots w_n$, where each w_i is in $\mathcal{L}(E_1)$. Then $w^R = w_n^R w_{n-1}^R \ldots w_1^R$. Each w_i^R is in $\mathcal{L}(E^R)$, so w^R is in $\mathcal{L}\left(\left(E_1^R\right)^*\right)$.

   Conversely, any string in $\mathcal{L}\left(\left(E_1^R\right)^*\right)$ is of the form $w_1 w_2 \ldots w_n$, where each w_i is the reversal of a string in $\mathcal{L}(E_1)$. The reversal of this string, $w_n^R w_{n-1}^R \ldots w_1^R$, is therefore a string in $\mathcal{L}(E_1^*)$, which is $\mathcal{L}(E)$.

   We have thus shown that a string is in $\mathcal{L}(E)$ if and only if its reversal is in $\mathcal{L}\left(\left(E_1^R\right)^*\right)$.

$\square$

Example: Let L be defined by the regular expression `(0+1)0*`. Then L^R is the language of $(\texttt{0*})^R(\texttt{0+1})^R$.

If we apply the rules for Kleene star and union to the two parts, and then apply the basis rule that says the reversals of `0` and `1` are unchanged, we find that L^R has regular expression `0*(0+1)`.

Decision Properties of Regular Languages

Fundamental Questions about Languages

1. Is the language *empty*?
2. Is the language *finite*?
3. Is the particular string w *in* the language?
4. Is the language a *subset* of another language?
5. Are the two languages *equivalent*?

Decision Procedures

Converting among representations

- ε-closure: $O(n^3)$
- ε-NFA to DFA: $n^3 2^n$
- DFA to ε-NFA: $O(n)$
- ε-NFA to RegEx: $O(n^3 4^n)$
- RegEx to ε-NFA: $O(n)$

Testing *emptiness* of a regular language

- Given an automaton, we can determine whether the accepting states are reachable, in $O(n^2)$ time.
- Given a regular expression, we can construct an ε-NFA and then determine the reachability of the accepting states, in $O(n)$ time. Alternatively, we can inspect the regex directly.

Testing *membership* in a regular language

- Given an automaton with s states and a string w of size n, we can simulate the automaton for w to determine whether it accepts w.
  - For DFA, this can be done in $O(n)$ time.
  - For NFA or ε-NFA, in $O(ns^2)$.

Emptiness, Finiteness, Infiniteness

Theorem 10: The language L accepted by a finite automaton with n states is *non-empty* iff the finite automaton accepts a word of length less than n.

Theorem 11: The language L accepted by a finite automaton M with n states is *infinite* iff the automaton accepts some word of length l, where $n \leq l < 2n$.

Proof: If w is in $\mathcal{L}(M)$ and $n \leq |w| < 2n$, then by the Pumping lemma, $\mathcal{L}(M)$ is infinite. That is, $w = xyz$, and for all i, xy^iz is in L. Conversely, if $\mathcal{L}(M)$ is infinite, then there exists w in $\mathcal{L}(M)$, where $|w| \geq n$. If $|w| < 2n$, we are done. If no word is of length between n and $2n - 1$, let w be of length at least $2n$, but as short as any word in $\mathcal{L}(M)$ whose length is greater than of equal to $2n$. Again by the Pumping lemma, we can write $w = xyz$ with $1 \leq |y| \leq n$ and $xz \in \mathcal{L}(M)$. Either w was not the shortest word of length $2n$ or more, or $|xz|$ is between n and $2n - 1$, a contradiction in either case. □