

COMPUTATION: DAY 4

BURTON ROSENBERG
UNIVERSITY OF MIAMI

Wednesday, February 4, 2026.

1. REGULAR EXPRESSIONS

1.1 A compact way to describe certain regular sets is with *regular expressions*. Regular expressions are widely used in programming, with libraries and packages that support them for text processing; for example, the Perl language has regular expressions built in. The syntax is simple yet powerful, and part of their usefulness is that, in fact, every regular language can be described by a regular expression. This fact is the content of Kleene's theorem.

1.2 The syntax of a regular expression consists of all symbols in an alphabet Σ of self-interpreted symbols, together with two additional symbols that are interpreted as the empty set and the empty string. It also includes operation symbols for union and Kleene star, explicit syntax for concatenation, and parentheses for grouping.

1.3 Formally, the set of regular expressions over an alphabet Σ is defined recursively as follows:

- (1) $\emptyset$ and ε are regular expressions.
- (2) For each symbol $a \in \Sigma$, a is a regular expression.
- (3) If R and S are regular expressions, then
 - (a) $(R + S)$ is a regular expression (union),
 - (b) (RS) is a regular expression (concatenation),
 - (c) (R^*) is a regular expression (Kleene star).

1.4 These are all the rules for constructing regular expressions. Parentheses may later be omitted according to standard rules of operator precedence, for convenience in writing and communication: Kleene star has the highest precedence, concatenation comes next, and union has the lowest precedence. These conventions are natural, by analogy with the familiar rules for exponentiation, multiplication, and addition in arithmetic.

1.5 For every regular expression R over an alphabet Σ , there exists a nondeterministic finite automaton N_R such that

$$\mathcal{L}(N_R) = R,$$

where $\mathcal{L}(N_R)$ is the set of strings accepted by the automaton, and R is understood as the language described by the regular expression. This is an abuse of notation: formally, R is a syntactic object, but here we are treating it as a subset of Σ^* .

1.6 For instance, we previously gave an NFA N for the regular expression a^*b ,

$$\mathcal{L}(N) = \{ a^n b \mid n \geq 0 \} = a^*b.$$

Theorem 1.1. A regular expression describes a regular language. In other words, for every regular expression R , there exists a nondeterministic finite automaton N_R such that

$$\mathcal{L}(N_R) = R.$$

Proof: There exist NFAs that accept a specific single-letter string. Regular sets form a Kleene algebra, so there are NFAs that accept the empty set $\emptyset$ and the empty string ε .

By induction on the structure of regular expressions, and using the closure properties of Kleene algebras, if there exist NFAs accepting languages A and B , then there are NFAs accepting $A + B$, AB , and A^* .

Hence, for every regular expression R , there exists an NFA N_R such that $\mathcal{L}(N_R) = R$, proving that every regular expression describes a regular language. $\square$

2. THE GENERALIZED NFA

2.1 To show that any language recognized by an NFA can be written as a Regular Expression, we generalize the NFA so that the edges can be regular expressions. To follow an edge with a regular expression is consume the length of input that matches the regular expression. There are several more rules, that are of a technical nature.

2.2 The rules for the transition function include certain technical restrictions that simplify the proof procedure that follows. We highlight these details here, together with the reasons they are imposed.

- (1) The proof proceeds by removing intermediate states—those that are neither the initial state $q_{initial}$ nor the accepting state q_{accept} —without changing the language accepted by the machine. For this reason, the transition function δ does not allow transitions into $q_{initial}$ or transitions out of q_{accept} , and q_{accept} is the unique accepting state.

- (2) For every ordered pair of states, except those leading into $q_{initial}$ and those leaving q_{accept} , there is a regular expression labeling the transition. The regular expression $\emptyset$ is used to indicate that the machine can never take that transition.
- (3) We emphasize that even self-loops at states are included in the domain of δ and are labeled by regular expressions.

2.3 With these preparations, we describe the systematic removal of intermediate states from a GNFA in a way that preserves the language accepted. This process is repeated until no intermediate states remain, at which point the automaton consists of a single regular expression that denotes the language of the original GNFA.

Definition 2.1 (Generalized NFA). A *Generalized NFA* (GNFA) is the 5-tuple

$$G = \langle Q, \Sigma, \delta, q_{initial}, q_{accept} \rangle$$

where,

- (1) Q is a finite state of states,
- (2) Σ is a finite set of symbols (said letters),
- (3) q_{accept} and $q_{initial}$ are distinct states from Q ,
- (4) $\delta : (Q \setminus q_{accept}) \times (Q \setminus q_{initial}) \longrightarrow \mathcal{R}$ is a map from state pairs to regular expressions over the alphabet Σ .

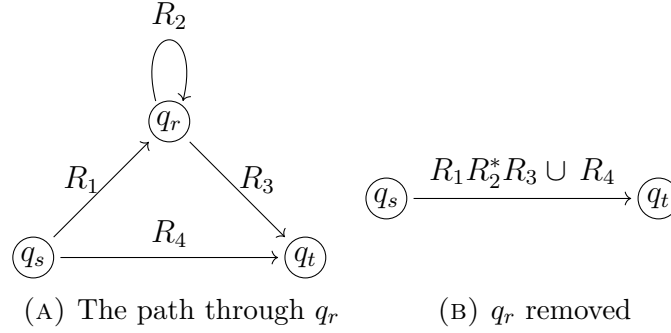
Definition 2.2. A *computation* of a GNFA G on $\sigma \in \Sigma^*$ is a decomposition of $\sigma = w_1 w_2 \dots w_k$ such a path of states exists from $q_{initial}$ to q_{accept} where the regular expression label R_i on the i -th state transition matches w_i . The language of the GNFA, $\mathcal{L}(G)$, is the set of all strings $\sigma \in \Sigma^*$ such that there exists a computation for that string.

Theorem 2.1 (Kleene's Theorem¹). The class of languages denoted by regular expressions is exactly the class of regular languages.

Proof. We prove the two directions separately.

First, given a regular expression $\mathcal{R}$, we inductively construct an NFA N that accepts the same language as $\mathcal{R}$. This construction follows the syntactic structure of $\mathcal{R}$, applying the union, concatenation, and Kleene star constructions to automata

¹Stephen C. Kleene, *Representation of Events in Nerve Nets and Finite Automata*, in *Automata Studies*, Annals of Mathematics Studies, no. 34, Princeton University Press, 1956 (originally 1951).

FIGURE 1. Joining path through q_r into the path q_s to q_t

corresponding to subexpressions. Thus every regular expression denotes a regular language.

Conversely, given an NFA N , we construct an equivalent regular expression $\mathcal{R}$. We first convert N into an equivalent generalized nondeterministic finite automaton (GNFA) G in the standard way. We then repeatedly eliminate intermediate states using the state-removal construction, without changing the language accepted. Figure 1 illustrates the removal of a state q_r and shows how, for each remaining pair of states (q_s, q_t) (including the case $q_s = q_t$), the label on the transition from q_s to q_t is updated.

This process continues until only the initial and accepting states remain, at which point the resulting transition label is a regular expression that denotes the language accepted by N . $\square$

3. NON-REGULAR LANGUAGES

3.1 Consider the language

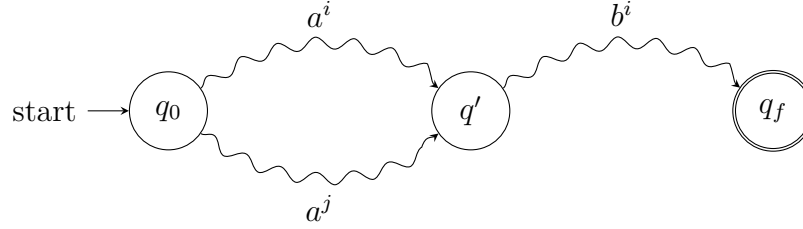
$$L_o = \{ a^i b^i \mid i \geq 1 \}$$

It cannot be regular.

Proof: Consider a very long string in L_o . Fix a machine M which claims to recognize L_o ,

$$\mathcal{L}(M) = L_o.$$

It has a certain number of states p . We will force the machine to revisit a state by creating a very long sequence of a 's. Then we will exploit that to show the machine cannot function as claimed.

FIGURE 2. The language $a^i b^i$ is non-regular.

As the machine processes successive symbols it must at some point repeat a state. There may be many loops, or loops inside the loop we are considering, but this does not matter. There are a state q for which both the strings a^i and a^{i+j} both leave the machine in state q .

Because the language is infinite, and a^{i+j} can be a prefix to a string in the language, let us continue the string as needed to arrive at a string,

$$a^i a^j a^k b^{i+j+k} \in \mathcal{L}(M).$$

This means the string $a^k b^{i+j+k}$ leads from q to an accepting state. The sequence a^i also leads M to state q , so this string must also be accepted by the machine.

$$a^i a^k b^{i+j+k}$$

is accepted by the machine, but that string is not in the language.

We demonstrated that this machine M will not correctly recognize the language, and that the argument holds for an M that we propose. Hence there is no FA recognizing the language, and it is not regular. $\square$

4. THE PUMPING LEMMA

4.1 The argument above demonstrates a property of infinite regular languages. If a language can be recognized by a FA, all sufficiently long strings must cause the machine to repeat states, and the section of the string that causes the state sequence to loop can be pumped, that is, removed or repeated. And if the language is regular, and this is the proper machine recognizing the language, then all of those pumped strings must be in the language.

4.2 This is formalized in the pumping lemma which essentially says: *If A is an infinite regular language, every sufficiently long string in A can be pumped.*

Theorem 4.1 (The pumping lemma). If A is an infinite regular language, then there exists an integer p , such that for every $s \in A$ with $|s| \geq p$, the string s can be written

as

$$s = xyz$$

where $|xy| \leq p$, and $|y| > 0$ and for all $i \geq 0$, $xy^iz \in A$.

4.3 To use the pumping lemma to show a language is not regular,

- (1) Cleverly pick a long enough s in the language.
- (2) Consider every possible way that $s = xyz$, according to the lemma.
- (3) For each such xyz , show an i such that xy^iz is not in the language.

Then we have demonstrated that the language does not have a property that all regular languages must have, and therefore it is not regular.

4.4 This is not an easy theorem to make use of. We are looking at a black box from the outside and what we know is that for a long enough string there is a loop. We do not know exactly what long enough means, and we have no idea what the loop might be. When we write $s = xyz$, we have to be sure not to assume much about y .

4.5 Consider our argument that shows a^ib^i is not regular. Since i has no bound, and then the loop has no choice but to be of the form aa^* , these difficulties are addressed.

Theorem 4.2. The language $S \subseteq \{a, b\}^*$ of all strings σ such the number of a symbols equals the number of b symbols is not regular.

Proof: The language a^*b^* is regular. On the assumption that S is regular so is $S \cap a^*b^*$. This is the set a^ib^i which we have shown is not regular. $\square$

4.6 Do not conclude that it is enough that we have equal number of letters for a language to be non regular. The set $(ab)^*$ is regular and the number of a 's equals the number of b 's.

4.7 Do not conclude that just because a language has the pumping property, that it is regular. The theorem begins with "if A is an infinite regular language". If it is not regular, the rest of the theorem has no relevance. Here is an example language,

$$\{a^ib^jc^k \mid j = k \text{ if } i > 0\}$$

It can be pumped. Any long enough string either begins with an a , and if so that is what we pump, or else does not begin with an a , then we pump on either a b or a c . The resulting strings of the pumping are all in the language. However it is not regular, as demonstrated by intersecting it with the regular language ab^*c^* .