Membership Properties for Regular Languages

5DV037 — Fundamentals of Computer Science
Umeå University
Department of Computing Science

Stephen J. Hegner
hegner@cs.umu.se
http://www.cs.umu.se/~hegner

RE-Based Closure Properties

Notation: Recall that $\mathrm{RegLang}(\Sigma)$ denotes the set of all regular languages over the alphabet Σ.

Theorem: The class of regular languages over Σ is closed under union, concatenation, and Kleene star. More precisely, given $L_1, L_2 \in \mathrm{RegLang}(\Sigma)$, the following languages are also in $\mathrm{RegLang}(\Sigma)$.

- $L_1 \cup L_2$
- $L_1 \cdot L_2$
- ${L_1}^*$

Proof: Based upon the closure of regular expressions under the corresponding operations. $\square$

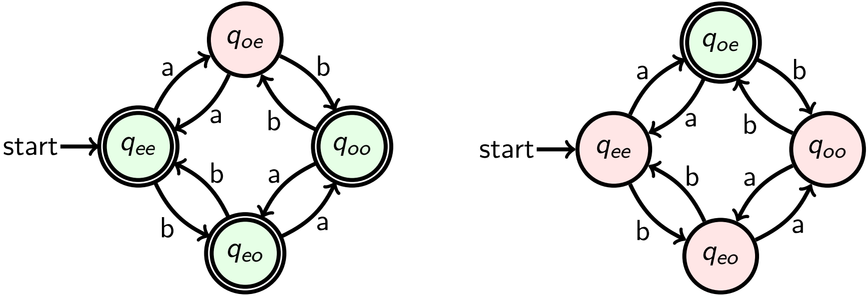
Closure under Complement

- Recall: the *complement* of the language L (relative to Σ) is $\overline{L} = \Sigma^* \setminus L$.

Theorem: The class of regular languages over Σ is closed under complement with respect to Σ.

Proof: Let $M = (Q, \Sigma, \delta, q_0, F)$ be a DFA with $\mathcal{L}(M) = L$, and let $M' = (Q, \Sigma, \delta, q_0, Q \setminus F)$. Then $\mathcal{L}(M') = \overline{L}$. $\square$

Example: The machine on the right accepts the complement of the language of the machine on the left.



- Note that the machine must be *deterministic* for this construction to work.

Closure under Intersection

Theorem: The class of regular languages over Σ is closed under intersection with respect to Σ. More precisely, if $M_1, M_2 \in \text{RegLang}(\Sigma)$, then $L_1 \cap L_2 \in \text{RegLang}(\Sigma)$ as well.

Proof 1: Use De Morgan's law

$$L_1 \cap L_2 = \overline{(\overline{L_1} \cup \overline{L_2})}$$

and the fact that $\text{RegLang}(\Sigma)$ is closed under union and complement. $\square$

Proof 2: Construct an accepter directly which runs accepters for each language in parallel. Let $M_1 = (Q_1, \Sigma, \delta_1, q_{0_1}, F_1)$ and $M_2 = (Q_2, \Sigma, \delta_2, q_{0_2}, F_2)$ be DFAs with $\mathcal{L}(M_1) = L_1$ and $\mathcal{L}(M_2) = L_2$. Define

$$M_1 \times M_2 = (Q_1 \times Q_2, \Sigma, \delta_1 \times \delta_2, (q_{o_1}, q_{0_2}), F_1 \times F_2)$$

with $(\delta_1 \times \delta_2) : ((q_1, q_2), a) \mapsto (\delta_1(q_1, a), \delta_2(q_2, a))$. Then $\mathcal{L}(M_1 \times M_2) = L_1 \cap L_2$. $\square$

- Proof 2 works only for DFAs!

Closure under other Set Operations

- There are two other set operations which will be of use in that which follows.

Observation: Let $L_1, L_2 \in \text{RegLang}(\Sigma)$. Then $L_1 \setminus L_2 \in \text{RegLang}(L)$ as well.

Proof: $L_1 \setminus L_2 = L_1 \cap \overline{L_2}$; use closure under intersection and complement . $\square$

- Let $L_1, L_2 \in \Sigma^*$. Define the *symmetric difference* of L_1 and L_2 to be $L_1 \triangle L_2 = (L_1 \setminus L_2) \cup (L_2 \setminus L_1)$.

Observation: IF $L_1, L_2 \in \text{RegLang}(\Sigma)$, so too is $L_1 \triangle L_2$.

Proof: Use the above result on closure under set difference, together with closure under union. $\square$

Closure under Homomorphism

- Let Σ and Σ' be alphabets. A *homomorphism* from Σ to Σ' a function $h : \Sigma \rightarrow \Sigma'^*$.
- A homomorphism extends to strings in a natural way:

$$h(a_1 a_2 \ldots a_k) = h(a_1) \cdot h(a_2) \cdot \ldots \cdot h(a_n)$$

- And to languages:

$$h(L) = \{h(\alpha) \mid \alpha \in L\}$$

Theorem: The set of regular languages over Σ is closed under homomorphism to a second alphabet Σ'

Proof outline: Appeal to substitution in REs. See the textbook for details. $\square$

Closure under Right Quotient

- Let L_1 and L_2 be languages over the same alphabet Σ. The *right quotient* of L_1 with L_2 is

$$L_1/L_2 = \{\alpha \in \Sigma^* \mid (\exists \beta \in L_2)(\alpha \cdot \beta \in L_1)\}$$

Theorem: Let $L_1, L_2 \in \mathrm{RegLang}(\Sigma)$. Then $L_1/L_2 \in \mathrm{RegLang}(\Sigma)$ as well.

Proof outline: Let $M = (Q, \Sigma, \delta, q_0, F)$ be a DFA with $\mathcal{L}(M) = L_1$, and let $F' = \{q \in Q \mid (\exists \alpha \in L_2)(\delta^*(q, \alpha) \in F)\}$. Then $M' = (Q, \Sigma, \delta, q_0, F')$ is an accepter for L_1/L_2. $\square$

- Note that the above proof is not constructive.
- How does one determine whether there is an $\alpha \in L_2$ for which $\delta^*(q, \alpha) \in F$?
- It is possible to limit the length of the candidate strings α, but that issue will not be pursued in detail at the moment.

Decision Questions about Regular Languages

- Typical decision questions include the following:
  - Given $L \subseteq \Sigma^*$ and $w \in L$, is $w \in L$?
  - Given $L \subseteq \Sigma^*$, is $\mathcal{L}(L) = \emptyset$?
  - Given $L_1, L_2 \subseteq \Sigma^*$, is $L_1 \cap L_2 = \emptyset$?
  - Given $L_1, L_2 \subseteq \Sigma^*$, is $L_1 = L_2$?
  - Given $L_1, L_2 \subseteq \Sigma^*$, is $L_1 \subseteq L_2$?
- For regular languages, the first three are answerable trivially by representing the language as a DFA (and discarding unreachable states.)
- Thus, they are answerable by running an algorithm.
- For the fourth, it suffices to note that $L_1 = L_2$ iff $L_1 \triangle L_2 = \emptyset$. Thus,

Observation: There is an algorithm to determine whether or not $L_1 = L_2$ for two regular languages L_1 and L_2. $\square$

- For the fifth, it suffices to note that $L_1 \subseteq L_2$ iff $L_1 \setminus L_2 = \emptyset$. Thus,

Observation: There is an algorithm to determine whether or not $L_1 \subseteq L_2$ for two regular languages L_1 and L_2. $\square$

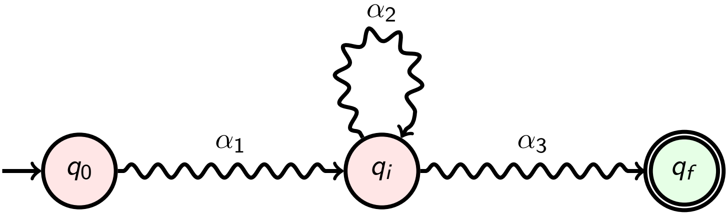
Establishing that a Language is Not Regular

- So far, the focus has been on techniques for establishing that a given language is regular.
- How does one show that a language is not regular?
- The most direct approach is to show that there is no DFA, NFA, RE, or regular grammar which accepts/characterizes/generates it.
- To this end, a result known as the *Pumping Lemma* is the most useful.

The Pumping Lemma for Regular Languages

- Suppose that a DFA $M = (Q, \Sigma, \delta, q_0, F)$ accepts a string $\alpha \in \Sigma^*$ which is longer than the number of states in Q.
- Then the computation must pass through the same state twice.
- In other words, the computation must contain a loop.

$$(q_0, \alpha_1\alpha_2\alpha_3) \vdash_M^* (q_i, \alpha_2\alpha_3) \vdash_M^* (q_i, \alpha_3) \vdash_M^* (q_f, \alpha_3)$$



- $\text{Length}(\alpha_1\alpha_2) < \text{Card}(q) =$ number of states in Q.
- This loop may be repeated any number of times.
- Thus, the machine accepts any string of the form $\alpha_1 \cdot \alpha_2^* \cdot \alpha_3$.

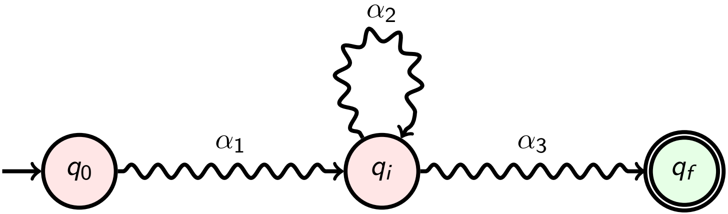
Formal Statement of the Pumping Lemma

Theorem (The Pumping Lemma for regular languages): Let Σ be a finite alphabet, and let $L \in \text{RegLang}(\Sigma)$. Then there is a constant $N \in \mathbb{N}$, depending only upon L, such that for any $\alpha \in L$ with $\text{Length}(\alpha) \geq N$, there is a decomposition

$$\alpha = \alpha_1 \cdot \alpha_2 \cdot \alpha_3$$

with

- $\text{Length}(\alpha_2) \geq 1$;
- $\text{Length}(\alpha_1) + \text{Length}(\alpha_2) \leq N$;
- $\alpha_1 \cdot (\alpha_2)^m \cdot \alpha_3 \in L$ for all $m \in \mathbb{N}$. $\square$



How to Use the Pumping Lemma

- Suppose that $L \subseteq \Sigma^*$ is a language which is to be proven not regular.
- You may assume that N is fixed, but you may not set it to any particular value.
- You may choose the string $\alpha \in L$ to "pump".
- It must be the case that $\text{Length}(\alpha) \geq N$.
- Use N as a parameter of the string α.
- You must take into account *all* decompositions of α into $\alpha_1\alpha_2\alpha_3$ which satisfy the conditions of the pumping lemma.
- In general, the Pumping Lemma can only be used to show that a language is not regular; it cannot be used to show that a language is regular.

An Example of the Use of the Pumping Lemma

Example: Let $L = \{a^k b^k \mid k \in \mathbb{N}\}$ (with the alphabet $\Sigma = \{a, b\}$).

- Show that L is not regular.
- Let N be the constant guaranteed for L by the Pumping Lemma.
- Choose $\alpha = a^N b^N$.
- Every decomposition $\alpha = \alpha_1\alpha_2\alpha_3$ according to the Pumping Lemma must be of the form $\alpha_1 = a^{n_1}$; $\alpha_2 = a^{n_2}$; $\alpha_3 = a^{n_3} b^N$; with $n_1 + n_2 \leq N$, $n_2 > 0$, and $n_1 + n_2 + n_3 = N$.
- Then, if L were regular, it would be the case that $\alpha_1(\alpha_2)^2\alpha_3 = a^{n_1+2n_2+n_3} b^N \in L$, which is clearly not the case.
- Alternately if L were regular, it would be the case that $\alpha_1(\alpha_2)^0\alpha_3 = a^{n_1+n_3} b^N \in L$, which is clearly not the case either.
- In fact, $\alpha_1(\alpha_2)^k\alpha_3 = a^{n_1+kn_2+n_3} b^N \notin L$ for any $k \neq 1$.
- Thus, there are many alternatives to "pump" in this example.

Further Examples of Application of the Pumping Lemma

- The same or very similar strings may be used to prove that related languages are not regular.

Example: $L = \{w \in \{a,b\}^* \mid \text{Count}\langle a, w\rangle = \text{Count}\langle b, w\rangle\}$.

- Let N be the constant guaranteed for L by the Pumping Lemma for this language.
- The same string $a^N b^N \in L$ may be used to show that this language is not regular, in exactly the same way.

Example: $L = \{a^{k_1} b^{k_2} \mid k_1, k_2 \in \mathbb{N} \text{ and } k_1 < k_2\}$.

- Notation as in the Pumping Lemma, choose $\alpha = a^N b^{N+1}$ and proceed as in the previous examples.
- Here one must pump *up* to show that $\alpha_1(\alpha_2)^2\alpha_3 \notin L$.

Example: $L = \{a^{k_1} b^{k_2} \mid k_1, k_2 \in \mathbb{N} \text{ and } k_1 > k_2\}$.

- Choose $\alpha = a^{N+1} b^N$, decompose in a manner similar to the previous examples, and pump *down*, showing $\alpha_1 {\alpha_2}^0 \alpha_3 = \alpha_1\alpha_3 \notin L$.

Further Examples of Application of the Pumping Lemma

Example: $L = \{w \in \{a, b\}^* \mid w = w^R\}$ (*palindromes*).

- Notation continues as in the statement of the Pumping Lemma.
- Choose $\alpha = a^N b a^N$.
- Pump up or down to show that the language is not regular.

Example: $L = \{ww^R \mid w \in \{a, b\}^*\}$.

- Choose $\alpha = a^N bba^N$ and proceed as above.

Example: $L = \{w\beta w^R \mid w, \beta \in \{a, b\}^*\}$.

- Careful!! This language is regular and equal to $\{a, b\}^*$.
- To obtain any $\beta \in \{a, b\}^*$, just choose $w = w^R = \lambda$.

Example: $L = \{w\beta w^R \mid w, \beta \in \{a, b\}^* \text{ and Length(w)} > 0\}$.

- This language is also regular, with
  $L = \{a_1 \ldots a_k \in \{a, b\}^* \mid k > 2 \text{ and } a_1 = a_k\}$
  $= \mathcal{L}((a \cdot (a + b)^* \cdot a) + (b \cdot (a + b)^* \cdot b))$.

A More Difficult Example

Example: Let $L = \{w = a^{k_1} b^{k_2} \mid k_1 \neq k_2\}$.

- Need to choose a string of the form $\alpha = a^{N_1} b^{N_2} \in L$ which can be pumped to $a^{N_2} b^{N_2}$.
- This is possible with $N_1 = N!$ and $N_2 = (N+1)!$.
- See the text for the argument.
- There is a better way!
- Note that $L' = \{a^{k_1} b^{k_2} \mid k_1 = k_2\} = \overline{L} \cap \mathcal{L}(a^* b^*)$.
- Since regular languages are closed under complement and intersection, if L were regular, so too would be L'.
- Hence, L cannot be regular.
- The Pumping Lemma is not always the best to use to show that a given language is not regular.

Are Programming Languages Regular?

- Most programming languages allow nested expressions, marked by parentheses or the like.

Example: `(X + (Y * Z) / (W + (A + 3))) - 2`

- To check that an expression is well formed, it is therefore necessary to verify that the parentheses are balanced.
- Let L_{paren} denote the language over $\{(,)\}$ which consists of all strings with balanced parentheses.

Examples: $(()()(()))\in L_{\text{paren}}$
$((()()(()))\notin L_{\text{paren}}$

Observation: L_{paren} is not regular.

Proof outline:

- For convenience, replace (by a and) by b.
- Choose $\alpha = a^N b^N \in L_{\text{paren}}$ and pump up or down. $\square$