Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Networked Systems Group (NSG)

HS 2024

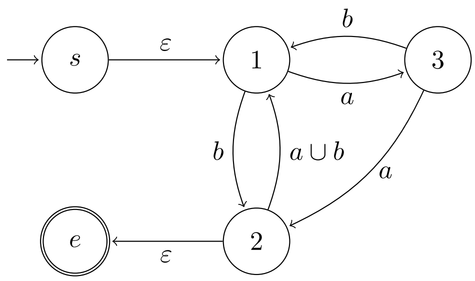
Prof. L. Vanbever / R. Schmid
based on Prof. R. Wattenhofer's material

Discrete Event Systems

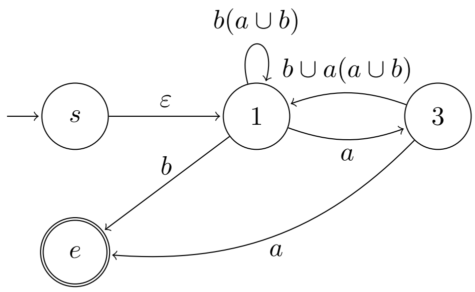
Exercise Sheet 3

1 From DFA to Regular Expression

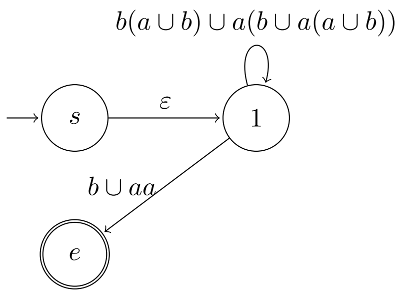
First generate the GNFA:



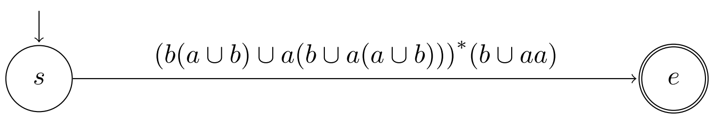
Then begin by ripping out any node. We start by removing node 2:



Then, we remove node 3:



Finally, we remove node 1 and derive the corresponding regular expression:



Note that we could have ripped out states in any particular order. However, some orders lead to smaller results than others:

1-2-3: $\big(\underbrace{b((a\cup b)b)^*}_{s\to t}\big)\cup\Big(\big(\underbrace{a\cup(b((a\cup b)b)^*(a\cup b)a)}_{s\to(3)}\big)\big(\underbrace{ba\cup(a((a\cup b)b)^*(a\cup b)a)}_{\text{loop at (3)}}\big)^*\big(\underbrace{a((a\cup b)b)^*}_{(3)\to t}\big)\Big)$

1-3-2: $\big(\underbrace{\emptyset}_{s\to t}\big)\cup\big(\underbrace{b\cup(a(ba)^*(a\cup bb))}_{s\to(2)}\big)\big(\underbrace{(a\cup b)b\cup(a\cup b)a(ba)^*(a\cup bb)}_{\text{loop at (2)}}\big)^*\big(\underbrace{\varepsilon}_{(2)\to t}\big)$

2-1-3: $\big(\underbrace{(b(a\cup b))^*b}_{s\to t}\big)\cup\big(\underbrace{(b(a\cup b))^*a}_{s\to(3)}\big)\big(\underbrace{(b\cup a(a\cup b))(b(a\cup b))^*a}_{\text{loop at (3)}}\big)^*\big(\underbrace{(a\cup(b\cup a(a\cup b)))(b(a\cup b))^*b}_{(3)\to t}\big)$

2-3-1: $\big(\underbrace{\emptyset}_{s\to t}\big)\cup\big(\underbrace{\varepsilon}_{s\to(1)}\big)\big(\underbrace{b(a\cup b)\cup a(b\cup a(a\cup b))}_{\text{loop at (1)}}\big)^*\big(\underbrace{b\cup aa}_{(1)\to t}\big)$

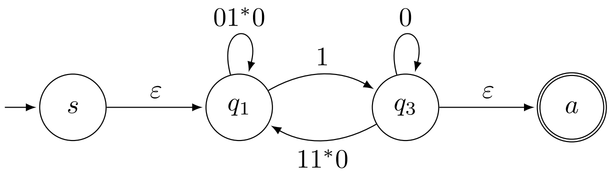
3-1-2: $\big(\underbrace{\emptyset}_{s\to t}\big)\cup\big(\underbrace{(ab)^*(b\cup aa)}_{s\to(2)}\big)\big(\underbrace{(a\cup b)(ab)^*(b\cup aa)}_{\text{loop at (2)}}\big)^*\big(\underbrace{\varepsilon}_{(2)\to t}\big)$

3-2-1: $\big(\underbrace{\emptyset}_{s\to t}\big)\cup\big(\underbrace{\varepsilon}_{s\to(1)}\big)\big(\underbrace{ab\cup(b\cup aa)(a\cup b)}_{\text{loop at (1)}}\big)^*\big(\underbrace{b\cup aa}_{(1)\to t}\big)$

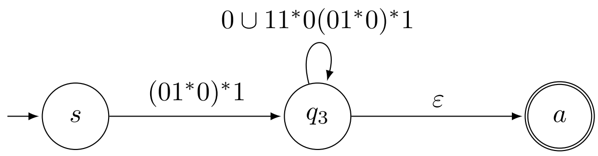
Hint: The annotations indicate where each of these subformulas can be found in the last step. Generally, it is a good idea to start ripping out states based on their in-degree multiplied with their out-degree, as this is the amount of edges they will affect. One can count loops as both in- and outgoing edges because they complicate the resulting formulas as well.

2 Transforming Automata [Exam HS14]

The regular expression can be obtained from the finite automaton using the transformation presented in the script. After ripping out state q_2, the corresponding GNFA looks like this:



After also removing state q_1, the GNFA looks as follows.



Eliminating the last state q_3 yields the final solution, which is $(01^*0)^*1(0\cup 11^*0(01^*0)^*1)^*$.

Note: Ripping out the interior states in a different order yields a distinct yet equivalent regular expression. The order q_3, q_2, q_1, for example, results in $((0\cup 10^*1)1^*0)^*10^*$.

3 Pumping Lemma

The Pumping Lemma in a Nutshell

Given a language L, assume for contradiction that L is regular and has the pumping length p. Construct a suitable word $w \in L$ with $|w| \geq p$ ("there *exists* $w \in L$") and show that for *all* divisions of w into three parts, $w = xyz$, with $|x| \geq 0$, $|y| \geq 1$, and $|xy| \leq p$, there *exists* a pumping exponent $i \geq 0$ such that $w' = xy^iz \notin L$. If this is the case, L is not regular.

a) We claim that L_1 is not regular and prove our claim with the pumping lemma recipe:

1. Assume for contradiction that L_1 was regular.
2. There must exist some p, s.t. any word $w \in L_1$ with $|w| \geq p$ is pumpable.
3. Choose the string $w = 1^p02^p \in L_1$ with length $|w| > p$.
4. Consider all ways to split $w = xyz$ s.t. $|xy| \leq p$ and $|y| \geq 1$.
   $\rightarrow$ Hence, $y \in 1^+$.
5. Observe that $xy^0z \notin L_1$ – a contradiction to p being a valid pumping length.
6. Consequently, L_1 cannot be regular.

b) Language L_2 can be shown to be non-regular using the pumping lemma. We will showcase how this might look without using the recipe presented above:
Assume for contradiction that L_2 is regular and let p be the corresponding pumping length. Choose w to be the word 0110^p1^p. Because w is an element of L_2 and has length more than p, the pumping lemma guarantees that w can be split into three parts, $w = xyz$, where $|xy| \leq p$ and for any $i \geq 0$, we have $xy^iz \in L_2$. In order to obtain the contradiction, we must prove that for every possible partition into three parts $w = xyz$ where $|xy| \leq p$, the word w cannot be pumped. We therefore consider the various cases.

(1) If y starts anywhere within the first three symbols (i.e. 011) of w, deleting y (pumping with $i = 0$) creates a word with an illegal prefix (e.g. $1\,0^p\,1^p$ for $y = 01$).
(2) If y consists of only 0s from the second block, the word $w' = xy^2z$ has more 0s than 1s in the last $|w'| - 3$ symbols and hence $c \neq d$.

Note that y cannot contain 1s from the second block because of the requirement $|xy| \leq p$.

We have shown that for all possible divisions of w into three parts, the pumped word is not in L_2. Therefore, L_2 cannot be regular and we have a contradiction.

Be Careful!

One may think one could show the same based on the closure properties of regular languages. However, this only works in **one direction**! That is, for an operator $\diamond \in \{\cup, \cap, \bullet\}$, we have:

If L_1 and L_2 are regular, then $L = L_1 \diamond L_2$ is also regular.

If either L_1 or L_2 or both are non-regular, we cannot deduce the non-regularity of L or vice-versa. Moreover, L being regular does not imply that L_1 and L_2 are regular as well. This may sound counter-intuitive which is why we give examples for the three operators.

- $L = L_1 \cup L_2$: Let L_1 be any non-regular language and L_2 its complement. Then $L = \Sigma^*$ is regular.
- $L = L_1 \cap L_2$: Let L_1 be any non-regular language and L_2 its complement. Then $L = \emptyset$ is regular.
- $L = L_1 \bullet L_2$: Let $L_1 = \{a^*\}$ (a regular language) and $L_2 = \{a^p \mid p \text{ is prime}\}$ (a non-regular language) then $L = \{aaa^*\}$ is regular.

Hence, to prove that a language L_x is non-regular, you assume it to be regular for contradiction. Then you combine it with a *regular* language L_r to obtain a language $L = L_x \diamond L_r$. If L is non-regular, L_x could not have been regular either.

4 Pumping Lemma Revisited

a) Let us assume that L is regular and show that this results in a contradiction.

We have seen that any regular language fulfills the pumping lemma. This means, there exists a number p, such that every word $w \in L$ with $|w| \geq p$ can be written as $w = xyz$ with $|xy| \leq p$ and $|y| \geq 1$, such that $xy^iz \in L$ for all $i \geq 0$.

In order to obtain the contradiction, we need to find at least one word $w \in L$ with $|w| \geq p$ that does not adhere to the above proposition. We choose $w = xyz = 1^{p^2}$ and consider the case $i = 2$ for which the Pumping Lemma claims $w' = xy^2z \in L$.

We can relate the lengths of $w = xyz$ and $w' = xy^2z$ as follows.

$$p^2 = |w| = |xyz| < |w'| = |xy^2z| \leq p^2 + p < p^2 + 2p + 1 = (p+1)^2$$

So we have $p^2 < |w'| < (p+1)^2$ which implies that $|w'|$ cannot be a square number since it lies between two consecutive square numbers. Hence, $w' \notin L$ and L cannot be regular.

b) Consider the alphabet $\Sigma = \{a_1, a_2, ..., a_n\}$ and the language $L = \cup_{i=1}^{n} a_i^* = a_1^* \cup a_2^* \cup \cdots \cup a_n^*$. In other words, each word of the language L contains an arbitrary number of just **one** symbol a_i. The language is regular, as it is the union of regular languages, and the smallest possible pumping number p for L is 1. But any DFA needs at least $n+2$ states to accept the empty word, distinguish the n different characters of the alphabet, and for a failing state. Thus, for a DFA, we cannot deduce any information from p about the minimum number of states. The same argument holds for an NFA.

5 Minimum Pumping Length

To begin with, observe that the minimum pumping length p of a language $L = L_1 \cup L_2$ is at most $p \leq max\{p_1, p_2\}$, where p_1 and p_2 are the minimum pumping lengths of L_1 and L_2, respectively. This holds because if there is already a string w that is pumpable in L_1, then w will also be pumpable in L. Hence, let $L_1 = 1^*0^+1^+0^*$ and $L_2 = 111^+0^+$.

- The minimum pumping length of L_2 cannot be 4 because 1110 cannot be pumped. Now consider the string s that belongs to L_2 and that has a size of 5. If $s = 11110$, then it can be divided into xyz where $x = 111$, $y = 1$ and $z = 0$ and thus can be pumped. If $s = 11100$, then it can be divided into xyz where $x = 111$, $y = 0$ and $z = 0$ and thus can be pumped. Similarly, all longer words can be pumped. The minimum pumping length for $L2$ is thus 5.
- A string s of size 3 and belonging to L1 can always be pumped.

Considering the word 1110, observe that it can also not be pumped in $L = L_1 \cup L_2$. In conclusion, the minimum pumping length of L is 5.

6 The art of being regular

We use the pumping lemma to show that L is not regular. To begin with, consider the equivalent language $L = \{a\#b \mid a = 2b\}$ and assume (for a contradiction) that L is regular. Hence, the pumping lemma holds and there is some valid pumping number p. We choose the string $w = 100^p\#10^p$ where $a = 100^p$ is equal to $2b$ ($b = 10^p$) for $p \geq 0$. Since $|w| > p$, we know that w must be pumpable for some split $w = xyz$. Following $|xy| \leq p$, we must consider two cases:

a) $x = \varepsilon$, $y \in 10^*$: Arithmetic is wrong for xy^0z. Left side is 0 but the right side isn't.

b) $x \in 10^*$, $y \in 0^+$: Arithmetic is wrong for xy^0z. Decreased left side but not right. In particular, it is no longer the case that $a > b$ (required since $b \neq 0$).

Hence, we conclude that the pumping lemma does not hold for the language L, which can thus not be regular.

Bonus tasks: – *solutions provided by student Angéline Pouget in HS20*

- *Determine whether* $L = \{x\#y \mid x + y = 3y\}$ *is context-free.*

  To begin with, we observe that

  $$\begin{aligned} L &= \{x\#y \mid x + y = 3y\} \\ &= \{x\#y \mid x = 2y\} \\ &= \{w0\#w \mid w \in 1(0 \cup 1)^*\}. \end{aligned}$$

  We prove that $L = \{w0\#w \mid w \in 1(0 \cup 1)^*\}$ is not context-free using the tandem-pumping lemma. First, we assume for contradiction that L is context-free and hence there is a number p such that any string in L of length $\geq p$ is tandem-pumpable within a substring of length p. We choose $w = 1^p0^p$ and thereby consider the word $\alpha = w0\#w = 1^p0^p0\#1^p0^p$ with $|\alpha| \geq p$.

  We now want to split $\alpha = uvxyz$ with $|vy| \geq 1$, $|vxy| \leq p$ and $uv^ixy^iz \in L$ for all $i \geq 0$. Because we have $|vxy| \leq p$, there are the following options:

  - $\# \notin vxy$ ($vxy = 1^m$ or $vxy = 0^m$ with $1 \leq m \leq p$ or $vxy = 1^n0^s$ with $n + s \leq p$). Any one of these sequences can either be before or after the # but independent of this choice, if we pump v and y and choose for example $i = 0$, we will have $\alpha' = w'0\#w''$ with $w' \neq w$ and hence $\alpha' \notin L$.
  - $\# \in vxy$. In this case, we can choose $x = \#$ because we know that there is only one # and therefore this cannot be the pumpable part. This leaves us with $v = 0^n$ and $y = 1^s$ with $1 \leq n + s \leq p - 1$ and if we for example set $i = 0$ this leaves us with $\alpha' = 1^p0^{p+1-n}\#1^{p-s}0^p$ which is $\notin L$.

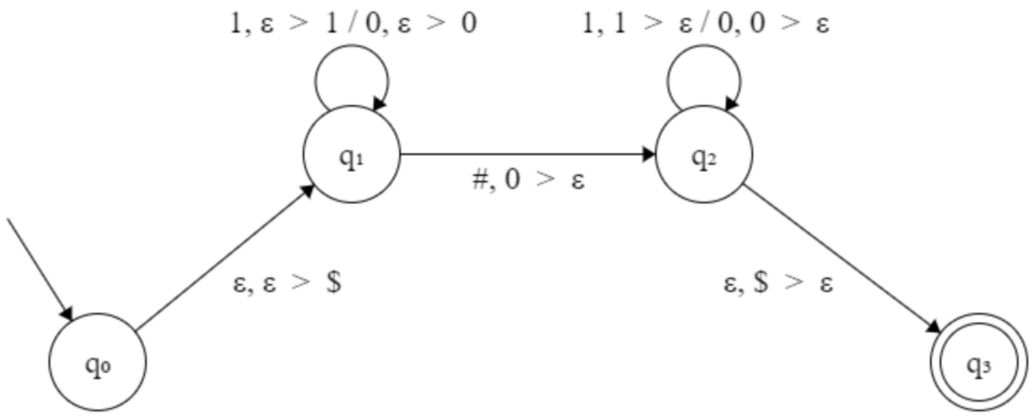
  Because we have now considered all possible splits of this word into $\alpha = uvxyz$, we can safely say that language L is not context-free.

- *Show whether* $L' = \{x\#y \mid x + reverse(y) = 3 \cdot reverse(y)\}$ *is context-free.*
  The reverse()-function takes an integer as a bitstring and reverses the order of its bits.

  Let $w' = reverse(w)$. Applying the same transformations as above, we obtain

  $$L' = \{x\#y \mid x = 2 \cdot reverse(y)\} = \{w0\#w' \mid w \in 1(0 \cup 1)^*\}.$$

  We can show that this language is context-free by drawing a push-down automaton that accepts this language. This automaton is depicted below with ">" representing stack operations "→".



  We could have alternatively shown that the language is context-free by providing a context free grammar (V, Σ, R, S) such as the following:

  - $V = \{S\}$

- $\Sigma = \{0, 1, \#\}$
- $R:\ S \to 1S1 \mid 0S0 \mid 0\#$
- $S = S$