Discrete Event Systems

Exercise Session 3

Roland Schmid

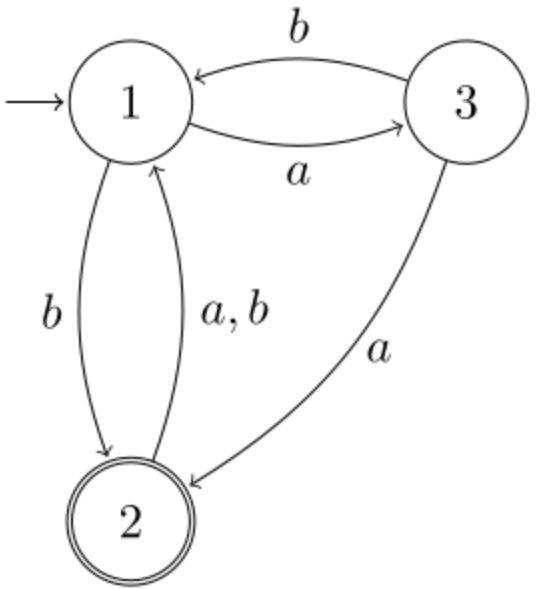
nsg.ee.ethz.ch

ETH Zürich (D-ITET)

3 October 2024

1 From DFA to Regular Expression

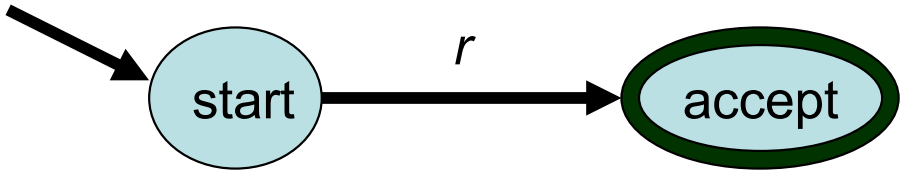
Build a language-equivalent GNFA for the DFA below. Then, as seen during the lecture, eliminate the intermediate states of the GNFA and derive the regular expression that represents the language accepted by the DFA.



NFA → REX conversion process

1. Construct a GNFA from the NFA.
   A. If there are more than one arrows from one state to another, unify them using "∪"
   B. Create a unique start state with in-degree 0
   C. Create a unique accept state of out-degree 0
   D. [If there is no arrow from one state to another, insert one with label Ø]

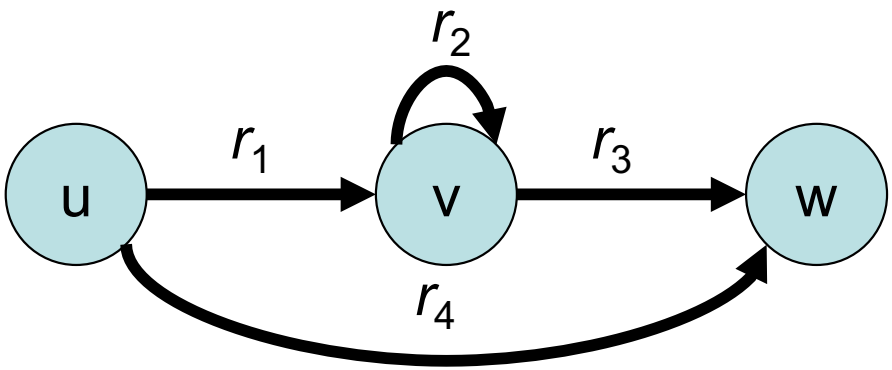
2. Loop: As long as the GNFA has strictly more than 2 states:
   Rip out arbitrary interior state and modify edge labels.



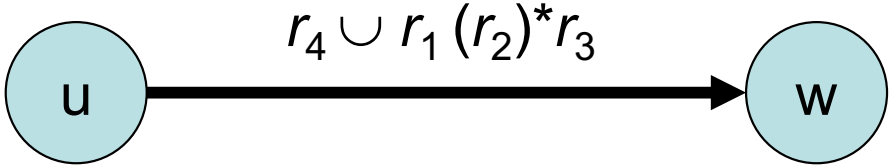
3. The answer is the unique label *r*.

NFA → REX: Ripping Out.

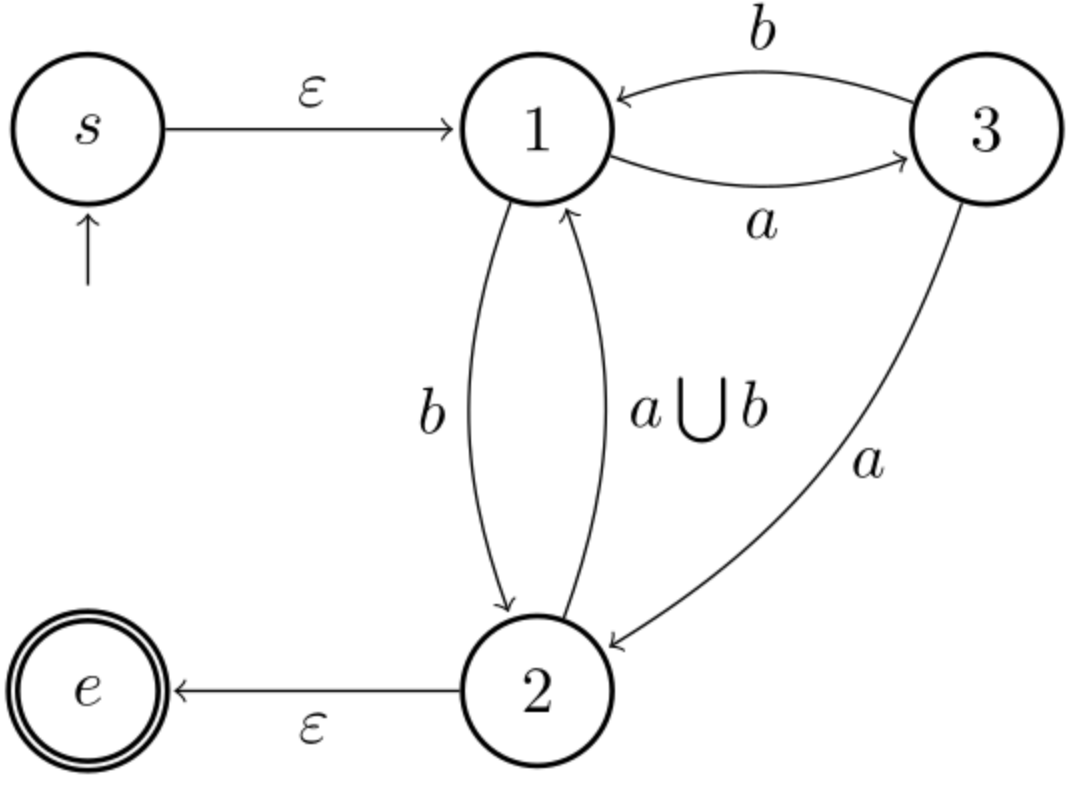
- Ripping out is done as follows. If you want to rip the middle state *v* out (for all pairs of neighbors u,w)…



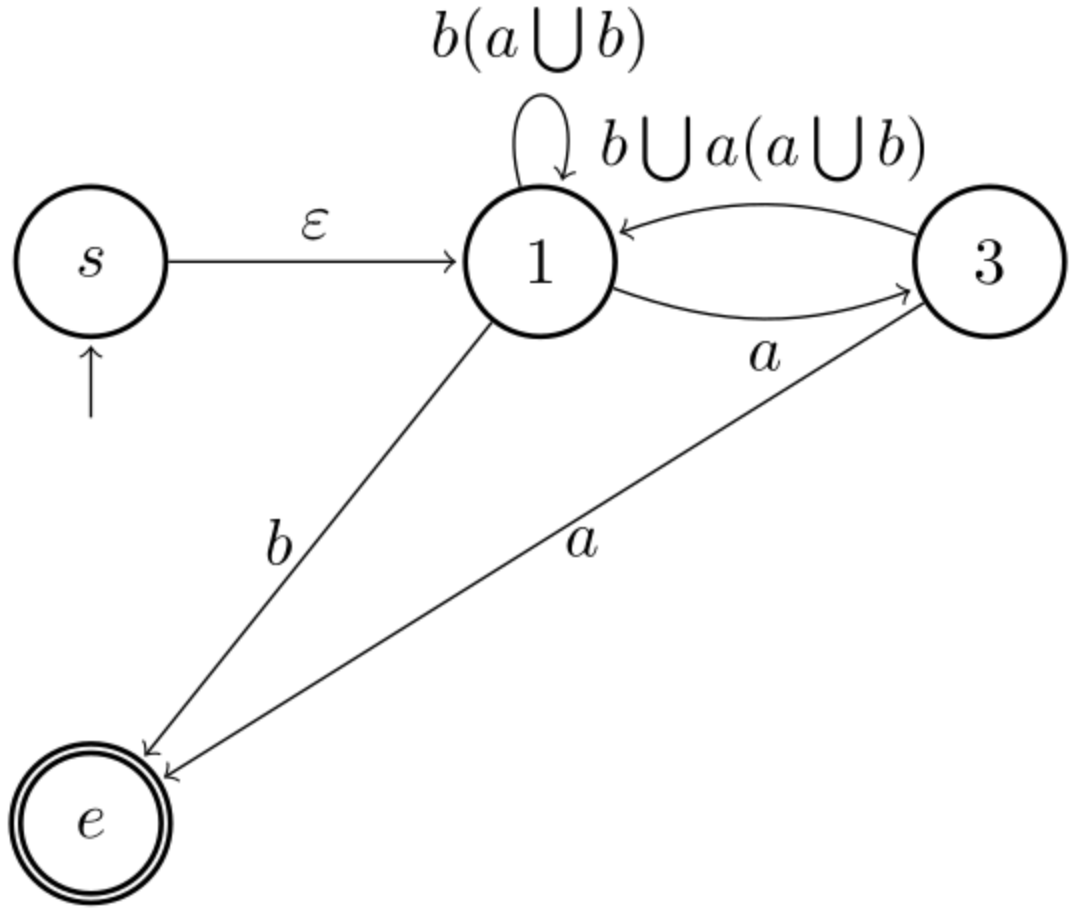
- … then you'll need to recreate all the lost possibilities from *u* to *w*. I.e., to the current REX label r_4 of the edge (*u*,*w*) you should add the concatenation of the (*u*,*v*) label r_1 followed by the (*v*,*v*)-loop label r_2 repeated arbitrarily, followed by the (*v*,*w*) label r_3. The new (*u*,*w*) substitute would therefore be:



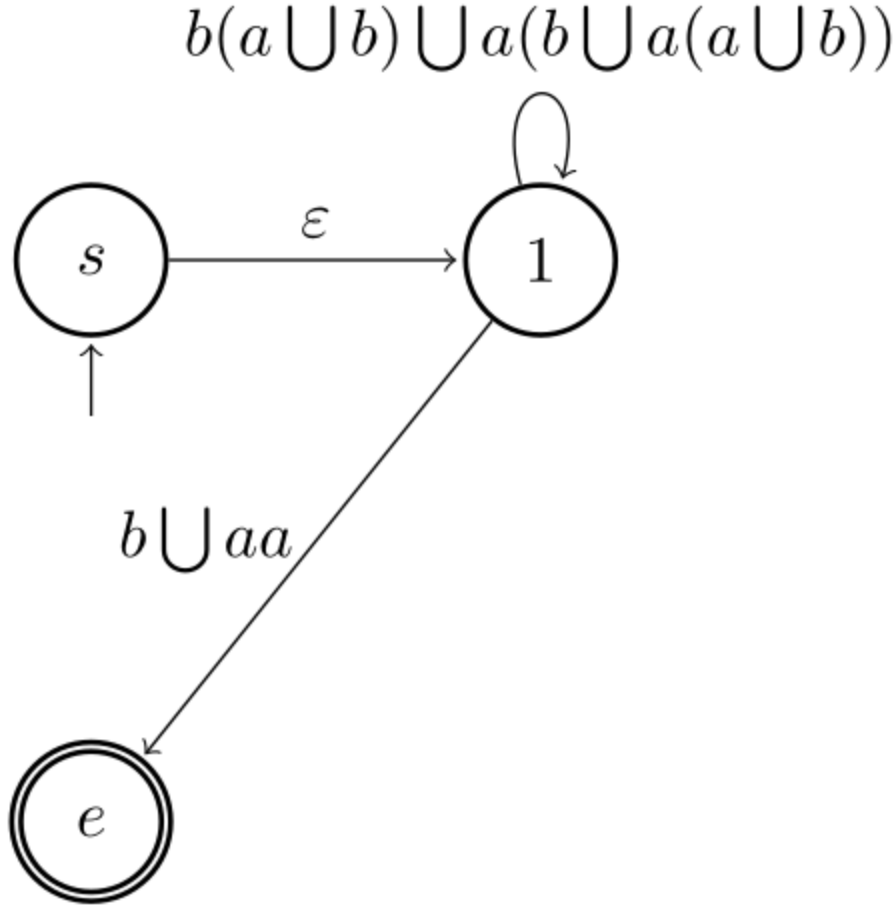
First generate the GNFA.



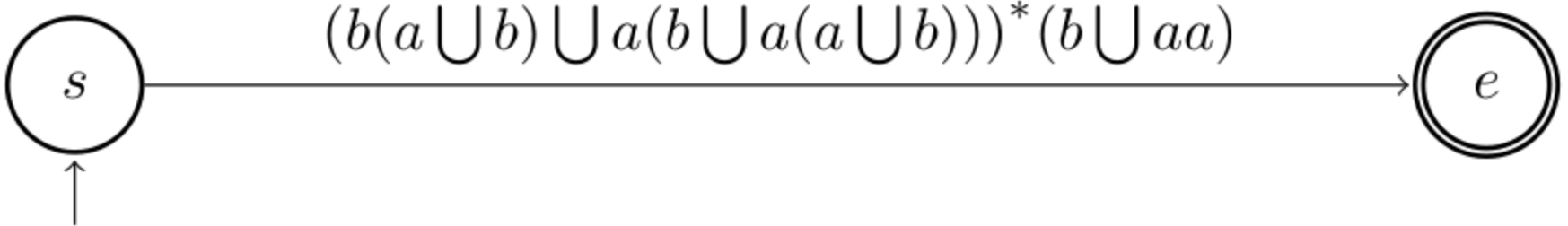
Then remove node 2.



Then remove node 3.



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

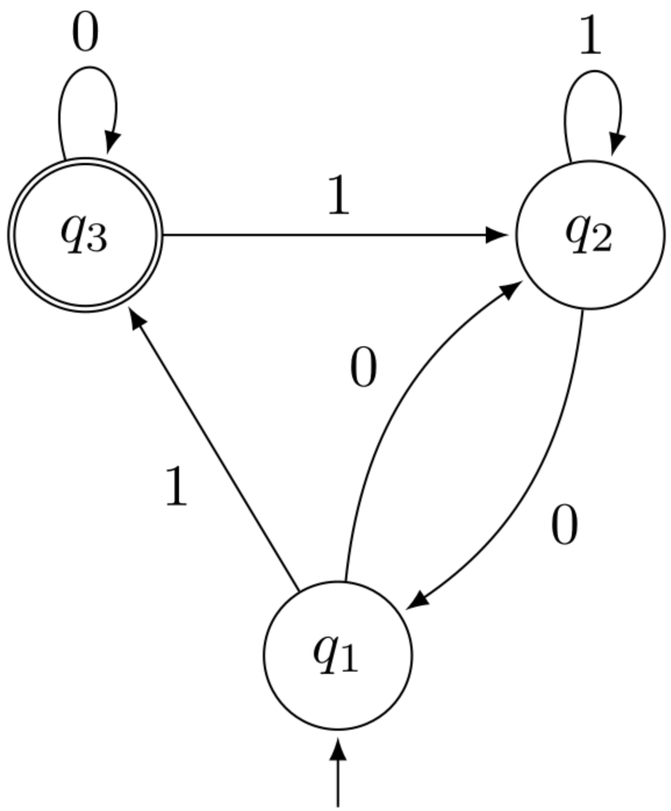
3-1-2: $\big(\underbrace{\emptyset}_{s\to t}\big)\cup\big(\underbrace{(ab)^*(b\cup aa)}_{s\to(2)}\big)\big(\underbrace{(a\cup b)(ab)^*(b\cup aa)}_{\text{loop at (2)}}\big)^*\big(\underbrace{\varepsilon}_{(2)\to t}\big)$

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]

Consider the following DFA over the alphabet $\Sigma = \{0, 1\}$. Give a regular expression for the language L accepted by the following automaton. If you like, you can do this by ripping out states as presented in the lecture.



3 Pumping Lemma

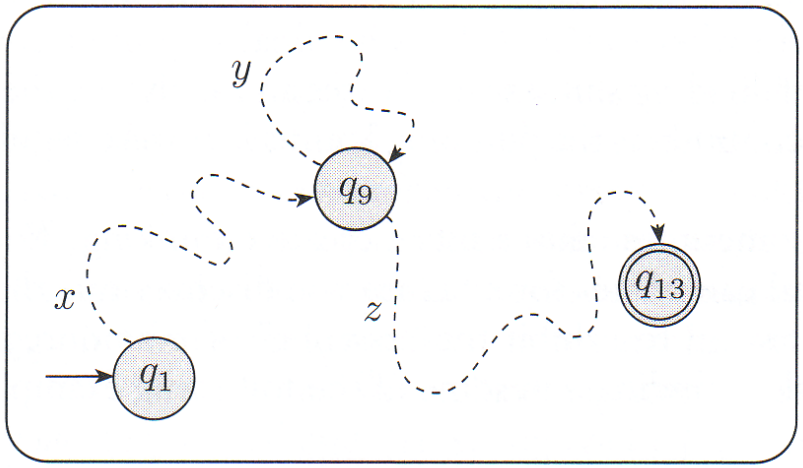
Which of the following languages is regular? Prove your claims!

a) $L_1 = \{1^n 0 2^n \mid n \geq 0\}$

b) $L_2 = \{0^a 1^b 0^c 1^d \mid a, b, c, d \geq 0 \text{ and } a = 1, b = 2 \text{ and } c = d\}$

Languages with unbounded strings

- Consequently, regular languages with unbounded strings can only be recognized by FA (finite! bounded!) automata if these long strings loop.



- The FA can enter the loop once, twice, …, and not at all.
- That is, language L contains all {xz, xyz, xy^2z, xy^3z, …}.

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 $w = 1^p 0 2^p \in L_1$ 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.

4 Pumping Lemma Revisited

a) Determine whether the language $L = \{1^{n^2} \mid n \in \mathbb{N}\}$ is regular. Prove your claim!

b) Consider a regular language L and a pumping number p such that every word $u \in L$ can be written as $u = xyz$ with $|xy| \leq p$ and $|y| \geq 1$ such that $xy^iz \in L$ for all $i \geq 0$.

Can you use the pumping number p to determine the number of states of a minimal DFA accepting L? What about the number of states of the corresponding NFA?

5 Minimum Pumping Length

Consider the regular language $L = 1^*0^+1^+0^* \cup 111^+0^+$. Give the minimum pumping length and briefly explain the intuition behind your answer.

Feedback