Théorie des Langages Rationnels

Properties of Rational Languages

Adrien Pommellet, LRE

April 14, 2023

Brzozowski–McCluskey's algorithm

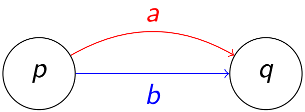
Inverting the pipeline

$$
\begin{tikzcd}
\text{Regular expressions} \arrow[d, "\text{Thompson's algorithm}"] \\
\varepsilon\text{-NFA} \arrow[d, "\text{Backward removal of } \varepsilon\text{-transitions}"] \\
\text{NFA} \arrow[d, "\text{Pruning procedure}"] \\
\text{pruned NFA} \arrow[d, "\text{Determinization procedure}"] \\
\text{DFA} \arrow[d, "\text{Moore's algorithm}"] \\
\text{Minimized DFA} \arrow[uuuuu, dotted, bend left=60, "\text{BMC's algorithm}"]
\end{tikzcd}
$$

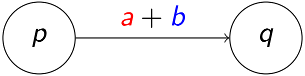
Brzozowski–McCluskey's algorithm

An intuition on edges

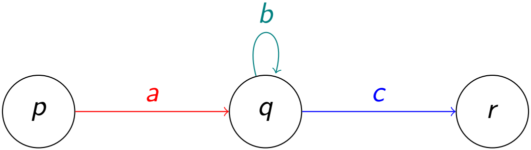
Consider the following pattern:



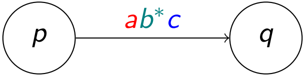
Were we to allow edges labelled by **regular expressions**, then this automaton would be equivalent to:



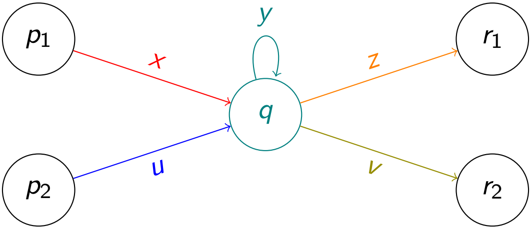
In a similar manner, if we consider the pattern:



Then this automaton is equivalent to:

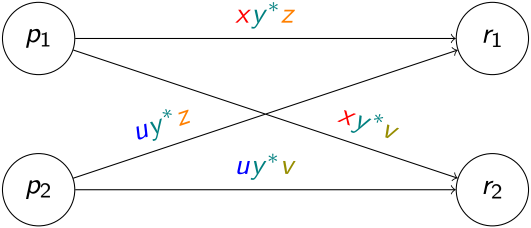


However, if we consider the pattern:



We take into account **every possible path**: xy^*z, xy^*v, uy^*z, uy^*v.

We therefore must add four edges, **one for each possible path**:

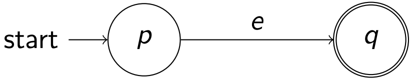


We create $|\textit{incoming edges}| \times |\textit{outgoing edges}|$ edges as we remove q.

Brzozowski–McCluskey's algorithm

The terminal case

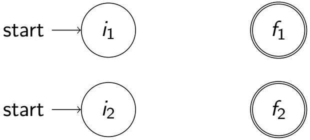
We will **remove almost every state and edge** of $\mathcal{A}$ until only the pattern below is left; then the regular expression e is such that $\mathcal{L}(\mathcal{A}) = \mathcal{L}(e)$.



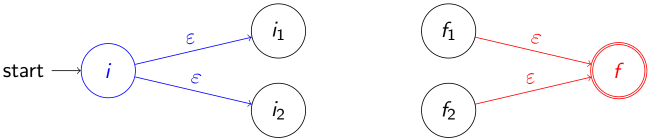
However, in order to reach this point, there must be **only one** initial state and one distinct accepting state.

Moreover, the initial state must have no **incoming** transitions, and the accepting state, no **outgoing** transitions.

If there are **multiple** initial and accepting states with the wrong edges:



Then we design a new initial state i, a new accepting state f, then use ε-transitions to link them to the **former** initial and accepting states.



Brzozowski–McCluskey's algorithm

A summary

1. Ensure that there is a single initial state with no incoming edges and a single, distinct accepting state with no outgoing edges.
2. Apply the **edge removal pattern** whenever possible.
3. Apply the **state removal pattern** whenever possible.
4. Stop once there is a single edge between the initial state and the accepting state.
5. This edge is then labelled by a **regular expression** e equivalent to the input automaton.

Brzozowski–McCluskey's algorithm

A consequence

Theorem

Given an automaton $\mathcal{A}$ on the alphabet Σ, there exists a regular expression $e \in \mathrm{Reg}_\Sigma$ such that $\mathcal{L}(\mathcal{A}) = \mathcal{L}(e)$.

Note that this theorem applies to DFA, NFA, and ε-NFA alike.

Moreover, depending on the order in which the removal patterns are applied, the algorithm may yield **different but equivalent** regular expressions.

Practical Application

Exercise 1. Find a regular expression equivalent to the automaton $\mathcal{A}$.

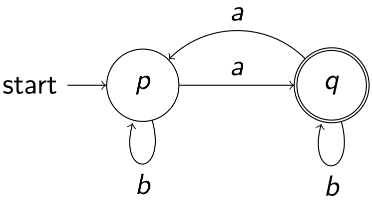


Figure 1: Automaton $\mathcal{A}$.

Answer I

Answer II

Answer III

Answer IV

Answer V

By design, rational languages are stable by union, Kleene star and concatenation. What about **other operations**?

Practical Application

Exercise 2. Find an automaton $\mathcal{A}_2$ on the alphabet $\Sigma = \{a, b\}$ such that $\mathcal{L}(\mathcal{A}_2) = \overline{\mathcal{L}(\mathcal{A}_1)}$.

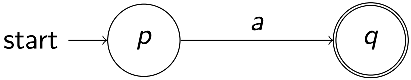


Figure 2: Automaton $\mathcal{A}_1$.

Answer

Theorem

If $L \in \mathrm{Rat}_\Sigma$, then $\overline{L} \in \mathrm{Rat}_\Sigma$ as well.

Proof. Let $\mathcal{A}_1$ be a **complete DFA** recognizing L. Consider the complete DFA $\mathcal{A}_2$ obtained by switching $\mathcal{A}_1$'s accepting and non-accepting states. Then $\mathcal{A}_2$ recognizes $\overline{L}$.

Note that $\mathcal{A}_1$ **must** be complete and deterministic to ensure each word admits exactly one path, that we can therefore alter.

Lemma

Given a rational language $L \in Rat_\Sigma$, $\mathrm{Pref}(L)$, $\mathrm{Suff}(L)$, $\mathrm{Fact}(L)$, *and the mirror language* L^R *are all rational.*

The proof is left as an exercise. Like the previous property, it relies on our ability to **compute a DFA that recognizes** L, then modifying it so that it recognizes instead another language derived from L.

A consequence of the previous theorem is the following:

Theorem

If $L_1, L_2 \in \mathrm{Rat}_\Sigma$, then $L_1 \cap L_2 \in \mathrm{Rat}_\Sigma$ as well.

Proof. $L_1 \cap L_2 = \overline{\overline{L_1} \cup \overline{L_2}}$, and both the union and the complementation preserve rationality.

But can we build a DFA explicitly recognizing $L_1 \cap L_2$?

Consider two DFA $\mathcal{A}_1$ and $\mathcal{A}_2$. Our intuition is to perform a synchronized product $\mathcal{A}_1 \times \mathcal{A}_2$ of their transitions:

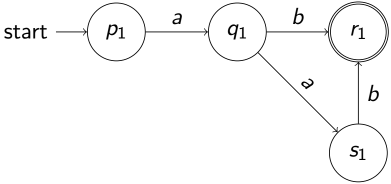
$$p_1 \xrightarrow{a} q_1 \qquad\qquad p_2 \xrightarrow{a} q_2$$

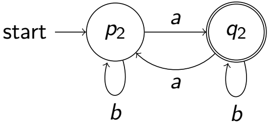
$$p_1, p_2 \xrightarrow{a} q_1, q_2$$

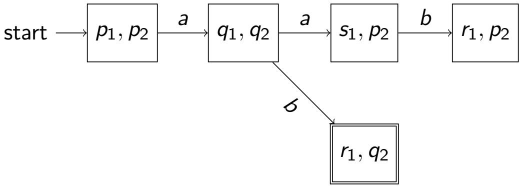
Intuitively, we simulate **both** DFA at the same time.

Decidability Properties of Rational Languages

Computing the synchronized product







These properties apply to the synchronized product $\mathcal{A}_1 \times \mathcal{A}_2$:

- The **initial state** of $\mathcal{A}_1 \times \mathcal{A}_2$ is $(i_1, i2)$ where i_1 (resp. i_2) is $\mathcal{A}_1$'s (resp. $\mathcal{A}_2$'s) initial state.
- A state (f_1, f_2) of $\mathcal{A}_1 \times \mathcal{A}_2$ is **accepting** if and only if f_1 (resp. f_2) is an accepting state of $\mathcal{A}_1$ (resp. $\mathcal{A}_2$).
- If $\mathcal{A}_1$ has n_1 states and $\mathcal{A}_2$ has n_2 states, then $\mathcal{A}_1 \times \mathcal{A}_2$ has **at most** $n_1 \times n_2$ states.

Lemma

Given $L \in Rat_\Sigma$, we can decide whether $L = \emptyset$ or not.

Proof. Let $\mathcal{A}$ be a DFA recognizing L. Using a **depth-first search** starting from $\mathcal{A}$'s initial state, we can determine whether an accepting state is reachable or not.

If there is **at least one** path from the initial state to an accepting state, then $\mathcal{A}$ accepts its label. If there is none, $L = \mathcal{L}(\mathcal{A}) = \emptyset$.

Lemma

Given $L_1, L_2 \in Rat_\Sigma$, we can decide whether $L_1 = L_2$ or not.

Proof. Note that $(L_1 = L_2) \iff (L_1 \subseteq L_2) \land (L_2 \subseteq L_1)$. But $(L_1 \subseteq L_2) \iff L_1 \cap \overline{L_2} = \emptyset$. $L_1 \cap \overline{L_2}$ being rational, we can decide whether it is empty or not.

In a similar fashion, $(L_2 \subseteq L_1) \iff L_2 \cap \overline{L_1} = \emptyset$. $L_2 \cap \overline{L_1}$ is also rational, thus we can also check its emptiness.

A practical algorithm based on this proof would be the following:

1. Compute two **complete DFA** $\mathcal{A}_1$ and $\mathcal{A}_2$ respectively recognizing L_1 and L_2.
2. Compute their **complements** $\overline{\mathcal{A}_1}$ and $\overline{\mathcal{A}_2}$.
3. Compute the **synchronized products** $\overline{\mathcal{A}_1} \times \mathcal{A}_2$ and $\mathcal{A}_1 \times \overline{\mathcal{A}_2}$.
4. Check the **emptiness** of both products by performing a depth first search; the languages are equal if both are empty.
5. The search will otherwise return a **counter-example** w such that $w \in L_1$ but $w \notin L_2$, or $w \in L_2$ but $w \notin L_1$.

That's all folks!