




Prof. Ryan Cotterell

Assignment #1

06/03/2024 - 11:50h



(Sub)questions marked with either  or  indicate a theory or a coding question, respectively.

Question 1: Weighted Languages (7 points; 2 + 2 + 3)

-  In $\mathbb{R}_{+, \times} \langle \langle \{a, b\}^* \rangle \rangle$, simplify the expression $3ab + 4ab + (5ba \times 7)$.
-  In $\mathbb{R}_{\min, +} \langle \langle \{a, b\}^* \rangle \rangle$, simplify the expression $3ab + 4ab + (5ba \times 7)$.
-  In general, we write ST as an abbreviation for $S \times T$. There is a danger of confusion here, since $3ab$ is ambiguous: it could mean either (i) the function that maps $ab \mapsto 3$ and maps everything else $\varepsilon \mapsto 0$, or (ii) the product $3 \times a * b$. Show that there is actually no danger because the power series (i) and (ii) are equal.

Question 2: Two Basic Definitions (6 points; 3 + 3)

This question requires you to implement two basic utilities in `rayuela` that correspond to definitions in the course notes.

-  We say a WFSA \mathcal{A} is **deterministic** if, for every state-letter pair $(q, a) \in Q \times \Sigma$, the transition function $\delta(q, a, q') \neq \mathbf{0}$ for at most one element $q' \in Q$ and, furthermore, there are no ε transitions in the automaton. Furthermore, require that \mathcal{A} has a single initial state. Implement the function `deterministic` in `rayuela/fsa/fsa.py` that checks whether the input FSA is deterministic.
-  We say a WFSA \mathcal{A} is **pushed** if, for every $q \in Q$, the following equation holds:

$$\left(\bigoplus_{(a, q') \in \Sigma \times Q} \delta(q, a, q') \right) \oplus \rho(q) = \mathbf{1} \quad (1)$$

Implement the function `pushed` in `rayuela/fsa/fsa.py` to check whether the input FSA is pushed.

Note: In class we always assume that ε is not part of Σ , and we define the set Σ_ε which also contains it. In `rayuela` the set `self.Sigma` from the FSA class is Σ_ε .


Question 3: Trimming a WFSA (14 points; 4 + 3 + 3 + 4)


In this question, we introduce a new particular class of semiring, **non-cancellative** semirings.

Definition 1. A **non-cancellative semiring** $\mathcal{W} = (\mathbb{K}, \oplus, \otimes, \mathbf{0}, \mathbf{1})$ is a semiring where the product $a \otimes b = \mathbf{0} \Leftrightarrow a = \mathbf{0} \vee b = \mathbf{0}$. In other words, a semiring is defined to be non-cancellative iff $\mathbf{0}$ is the only annihilator for the operation \otimes .


In the remaining part of the question, we will explore the optimization of WFSAs defined on non-cancellative semirings by removal of unnecessary states. We call this operation **trimming**. Importantly, trimming is not minimization because we will *not* merge any states; merging states will be discussed in lecture 4. We will walk you through the four steps necessary to trim a WFSAs.

- a) It is a well-known fact that the regular languages are closed under **reversal**. You are going to implement a constructive proof of this fact. Design an algorithm that reverses an FSA.


 Prove correctness of your algorithm, i.e., show that for any string $y \in \Sigma^*$ accepted by the original automaton, \overleftarrow{y} is accepted by its reversal. **Hint:** You can prove the statement by induction on the length of the string.

 Implement the function `reverse` in `rayuela/fsa/fsa.py` that computes the reversal.


- b) Given a WFSAs \mathcal{A} , a state $q \in Q$ is called **accessible** if it is reachable from a state q' with $\lambda(q') \neq 0$. Design an algorithm to compute the set of accessible states in an automaton.

 Implement the function `accessible` in `rayuela/fsa/fsa.py` that returns the set of accessible states.

- c) Given a WFSAs \mathcal{A} , a state $q \in Q$ is called **co-accessible** if a state q' with $\rho(q') \neq 0$ is reachable from q . Design an algorithm to compute the set of co-accessible states in an automaton.

 Implement the function `coaccessible` in `rayuela/fsa/fsa.py` that returns the set of co-accessible states.

- d) Provide an algorithm to trim a WFSAs.

 Explain why removing non-accessible and non-co-accessible states is the most we can remove. More formally, this means that for all remaining states, there exists a string that has an accepting path through that state. Explain why reversal is a useful subroutine in the construction.


 Implement the function `trim` in `rayuela/fsa/fsa.py` that trims an automaton.

Question 4: Union, Concatenation and Kleene Closure (13 points; 4 + 4 + 5)

Regular languages are famously closed under union and concatenation. In this question, you will implement algorithms for computing the union and concatenation of two arbitrary WFSAs. In so doing, you are providing a constructive proof of the aforementioned closure properties.

- a) Give an algorithm to compute the **union** of two WFSAs.

 Prove its correctness and analyze its runtime complexity.

 Implement the function `union` in `rayuela/fsa/fsa.py`.

- b) Give an algorithm to compute the **concatenation** of two WFSAs.

 Prove its correctness and analyze its runtime complexity.

 Implement the function `concatenate` in `rayuela/fsa/fsa.py`.

c) Give an algorithm to compute the **Kleene closure** of a WFSA.



Prove its correctness and analyze its runtime complexity.



Implement the function `kleene_closure` in `rayuela/fsa/fsa.py`.

Hint: Recall that the Kleene star of a language L is defined as follows: $L^* \stackrel{\text{def}}{=} \bigcup_{n=0}^{\infty} L^n$ where $L^n \stackrel{\text{def}}{=} \underbrace{L \circ L \cdots \circ L}_{n \text{ times}}$. In the weighted case, the Kleene star is defined identically—we just apply weighted concatenation. The proof of correctness should proceed by induction on n .