Formal Languages — CSE 4083 & CSE 5210

Grant Butler, CSE4083 Spring 2022, Computer Science B.S.

1 Deterministic Finite Automata

1. Consider a DFA $M = (Q, \Sigma, \delta, s, f)$ with States $Q = \{s, q_1, q_2, f\}$ where s is the start and f is the final state;

Alphabet $\Sigma = \{0, 1\}$ and transition function δ.

Construct a state transition table for δ (or you can draw a state transition diagram) that recognizes regular expressions that are binary strings and multiples of 3, for example, the strings:

$$0, 11, 110, 1001, 1100, ...$$

would be accepted strings, but

$$1, 10, 100, 101, ...$$

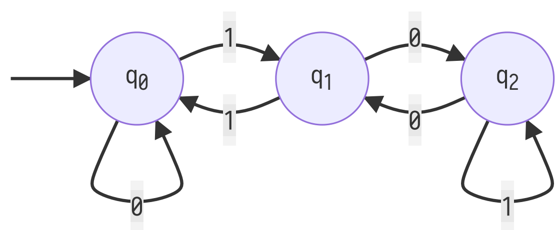
would not be accepted.

> *Hint: Think, if $n = 3k$ is a multiple of 3, then the next multiple of 3 is $3k + 3$.
>
> This could be accomplished by a transition from the current state to a next state by scanning 3 >ones.

Given language $L = \{0, 11, 110, 1001, 1100, ...\}$ and alphabet $\Sigma = \{0, 1\}$, the transition table for transition function δ is:

	0	1
q_0	q_0	q_1
q_1	q_2	q_0
q_2	q_1	q_2

There are three states in the DFA. The transition diagram is as follows:



2 Nondeterministic Finite Automata

2. Explain how any NFA (with λ (or ϵ) transitions) can be converted into a DFA that accepts the same language as that accepted by the NFA. That is, the expressive power of NFAs and DFAs are equivalent. This is known as the Rabin-Scott Theorem.

The Rabin-Scott Theorem states the list of languages DFAs can identify is the same as those that NFAs can recognize.

Then, we need to prove that they are in fact equal. We can use subsets to our advantage. If we prove that both the NFA and DFA are equivalent subsets of each other, then they are equivalent.
i.e.:

$$\begin{aligned} & L(DFA) \subseteq L(NFA) \\ & L(DFA) \supseteq L(NFA) \\ \implies & L(DFA) = L(NFA) \end{aligned}$$

Let the NFA be defined as $A = (Q, \Sigma, \delta, q_0, f)$ where f is the final state and q_0 is the initial state.
Then, we can build the DFA $B = \ A' = (Q', \Sigma, \delta, {q_0}', f')$ by doing the following:

- Replace states Q with the power set of Q, where $Q' := P(Q)$.*
- Set starting state of DFA to ${q_0}'$, where ${q_0}' := \{s_0\} \in Q$.*
- Then define the all words $\sigma_0, \sigma_1, \ldots, \sigma_n \in \Sigma*$ and all states $C \in Q'$ with transition function $\delta : (Q' \times \Sigma*) \to Q'$ by:

$$\delta(\ldots\delta(\delta(q_0, \sigma_0), \sigma_1), \ldots, \sigma_n) = C \in Q'$$
$$\iff$$
$$\exists\ q_0 \in S, \Delta(\ldots\Delta(\Delta(\{q_0\}, \sigma_0), \sigma_1), \ldots, \sigma_n) = s \in C \subseteq Q$$

This says that the DFA is equivalent in the state $C \in Q'$ if and only if the word for the NFA gives one of the states $s \in C \subseteq Q$. The final states f' is described by every $C \in Q'$ having at least one final state $c \in f$. Thus, the DFA A' accepts the same language and words as the NFA A, so they are equivalent.

* Note the power set, Q', has finitely many elements, similar to Q.

★ For a string $\sigma_0, \ldots, \sigma_n \in \Sigma * \{\sigma_0, \sigma_1, \ldots, \sigma_n\} \in \Sigma*$ the NFA is in a state q. $\Longrightarrow$ NFA A's beginning state $q_0 \in Q$ with $q = \Delta(\ldots(\Delta(s_0, \sigma_0), \sigma_1), \ldots, \sigma_n) \in F$.

3 Regular Expressions in the Programming World

3. Consider a programming language that has identifiers that start with a lowercase ASCII letter

$$A = \{a..z\}$$

followed by a string of 1 or more digits

$$D = \{0..9\}$$

or 1 or more lowercase ASCII letters. Show how to write this specification as a regular expression.

The expression:

```
[a-z]([0-9]+|[a-z]+)
```

gives the correct language, where `[a-z]` gives any lowercase ASCII character, and then `[0-9]+|[a-z]+` choses between any number of digits OR any number of lowercase ASCII characters to follow.

4 Closure Properties of Languages

Answer these *True* (T) or *False* (F) questions. Give a brief explanation of your answer

> ☙(for example, explain how to construct a machine that implements the property.)

4. Regular languages are closed under intersection.

TRUE. Two regular languages, L_1, L_2, are closed under intersection through this logic:

$$\begin{aligned} (L_1 \ \cap \ L_2) &= (L_1{}' \ \cup \ L_2')' \\ \implies (L_1' \ \cup \ L_2')' &= (\text{Regular}' \ \cup \ \text{Regular}')' && \because \ \text{L}_1 \text{ and } \text{L}_2 \rightarrow \text{regular} \\ &= (\text{Regular} \ \cup \ \text{Regular})' \\ &= (\text{Regular})' && \because \ \text{union of two regular} \rightarrow \text{regular} \\ &= \text{Regular} && \because \ \text{complement of regular} \rightarrow \text{regular} \end{aligned}$$

5. Regular languages are closed under Kleene-star.

TRUE. For a language to be regular, there has to be a repeating definite pattern, so for an existing DFA to accept a Kleene-star (Σ*), just start at the final state and repeat the transition of the starting state. The DFA will then accept the Kleene-star of that language.

5 Decision Properties of Languages

6. What does it mean to say that a “yes” or “no” question is *undecidable*?

If there isn't an algorithm that can answer yes/no in finite time, then it is undecidable.

***True* (T) or *False* (F):**

---

7. It is decidable whether or not the language of a DFA is empty or non-empty. Give an explanation of your answer.

TRUE:

Language accepted by a finite state machine being empty or non-empty is decidable because the DFA can be minimized and then see it it has a single state with no accepting state.

8. It is decidable whether or not the language of a DFA is finite or infinite.

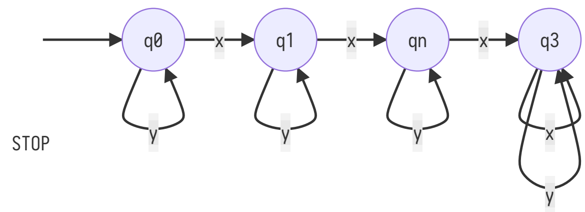
TRUE:

If there are any loops in a DFA, then it will accept an infinite language, and if there aren't any, it will accept a finite language.

9. It is undecidable whether or not the a string s is accepted by a DFA.

FALSE:

If a language L has a number of x's is 2:



The string *xxy* is valid, while the string *xxxy* is not.

xxy goes to final state q_n so it is accepted, but *xxxy* does not, so it is rejected.

10. It is decidable whether or not two regular languages L_1 and L_2 are equal. Give an explanation of your answer.

TRUE:

A DFA for L_1 should be able to accept every string for L_2, so they would be equal. If the DFA does not accept all strings for L_2, then they are not equal.

6 Equivalence Relations

11. On the set $\mathbb{N}$ of natural numbers define an equivalence relation $n \equiv m$ if and only if

$$n \mod 3 = m \mod 3$$

> *Hint: Recall any natural number n can be written as $n = 3q + r$ n with quotient q and remainder r. And
>
> $n \mod 3 = \{kr : k \in \mathrm{N}\}$ The set of all natural numbers that have a remainder of r when divided by 3.

Prove that ≡ is an equivalence relation on the set of natural numbers.

An equivalence relation is one with reflexive, transitive and symmetric relations. So,

- Reflexive ⟹ (a, a) ∈ $\mathbb{N}$

  Set of natural Numbers : 1, 2, 3, ..., n

  $n \equiv m \iff n \mod 3 = m \mod 3$

  ⟹ (1, 1), (2, 2), (3, 3), ..., (n, n).

  ⟹ reflexive

- Symmetric ⟹ if (a, b) ∈ $\mathbb{N}$, then (b, a) ∈ $\mathbb{N}$

  $n \equiv m \iff n \mod 3 = m \mod 3$

  Any set $\geq (2, 2) \in \mathrm{N}$ and if $2 \mod 3 = 2 \mod 3$

  ⟹ symmetric

- Transitive ⟹ (a, b), (a, c) ∈ $\mathbb{N}$, then (a, c) also ∈ $\mathbb{N}$

  $2 \mod 3 = 2 \mod 3$ && $2 \mod 3 = 3 \mod 3$

  $\Rightarrow 2 \mod 3 = 3 \mod 3$

  ⟹ transitive

  ∴ the relation is an equivalence relation.

7 The Pumping Lemma for Regular Languages

12. DFAs can't count to an arbitrary natural number! Use the pumping lemma for regular languages to show that language

$$EQ = \{w \in \{a, b\}* : w = a^i b^i\}$$

is not regular. Here the number of $a's$ in the prefix of w equals the number of $b's$ in the suffix of w.

For Regular Language L:

Z ∈ L and |Z| ≥ n

and satisfying:

i. Z = uvw

ii. |uv| ≤ |Z|

iii. |v| ≥ 1 and |u| ≠ 0

$$L = a^i b^i$$
$$Z = a^k b^k$$
$$|Z| = 2k \geq n$$

$$Z = a^{k-1}$$
$$\text{where } u = a^{k-1}$$
$$v = a$$
$$w = b^k$$

$$|a^{k-1}a| \leq |a^k b^k|$$
$$k \leq 2k$$

$$|v| \geq 1$$
$$|a| \geq 1$$
$$1 \geq 1$$
$$|u| \; != \; 0$$
$$|a^{k-1} \; != \; 0$$
$$k-1 \; != \; 0$$

$$uv^3w = a^{k-1}a^3b^k$$
$$\implies = a^{k+2}b^k$$

∴ It does not belong to language L, since the number of a's is greater than the number of b's ⟹ L is not a regular language.

8 Context Free Languages

13. Consider the CFG G defined by the productions:

$$S \to aS|Sb|a|b$$

Prove by induction that no string in $L(G)$ has ba as a sub-string.

> Hint: To show this do induction on the length of the strings.

S → aS:

a to the strings starting with a, then prefix is aa

a to strings starting with b, then the prefix is ab

no ba substring

S → Sb:

b to strings ending in a, then suffix ab

b to strings starting with b, then suffix is bb

no substring ba

∴ there is no substring ba for length k+1.

14. Give simple English language descriptions for the strings generated by the productions following four grammars $(G = (V, T, P, S))$:

1. $G_1 \to S|aS|a$

Strings with one or more a's.

2. $G_2 : S!aSa|aa|a$

Strings with one or more a.

3. $G_3 : S|SaS|a$

Strings with a's in the odd indexes.

Question	Points	Score
1	10	
2	15	
3	10	
4	5	
5	5	
6	5	
7	5	
8	5	
9	5	
10	5	
11	10	
12	10	
13	5	
14	5	
Total:	100	