Week Three, Discussion: Regular Expressions, Pumping Lemmas, Context-Free Grammars

HW2 Induction

Inductive Fallacies!

Prove: any maximum-leaf k-ary tree of height n (edges) has k^n leaves.

The base case is indeed $k^0 = 1$ which corresponds to a root, leaf node (possible!).

Wrong: Assume we have a tree of height n which has k^n leaves. Attach k nodes to all those leaves. Now we have $k \cdot k^n = k^{n+1}$ leaves. Done.

This is WRONG because you must actually *start* with a tree of height n+1 and show how to construct a maximum leaf tree.

Regular Expressions

RegEx in programming *IS NOT* like RegEx in formal theory.

- Programming RegEx is more powerful with features like forward lookahead and capturing.
- Formal RegEx does NOT have this.

Some notational things:

- $0^* := \{0\}* = \{\epsilon, 0, 00, 000, \dots\}$
- $\{0,1\}^* = \{\epsilon, 0, 1, 00, 01, 10, 11, \dots\}$

$$\begin{gathered} 01^* \cup 0^*1 = (\{0\} \circ \{1\}^*) \cup (\{0\}^* \circ \{1\}) \\ = \{0, 01, 011, 0111, \dots\} \cup \{1, 01, 001, \dots\} \end{gathered}$$

Rules for RegEx:

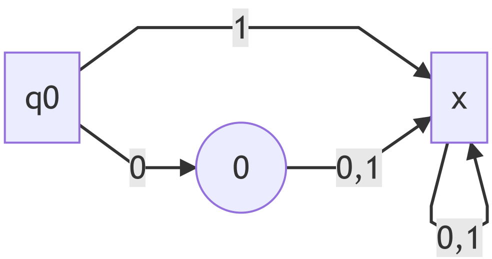
1. $a \in \Sigma$ is one character
    1. a is implicitly the set containing it i.e. $a := \{a\}$
2. $\emptyset := \{\}$
3. Union: $1 \cup \epsilon := \{1, \epsilon\}$

4. Concatenation: $(0 \cup \epsilon) \circ (1 \cup \emptyset)$
    1. $= \{0, \epsilon\} \circ \{1\}$
    2. $= \{01, (\epsilon)1\}$
5. Kleene-star 🌟 R^* where R is a regular expression
    1. All the above describe *finite* languages. No matter how many concatenations you do, it will be finite.
    2. Kleene-star describes *infinite* languages 🌃
    3. $((0 \cup \epsilon) \circ (1 \cup \emptyset))^*$
    4. $= (\{01, 1\})^*$
    5. $= \{\epsilon, 01, 1, 011, 101, \dots\}$

DFA from RegEx

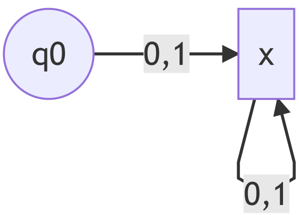
Literals:

$$R := \{0\}$$



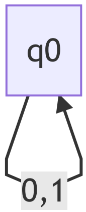
Empty string (accept nothing (accept literally nothing)):

$$R := \{\epsilon\}$$



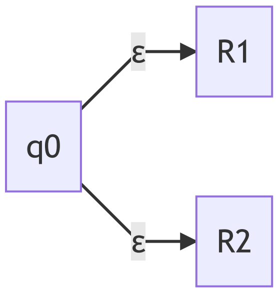
Empty language (accept nothing (reject everything)):

$$R := \emptyset = \{\}$$



Union of two regular expressions:

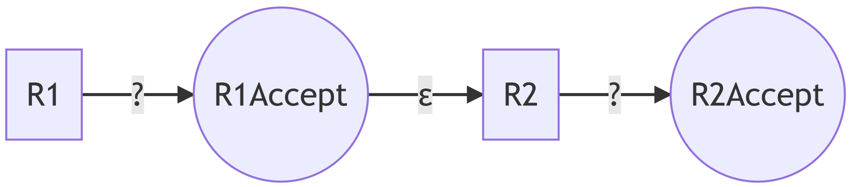
$$R := R_1 \cup R_2$$



Concatenation of two regular expressions:

$$R := R_1 \circ R_2$$

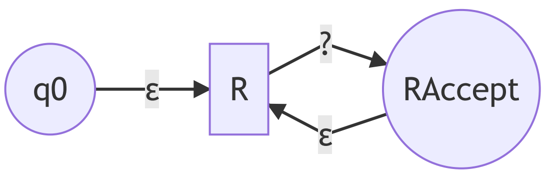
Accept the strings where the first half comes from R_1 and the second half comes from R_2



Kleene-star:

$$R^*$$

Basically, any string accepted by R will also be accepted by R^*, so we can just re-use the machine for R with an additional entry point to accept the empty string:



Showing a Language is Not Regular

Let L := $\{0^n1^n | n$ in N(atural Numbers)$\}$

Claim: L has no FSM.

Proof:

1. Suppose L has an FSM
2. This FSM has a finite number of states, k
3. Consider the string s := $0^{k+1}1^{k+1}$
4. We have k states
5. Our string s has 2(k+1) = 2k+2 symbols
6. By the 🕊️ pigeonhole principle 🏠 , we must have some loop within our FSM, since we have more than k symbols but only k states
7. We can divide our states into "x," "y," and "z"
    1. "x" precedes the loop
    2. "y" is the loop
    3. "z" follows the loop
8. len(xy) <= k+1
9. Per 8, x and y have only 0's (since they are not long enough to get past the 0^{k+1})
10. Since y is a loop, we can remove it and still get an accepted string
11. Therefore, xz is accepted
12. However, xz cannot be in L because xz does not have the same amount of 0's (since y has some amount of 0's)
13. ⚠️ Contradiction. ⚠️ (11) and (12).

Pumping Lemma 🚲 💨

Let L be a regular language. For this language there exists some number p, called the "pumping length" which we can also think of as the number of states, such that any string w in L can be written as w := xyz

1. xy^iz in L
    1. i = 0 means "pump down"
    2. i > 0 means "pump up"
2. len(y) > 0

1. y is essentially our "pump" which we can pump up to "overfill" our string to overwhelm the FSM accepting L

3. len(xy) <= p

You cannot choose one specific partition of w = xyz. Therefore, you must show ALL possible partitions of xyz.

You do select w itself though.