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:

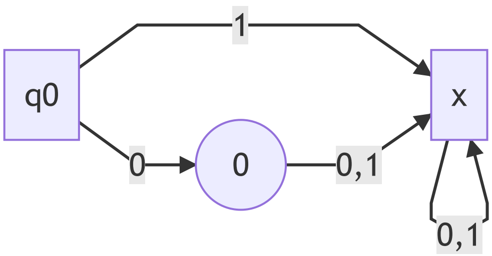
1. $a \in \Sigma$ is one character
    1. a is implicitly the set containing it i.e. $a := \{a\}$
2. $\emptyset := \{\}$
3. Union: $1 \cup \epsilon := \{1, \epsilon\}$

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. $= \{\epsilon, 01, 1, 011, 101, \ldots\}$

DFA from RegEx

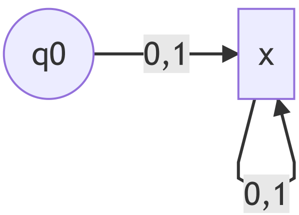
Literals:

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



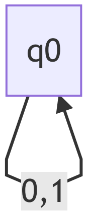
Empty string (accept nothing (accept literally nothing)):

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



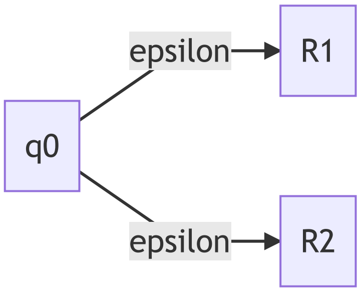
Empty language (accept nothing (reject everything)):

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



Union of two regular expressions:

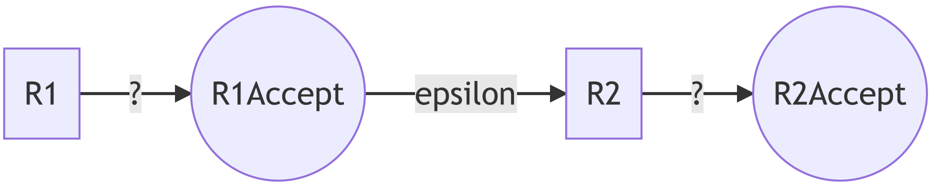
$$R := R_1 \cup R_2$$



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: