Week Two, Lecture One

Outline of Week Two

1. *Formal* model of finite automaton
    1. Finite state languages
    2. Examples
2. NFA - nondeterminism
    1. Classes of languages
    2. Closure properties
3. Regular expressions

Homework Review

Anything concatenated with the empty set is always, always the empty set.

$$\{\} \circ \_\_ = \{\}$$

Mathematically, "cardinality" refers to the size of a set. So, don't refer to the length of a string as its "cardinality." A word is a sequence, not a set. Just call it the length.

You cannot do $a \circ L$. Concatenation needs the same type for each operand i.e. $\{a\} \circ L$.

Strings

When your alphabet has certain symbols which are composed of other symbols in the alphabet, we can have ambiguous situations which have to be resolved by a compiler.

If $\Sigma = \{a, b, c, ..., z, <, >, \text{if}, \text{then}, ;, =\}$

then we can have ambiguity:

```
if counter < max then counter + 1
ifcounter<maxthencounter+1
```

The steps of a compiler are as follows:

1. A sequence of symbols is the input, which is then transormed by the **lexical scanner** into **tokens**
2. The tokens are transformed by the parser into an AST

The mathematical foundations are that a lexical scanner is a *finite automaton*! And a parser is a *pushdown automaton*.

- Lexer = FSM
- Parser = PDA

Substrings

Take a word w to be:

$$w = r \cdot s \cdot t$$

r, s, and t are all **substrings** of w. As are, rs and st.

r, rs are **prefixes** of w. t, and st are **suffixes**.

If u is a substring of w and $u \neq w$ and $u \neq \epsilon$, then u is a **proper substring** of w. The stipulation that u not be the empty string is per the Sipser textbook and not universally agreed upon.

Can rt be a substring of w?

▶ Answer
A **proper prefix** is a prefix which is not equal to the entire string. r is a proper prefix of w if $r \neq w$ and $\exists\, v \ni rv = w$ (reads "There exists a string v such that r concatenated with v is w").

- Note that v is not banned from being w! So, ϵ CAN BE a proper prefix of any non-empty word.

String Notation

Given Σ and some word w over Σ, then w^R is w backwards (reversed).

If $a \in \Sigma$ and $w \in \Sigma^*$, $\#(a, w)$ is the number of occurrences of a in w.

- $\#(a, aab) = 2$
- $\#(a, bb) = 0 = \#(a, \epsilon)$

Formal Definition of FSM

A DFA M is defined formally as $M = (Q, \Sigma, \delta : Q \times \Sigma \to Q, q_0, F \subseteq Q)$

The term F comes from historically when they were called "final states." Then people realized that the final state of the machine isn't always accepting! But the name stuck.

The initial configuration of M is the initial state q_0 with the input on the input tape and the input tape head on the first symbol.

Define a **computation** of M on input w where $w = w_1 ... w_m$ as:

- A sequence of states of M from the initial state to every given transition state such that:

$$\text{for } 0 \leq i \leq m-1 : \delta(q_i, w_{i+1}) = q_{i+1}$$

The computation is **accepting** if $q_m \in F$. Otherwise, it is rejecting.

We can also have the **transitive closure of delta** or **delta star** δ^*, defined recursively (*structural recursion* to be specific) on the input over $Q \times \Sigma^*$

$$\delta^*(q, \epsilon) \equiv q \text{ for } q \in Q$$

i.e. If you get the empty string on a node, you stay on that node

i.e. For a string of length 0, you don't move in the FSM.

$$\delta^*(q, ay) \equiv \delta^*(\delta(q, a), y) \text{ for } q \in Q, a \in \Sigma, y \in \Sigma^*$$

So, ay is a string split into two parts: a where a is one character and y where y is the rest of the string. Take a, with your current state, and feed it to the regular transition function, and get back some state q'. Take q' and the remainder of the string after the first character y, and recurse on that. You gradually chomp down on the string.

- This is an example of structural recursion because we have two cases: delta star where the string is empty and delta star where the string is at least one character

Delta star is NOT an FSM, it is just the recursive transition function. It is a *component* of an FSM.