**Construction and use of regular expressions and finite state automata**

**Turing machine TODO**

Theoretical

Pretty much any computer today is a turing machine

**Chomsky and Chomsky’s hierarchy TODO**

**Language and formal language**

Contains alphabet, rules, and words.

We must define an alphabet to create a formal language

An alphabet = **Σ**

**Σ =** {a, b}, our alphabet now only contains the options a and b. finite

**Σ** \*= Set of all possible words/strings in our alphabet – {a, b, aa, ab, ba, bb, aaa, …. forever }

**L =** our language. Always a subset of our **Σ** \* and possibly infinite

A language has grammar(syntax). This is used to proof that a given string is a member of your language

**Construcing a regular expression**

1. **Σ = {a, b, c}** we define our alphabet
2. **L** ⊆ **Σ\***  we decide that out language only can contain the letters of our alphabet, but with any possible combination (infinite)
3. we now make a rule that only input starting with a and ending with c will work in our machine.

**L={w | w begins and ends with c}**

**L= {a, abc, abc, abbc, abbbc, abbbbbbbbc……}**

1. **(ab\*c)** is the regular expression of our formal language L, which makes our formal language regular
2. Because we can make a regular expression on out formal language, we can now create a finite state machine as **Kleene’s theorem** states

Diagram

Description automatically generated

**Finite state automata**

the language accepted by finite automata can be simply defined by simple expressions known as Regular Expressions.

Graphical user interface, text

Description automatically generated

**Kleene’s theorem**

-For any language that is accepted by a FSA there is a regular  
expressions that defines the same language.

-For any language described by a regular expression there is an FSA  
that accepts the same language.

FSA recognize patterns (eg used in compilers) and regular expressions  
describe patterns (they describe a language) – the two things are  
equivalent.