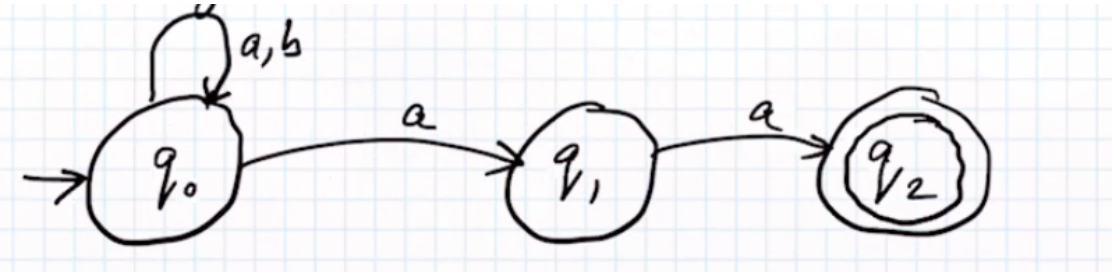
§1 05-05

§1.1 Quantifiers

Can think of $\exists$ as Eloise and $\forall$ as Abeland. Think of statements as a game between these two. The statement is true if Eloise has a winning strategy, and false otherwise (i.e. Abeland has a winning strategy).

§1.2 Design a machine that accepts string ending in "aa".



§1.3 NFA vs DFA

1. There can be multiple next states or no next state. NFA's can make choices. There are multiple computation paths corresponding to different choices. A word is accepted if there exists a path leading to an accept state.

If a machine jams, that is equivalent to rejection. Therefore only sequences that end with "aa" can end up at the accept state.

> **Theorem 1.1**
> The language accepted by any NFA is a regular language. i.e. could also have been accepted by a DFA.

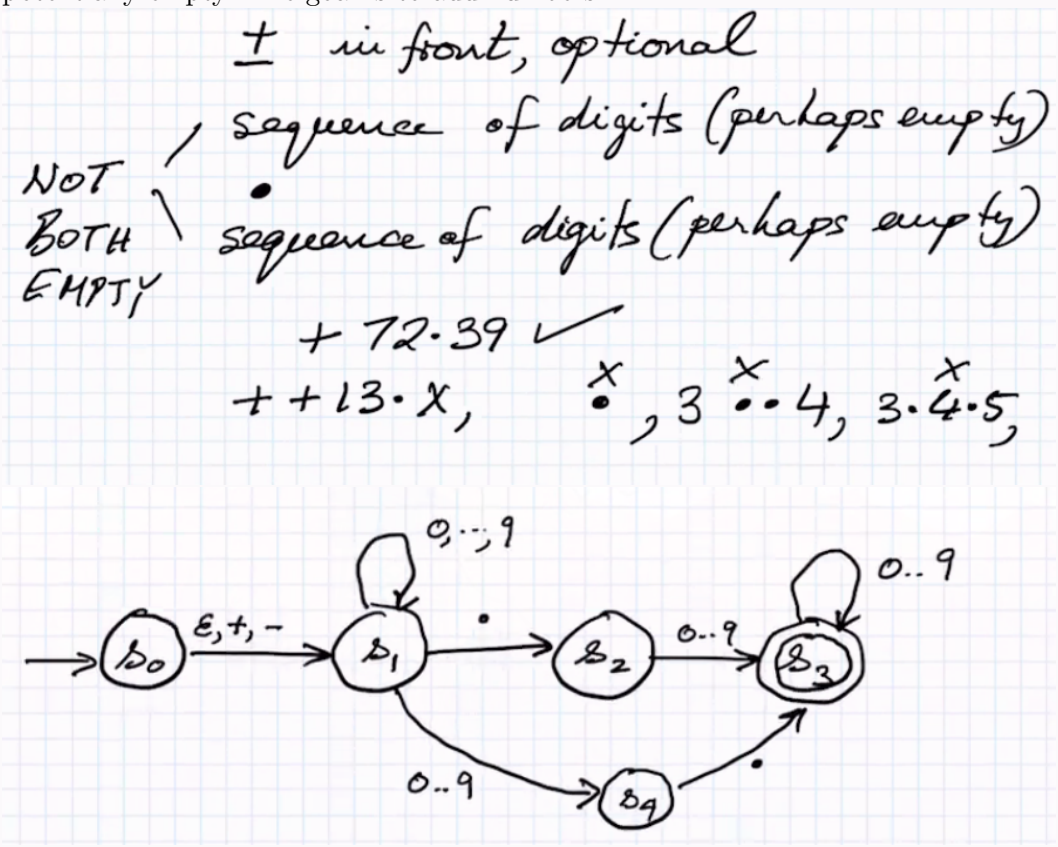
Remark 1.2. There is an algorithm for converting an NFA into an equivalent DFA. So you can go crazy with an NFA and know that it is still a regular language.

Note 1.3. Another extension: NFA + ϵ This machine can make jumps wihtout reading any input.

$$DFA \equiv NFA \equiv NFA + \epsilon \equiv \text{Regular Languages}$$

Example 1.4 (Decimal Numbers)

$\pm$ in front, optional. Two sequences of digits with one or the other (but not both) potentially empty. The goal is to add numbers.



1. Exactly plus, or exactly minus, or nothing. Otherwise the machine will jam.
2. ϵ is not just "hey I can jump to anywhere at any time"; it is part of the design of the machine.
3. There should be at least one thing after the decimal point.
4. No transition for another dot or plus or minus so any of those would cause the input to get rejected.
5. Allow for the possibility of ending with decimal point.
6. Need to two path to acceptance to distinguish between either before the decimal or after the decimal being empty, but not both.

This thing is doing a lot of guessing.

Note 1.5. In order to accept the language, it most both accept the right things and reject the wrong things. Cannot just accept everything.

Definition 1.6 (Formal definition of NFA)**.** An NFA is a 4-tuple:

1. Q: a finite set of states
2. $Q_0 \subseteq Q$: a (finite) set of start states

3. $F \subseteq Q$: a (finite) set of accept states

4. $\Delta : Q \times \Sigma \to 2^Q$.
   Or with ϵ. $\Delta : Q \times (\Sigma \cup \{\epsilon\}) \to 2^Q$

$$\Delta^* : Q \times \Sigma^* \to 2^Q$$
$$\Delta^*(q, \epsilon) = \{q\}$$
$$\Delta^*(q, w \cdot a) = \cup_{q' \in \Delta^*(q,w)} \Delta(q', a)$$

5. The accept state

$$L(N) = \{w \in \Sigma^* \mid \exists q \in Q_0, \Delta^*(q, w) \cap F \neq \varnothing\}$$

Theorem 1.7

Given an NFA $N = (Q, Q_0, F, \Delta)$ there is a DFA $M = (S, s_0, F', \delta)$ such that $L(N) = L(M)$.

Proof. Keep track of all the places where the m/c (machine) could be.

$$S = 2^q$$
$$s_0 = Q_0$$
$$F' = \{S \subseteq Q \mid S \cap F \neq \varnothing\}$$

Take the union of all the places where it could go.

$$\delta(s, a) = \cup_{q \in s} \Delta(q, a)$$

It is easy to see that this works (favorite line of math professors) as advertised. □

Remark 1.8. It is possible to have an exponential blow up in the number of states.

Note 1.9. $S = 2^Q$ refers to the set of all subsets of Q (power set). The power set of S can be identified with the set of all functions from S to a given set of two elements 2^S.

This can be understood where you represent each of the elements in the set as a digit in a binary number. Then, in order to find all possible sets, you would find all possible binary numbers with $|S|$ digits, which gives $2^{|S|}$.

For $NFA + \epsilon$ we define ϵ-closure of a set to be the places one can get to with an ϵ-move.

Example 1.10

Let L be a regular language. Let $\frac{1}{2}L = \{w \in \Sigma^* \mid \exists x \in \Sigma^*, wx \in L \ \wedge \ |w| = |x|\}$.

Look at left half of word and guess that there is something you can tack on to the right such that the entire word is something the original machine would recognize.

Assume DFA (S, s_0, F, δ) recognizes L. We will construct an NFA for $\frac{1}{2}L$.

$$Q = S \times S \times 2^S$$

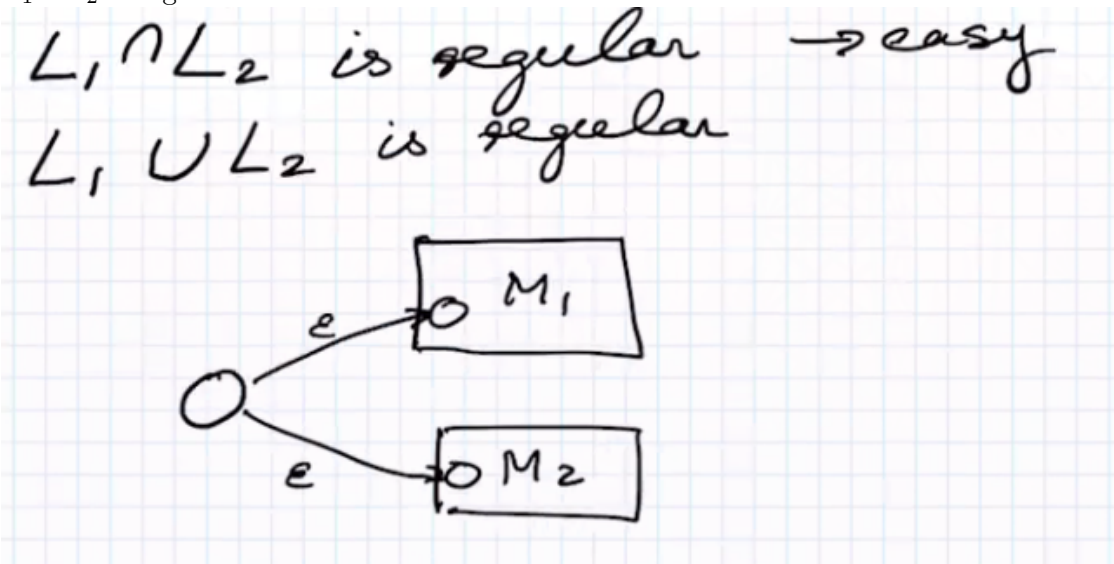
First index for . Second index for guessing where DFA will be when w ends. Third index is to try and track x the second half of the word.

$$Q_0 = \{(s_0, s, \{s\})\}$$
$$F' = \{(s, s, X) \mid X \cap F \neq \varnothing\}$$
$$\Delta((s, \overline{s}, X), a) = \{(s', \overline{s}, X') \mid \delta(s, a) = s'\}$$

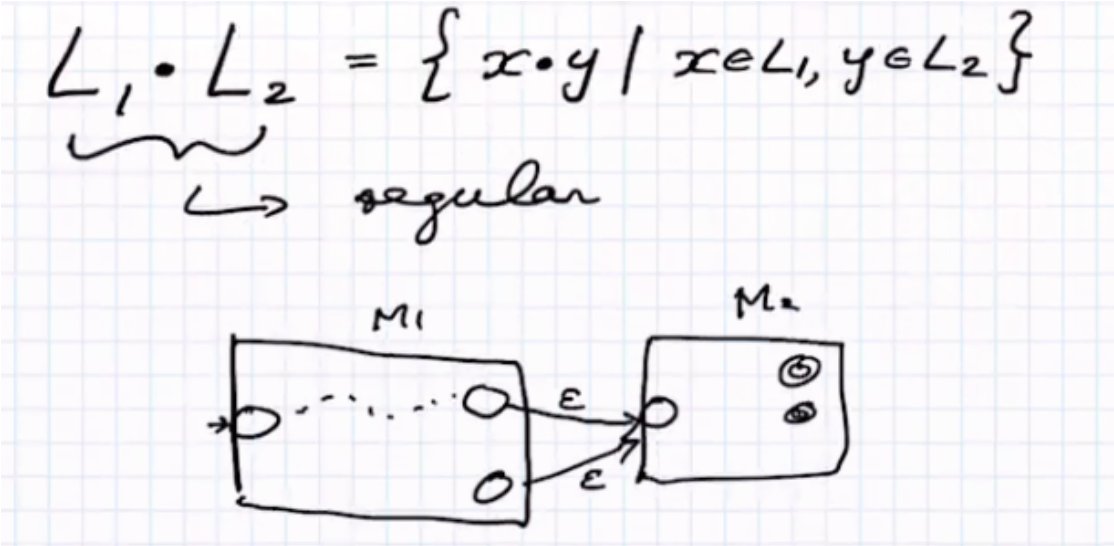
where X' is the set of states DFA can reach from x reading any symbol.

Closure properties of regular languages.

1. $L_1 \cap L_2$ is regular. Define a machine for L_1 and L_2. You create a state space with the cartesian product of states for L_1 and L_2. Then you go through in parallel and if it's in the accept state of both then it's accepted.
2. $L_1 \cup L_2$ is regular.



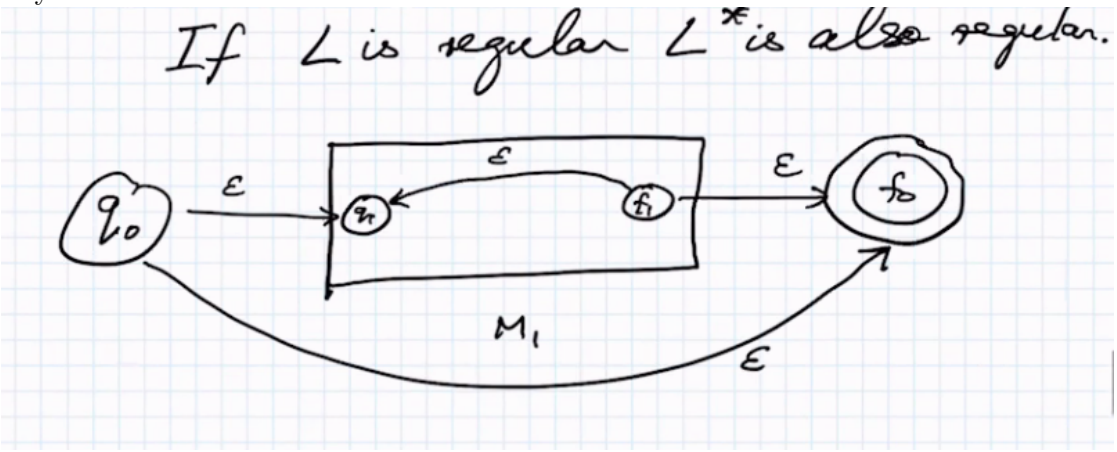
3. Concatenate language. $L_1 \cdot L_2 = \{x \cdot y \mid x \in L_1, y \in L_2\}$.



4. If L is a language, L^* is a new language where

$$L^* = \{w_1 \cdots w_k \mid w_i \in L\} \cup \{\epsilon\}$$

Basically take as many words as you like and glue them all together. Called the kleene (clay-knee) star operator. If L is regular $\Rightarrow L^*$ is regular but not the other way around.



5. Shuffle.

$$L_1 // L_2 = \{x_1 y_1 x_2 y_2 \cdots x_k y_k \mid x_1 x_2 \cdots x_k \in L_1,\ y_1 y_2 \cdots y_k \in L_2\}$$

§1.4 Regular Languages as an algebra

Definition 1.11 (Regular expressions)**.** Notation for describing regular languages in a machine independent fashion.

1. If R_1, R_2 are regular expressions, then $R_1 + R_2,\ R_1 \cdots R_2$ are regular expressions.

   If R is a regular expression, R^* is a regular expression.

A regular expression denotes a language.

$$\begin{aligned}
[\![\varnothing]\!] &= \varnothing, \\
[\![\epsilon]\!] &= \{\epsilon\}, \\
[\![R_1 \cdot R_2]\!] &= [\![R_1]\!] \cdot [\![R_2]\!], \\
[\![R^*]\!] &= ([\![R]\!])^* / [\![\overline{R}]\!] = \overline{([\![R]\!])}
\end{aligned}$$

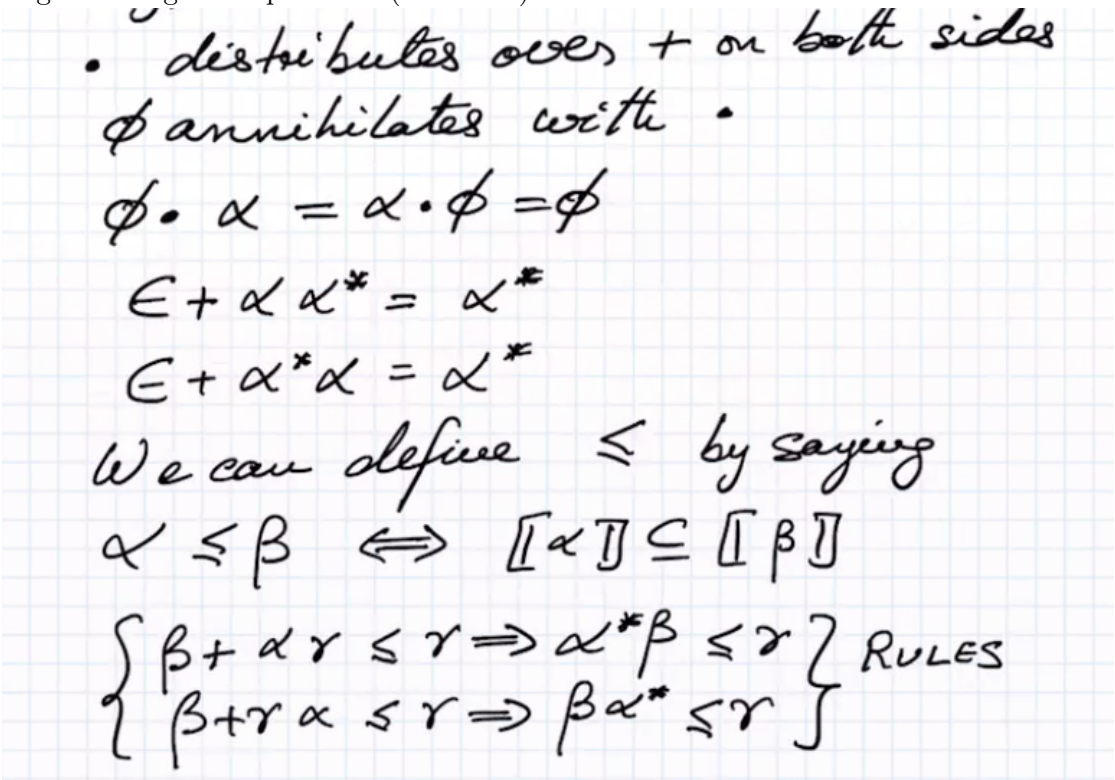
Example 1.12
$(aa + b)^* = \{aa, \epsilon, b, aab, bbaab, \cdots\}$

Theorem 1.13 (Kleene Theorem)
A language is regular iff it can be described by a regular expression.

Proof. Easy to see that reg exp implies regular. We already showed relevant closure properties. Reverse direction later if even. □

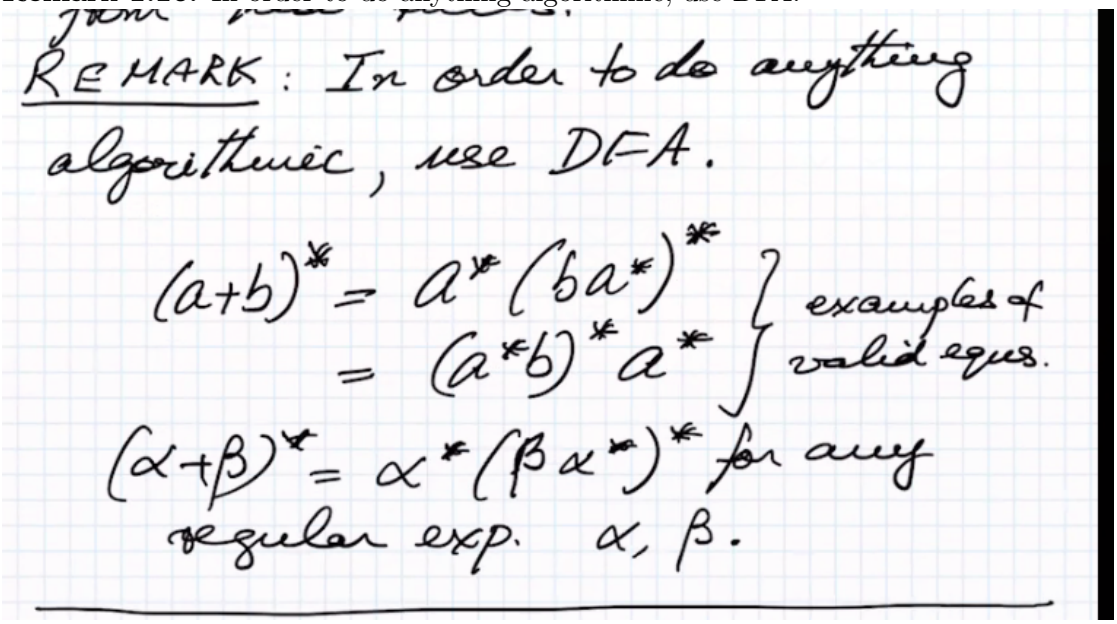
Note 1.14. "+" is notation for "or".

We have a set of equational axioms for reasoning about equality of regular expressions. Reg set of regular expressions (without -).



Theorem 1.15
These rules are complete. i.e. every valid equation can be derived from these rules.

Remark 1.16. In order to do anything algorithmic, use DFA.



§1.5 From DFA to reg exp

Enumerate the states of a DFA $1, 2, \ldots, k$. For every pair of states i, j we define R_{ij}^n as a regular expression describing all strings that take the DFA from i to j only traversing states $1, \ldots, n$ along the way.

Example 1.17

R_{ij}^0 would represent only direct jumps from i to j. Either the empty word, or a single symbol.

We can compute R_{ij}^{n+1} from R_{ij}^n. Then you can build the DFA by union all regular expressions describing jumps from start to accept states.