Definition: Given an alphabet Σ a **language** L over Σ is a subset $L \subseteq \Sigma^*$.

abab ⇝ { abab, aabab, babab, ... }

abc ≠ cba infinite language

Recall that given a DFA

$$\delta : Q \times \Sigma \to Q$$

we defined its extension from input symbols to words

$$\hat{\delta} : Q \times \Sigma^* \to Q.$$

$$Q \to (\Sigma^* \to Q)$$

accepting states $\quad final : Q \to Bool$

$$Q \to (\Sigma^* \to Bool)$$

language

denotational semantics of automata:
given an automaton, there is a function

$$[\![-]\!] : Q \to (\Sigma^* \to Bool)$$

$[\![q]\!]$ the language accepted in state q
the "future" of q

Recall from the previous lecture (I put in some screenshots):

```
initial = 0

final 1 = True
final 0 = False

delta  0 'a' = 0
delta  0 'b' = 1
delta  1 'a' = 0
delta  1 'b' = 1
```

```
run delta q [] = q
run delta q (c:cs) = run delta (delta q c) cs
```

To summarise the text above in two equations:

$$\hat{\delta}(q, \epsilon) = q$$
$$\hat{\delta}(q, w) = \delta(\hat{\delta}(q, x), a) \qquad w = xa$$

In fact, as a specification for a recursive program, I prefer the mathematically equivalent but tail recursive definition

$$\hat{\delta}(q, \epsilon) = q$$
$$\hat{\delta}(q, w) = \hat{\delta}(\delta(q, a), x) \qquad w = ax$$

In Haskell, the latter can be written as [15]

Aims

Given DFAs A_1, A_2 learn how to construct

- the negation of A_1
- the sequential composition of A_1 and A_2
- the asynchronous parallel composition of A_1 and A_2
- the synchronized parallel composition of A_1 and A_2
- the Kleene star A_1^*

closely related to regular expressions

Excursion:

Algebra of Automata

Algebra of Numbers

operations $+, *, -, /, \ldots$

equations $1 + 2 * 3 = 1 + 6 = 7$ } Numbers

negation $\approx$ $-$

composition $\approx$ $*$

parallel comp $\approx$ $+$

Kleene star $\approx$ $\ldots$ loop } Automata

"negation" of automata

$L \in \Sigma^*$ $\quad L^c = \Sigma^* \setminus L$

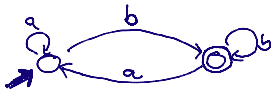
all words not in L

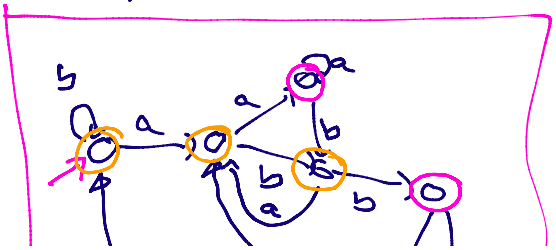
Given $A = (\Sigma, Q, \delta, q_0, F)$

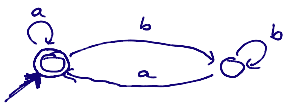
Σ: input symbols; Q: states; δ: transition function; q_0: initial state; $F \subseteq Q$: final states

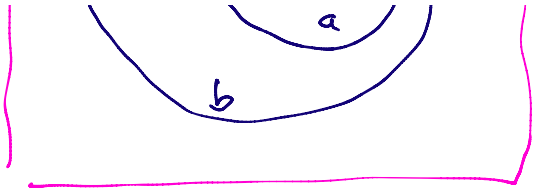
$[\![q_0]\!] = L$

what is A^c ? $\quad [\![q_0]\!] = L^c = \Sigma^* \setminus L$

$A =$ 



$A^c =$ 

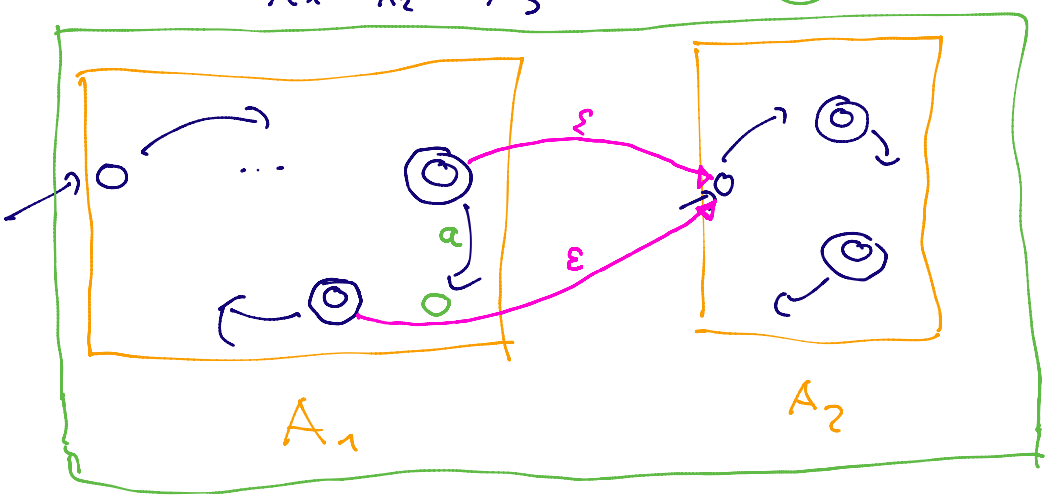


$$A = (\Sigma, Q, \delta, q_0, F)$$

$$A^c = (\Sigma, Q, \delta, q_0, Q \setminus F)$$

Exle: $\underbrace{abc}_{A_1}\ \underbrace{cba}_{A_2}$ SEQUENTIAL COMPOSITION

Exle: $\underbrace{abc}_{A_1}\ \underbrace{\varepsilon}_{A_2}\ \underbrace{cba}_{A_3}$ not (D)FA



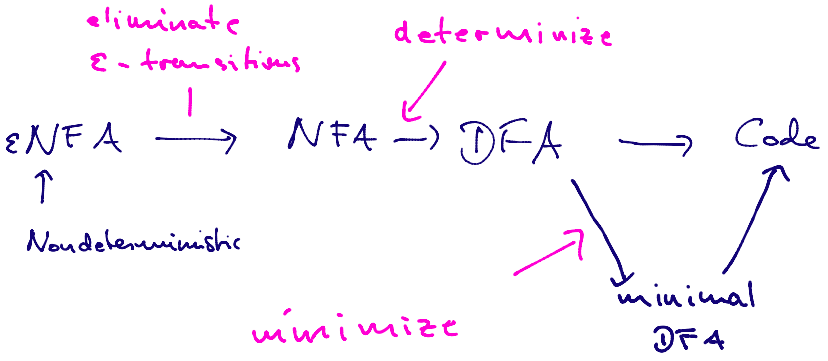
$A_1 ; A_2$

new idea
transitions that don't read input

("goto" without reading)

ε - transitions

NEXT LECTURE



ASYNCHRONOUS COMPOSITION UNION OF LANGUAGES

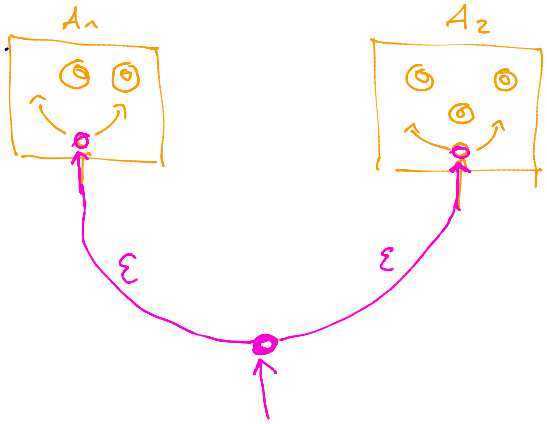
"aa" or "abb"

A_1 A_2

$L_1 \cup L_2 = \{ w \in \Sigma^* \mid w \in L_1 \text{ or } w \in L_2 \}$

L_1 L_2

$L_1 \cup L_2$ union of languages



Kleene star, repetition of languages

Exle: abc * cba

↓

wildcard ? loop

Given A that accepts L

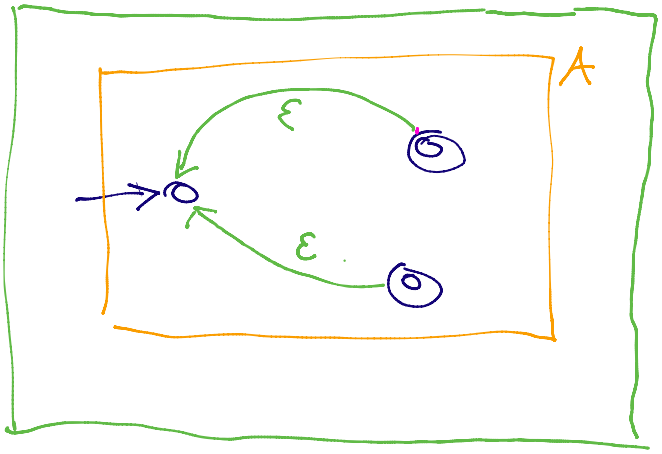
We want to construct A^* that accepts L^*. What is L^* ?

Exle Σ^* repeating: all symbols in Σ

L^* repeating words in L

$$L^* \stackrel{\text{def}}{=} \bigcup_{n \in \mathbb{N}} L^n$$

$$w \in L^n \iff w = w_1 w_2 w_3 \cdots w_n \quad \forall i .\ w_i \in L$$



How it works: A^* recognizes L^*

$$L^* = \{ w_1 w_2 w_3 \dots w_n \mid n \in \mathbb{N},\ w_i \in L \}$$