

---

# Finite Automata with $\varepsilon$ -Transitions & Regular expressions

Martin Fränzle

Informatics and Mathematical Modelling  
The Technical University of Denmark

02140 Languages and Parsing, MF, Fall 2004 – p.1/22

## What you'll learn

---

### 1. $\varepsilon$ -NFA:

- Definition
- Equivalence to NFA & DFA

### 2. Regular expressions:

- Syntax
- Semantics

### 3. The Kleene theorem:

Regular expressions are equivalent to finite automata:

- Any finite automaton can be translated to a regular expression describing the same language,
- any regular expression can be translated to a finite automaton describing the same language.

02140 Languages and Parsing, MF, Fall 2004 – p.2/22

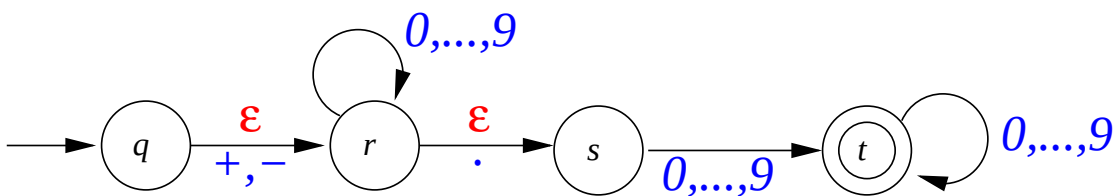
## Finite Automata with Autonomous Transitions:

### $\epsilon$ -NFAs

02140 Languages and Parsing, MF, Fall 2004 – p.3/22

## $\epsilon$ -NFAs — informally

- Have all features of an NFA, plus
- the freedom to do autonomous moves without consuming input, i.e. “silent” moves



02140 Languages and Parsing, MF, Fall 2004 – p.4/22

## $\varepsilon$ -NFAs — formally

An  $\varepsilon$ -NFA  $A = (Q, \Sigma, \delta, q_0, F)$  consist of

1. a finite set of states  $Q$ ,
2. a finite set of input symbols  $\Sigma$ , called the “alphabet of  $A$ ”,
3. a transition function  $\delta : Q \times (\Sigma \cup \{\varepsilon\}) \rightarrow \mathcal{P}(Q)$ , (thus assigning a – possibly empty – successor state set to silent moves also),
4. a start state  $q_0 \in Q$ ,
5. a set of final or accepting states  $F \subseteq Q$ .

The automaton

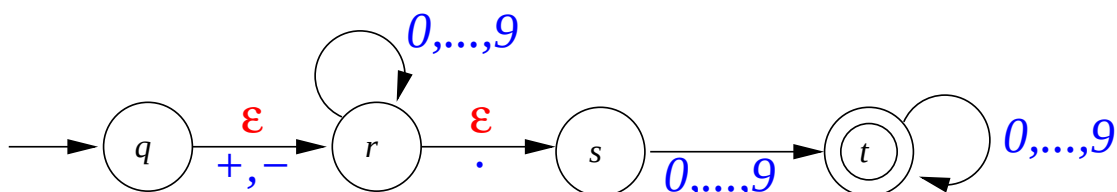
1. starts in state set  $\{q_0\}$ ,
2. proceeds from state  $p$  to state set  $\delta(p, a)$  if the current input is  $a$ , or
3. proceeds from state  $p$  to state set  $\delta(p, \varepsilon)$  *irrespective of current input*,
4. accepts a string (= sequence of symbols) iff that sequence of inputs drives it from  $q_0$  to a state set *containing* some  $q \in F$ .

02140 Languages and Parsing, MF, Fall 2004 – p.5/22

## Transition tables

Do now assign a successor state-set for  $\varepsilon$  also:

|                 | $\varepsilon$ | $+, -$      | $.$         | $0, \dots, 9$ |
|-----------------|---------------|-------------|-------------|---------------|
| $\rightarrow q$ | $\{r\}$       | $\{r\}$     | $\emptyset$ | $\emptyset$   |
| $r$             | $\{s\}$       | $\emptyset$ | $\{s\}$     | $\{r\}$       |
| $s$             | $\emptyset$   | $\emptyset$ | $\emptyset$ | $\{t\}$       |
| $*t$            | $\emptyset$   | $\emptyset$ | $\emptyset$ | $\{t\}$       |



02140 Languages and Parsing, MF, Fall 2004 – p.6/22

## The $\varepsilon$ closure

Given  $\delta : Q \times (\Sigma \cup \{\varepsilon\}) \rightarrow \mathcal{P}(Q)$  and  $q \in Q$ , we define  $\text{EClose}(q)$  to be the *smallest subset of  $Q$*  satisfying:

**Base case:**  $q \in \text{EClose}(q)$ ;

**Recursion:** If  $p \in \text{EClose}(q)$  then  $\delta(p, \varepsilon) \subseteq \text{EClose}(q)$ .

The function  $\text{EClose} : Q \rightarrow \mathcal{P}(Q)$  assigns to a state  $q$  all states that are reachable from  $q$  via silent moves only.

$\text{EClose}(q)$  is called the  $\varepsilon$ -closure of  $q$ .

## Extending the trans. fct. to strings

Given  $\delta : Q \times (\Sigma \cup \{\varepsilon\}) \rightarrow \mathcal{P}(Q)$ , we define a function  $\hat{\delta} : Q \times (\Sigma^*) \rightarrow \mathcal{P}(Q)$  via recursion:

**Base case:**  $\hat{\delta}(q, \varepsilon) \stackrel{\text{def}}{=} \text{EClose}(q)$  for each  $q \in Q$ ;

**Recursion:**  $\hat{\delta}(q, xa) \stackrel{\text{def}}{=} \bigcup_{p \in \hat{\delta}(q, x)} \bigcup_{r \in \delta(p, a)} \text{EClose}(r)$  for each  $q \in Q$ ,  
 $x \in \Sigma^*$ ,  $a \in \Sigma$ .

## The language of an $\varepsilon$ -NFA

**Def:** The language  $L(A)$  of an  $\varepsilon$ -NFA  $A = (Q, \Sigma, \delta, q_0, F)$  is

$$L(A) \stackrel{\text{def}}{=} \{w \in \Sigma^* \mid \hat{\delta}(q_0, w) \cap F \neq \emptyset\}.$$

The language  $L(A)$  thus is the set of strings that drive  $A$  from the initial state to a set of states containing some accepting state, perhaps via a path entailing silent moves.

## Determinizing $\varepsilon$ -NFA

$\varepsilon$ -NFA can be determinized by essentially the same construction as for NFA — only the  $\varepsilon$ -closure has to be added:

1. The **start state** of the DFA is  $\text{EClose}(q_0)$  instead of  $\{q_0\}$ ;
2. the **transition function** of the DFA is

$$\delta_D(q, a) = \bigcup_{r \in q} \bigcup_{s \in \delta(r, a)} \text{EClose}(s) \text{ instead of}$$

$$\delta_D(q, a) = \bigcup_{r \in q} \delta(r, a).$$

Thus, DFA, NFA, and  $\varepsilon$ -NFA recognize the same languages!

---

## Regular expressions

### Syntax & semantics

02140 Languages and Parsing, MF, Fall 2004 – p.11/22

## Why regular expressions?

---

- Automata are low-level programming:
  - you have to figure out the appropriate number of states (and to name them),
  - you are linking them together via “*spaghetti code*”
- Regular languages are higher-level programs:
  - you don’t talk about states — you talk about (sets of) strings,
  - only nested iteration, like in “*structured programming*”

02140 Languages and Parsing, MF, Fall 2004 – p.12/22

# Kleene closure

Let  $L \subseteq \Sigma^*$ .

**Def.:** The *Kleene closure*  $L^*$  of  $L$  is the set

$$L^* \stackrel{\text{def}}{=} \left( \begin{array}{l} \{\varepsilon\} \\ \cup \{w_1 \mid w_1 \in L\} \\ \cup \{w_1 \cdot w_2 \mid w_1 \in L, w_2 \in L\} \\ \cup \{w_1 \cdot w_2 \cdot w_3 \mid w_1 \in L, w_2 \in L, w_3 \in L\} \\ \cup \{w_1 \cdot w_2 \cdot w_3 \cdot w_4 \mid w_1 \in L, w_2 \in L, w_3 \in L, w_4 \in L\} \\ \vdots \end{array} \right)$$

**N.B.:**  $L^*$  is called a *closure* because  $(L^*)^* = L^*$ .

02140 Languages and Parsing, MF, Fall 2004 – p.13/22

# Regular expressions

- The constants  $\emptyset$  and  $\varepsilon$  are regular expressions, denoting the languages  $L(\emptyset) = \emptyset$  and  $L(\varepsilon) = \{\varepsilon\}$
- For any symbol  $a$ , the expression  $a$  is a regular expression, denoting the language  $L(a) = \{a\}$
- If  $E$  and  $F$  are regular expressions then so is  $E + F$ , denoting the language  $L(E + F) = L(E) \cup L(F)$
- If  $E$  and  $F$  are regular expressions then so is  $EF$ , denoting the language  $L(EF) = \{vw \mid v \in L(E), w \in L(F)\}$
- If  $E$  is a regular expression then so is  $E^*$ , denoting the language  $L(E^*) = (L(E))^* = L(\varepsilon) \cup L(E) \cup L(E E) \cup L(E E E) \cup \dots$
- If  $E$  is a regular expression then so is  $(E)$ .

**Precedence rules:**  $*$  comes before juxtaposition comes before  $+$ .

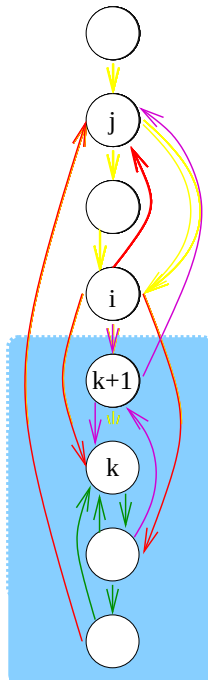
02140 Languages and Parsing, MF, Fall 2004 – p.14/22

## Regular expressions

### Equivalence to finite automata

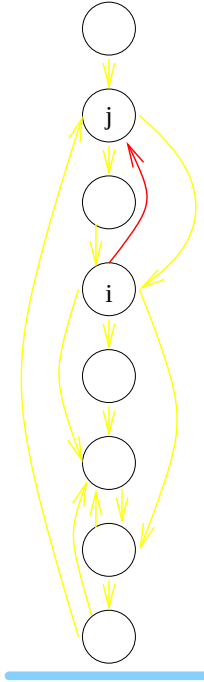
02140 Languages and Parsing, MF, Fall 2004 – p.15/22

## The $i, j, k$ construction



02140 Languages and Parsing, MF, Fall 2004 – p.16/22

## The $i, j, k$ construction: base case



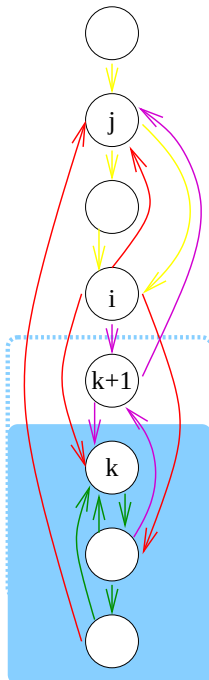
If  $i \neq j$ :

- $R_{ij}^{(0)} = \emptyset$  iff  $\{\sigma \in \Sigma \mid \delta(i, \sigma) = j\} = \emptyset$ ,
- $R_{ij}^{(0)} = a$  iff  $\{\sigma \in \Sigma \mid \delta(i, \sigma) = j\} = \{a\}$ ,
- $R_{ij}^{(0)} = a_1 + \dots + a_n$  iff  $\{\sigma \in \Sigma \mid \delta(i, \sigma) = j\} = \{a_a, \dots, a_n\}$ .

If  $i = j$ :

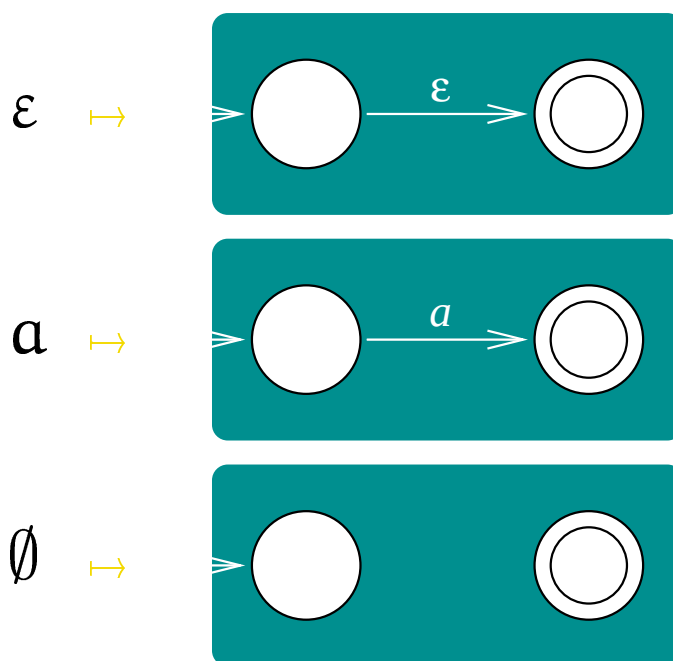
- $R_{ij}^{(0)} = \varepsilon$  iff  $\{\sigma \in \Sigma \mid \delta(i, \sigma) = j\} = \emptyset$ ,
- $R_{ij}^{(0)} = \varepsilon + a$  iff  $\{\sigma \in \Sigma \mid \delta(i, \sigma) = j\} = \{a\}$ ,
- $R_{ij}^{(0)} = \varepsilon + a_1 + \dots + a_n$  iff  $\{\sigma \in \Sigma \mid \delta(i, \sigma) = j\} = \{a_a, \dots, a_n\}$ .

## The $i, j, k$ construction: step $k \rightsquigarrow k + 1$



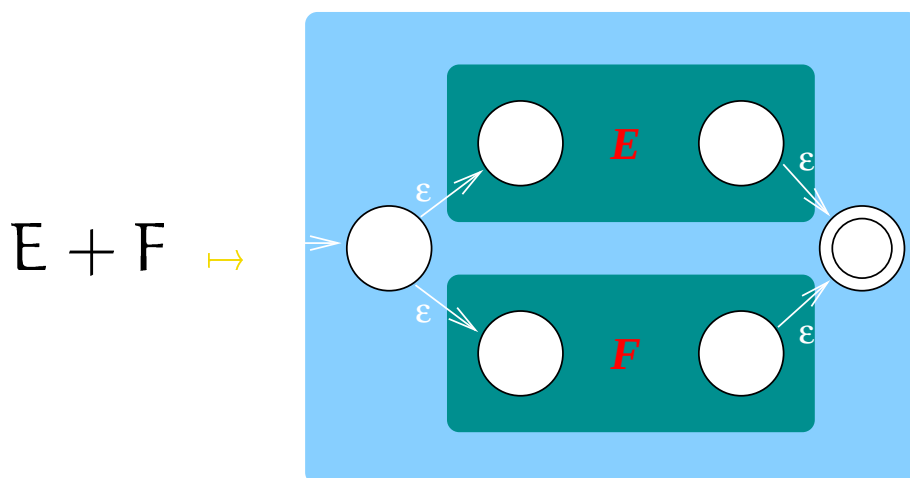
$$R_{ij}^{(k+1)} = R_{ij}^{(k)} + R_{i(k+1)}^{(k)} \left( R_{(k+1)(k+1)}^{(k)} \right)^* R_{(k+1)j}^{(k)}$$

## Regular expressions to $\epsilon$ -NFAs



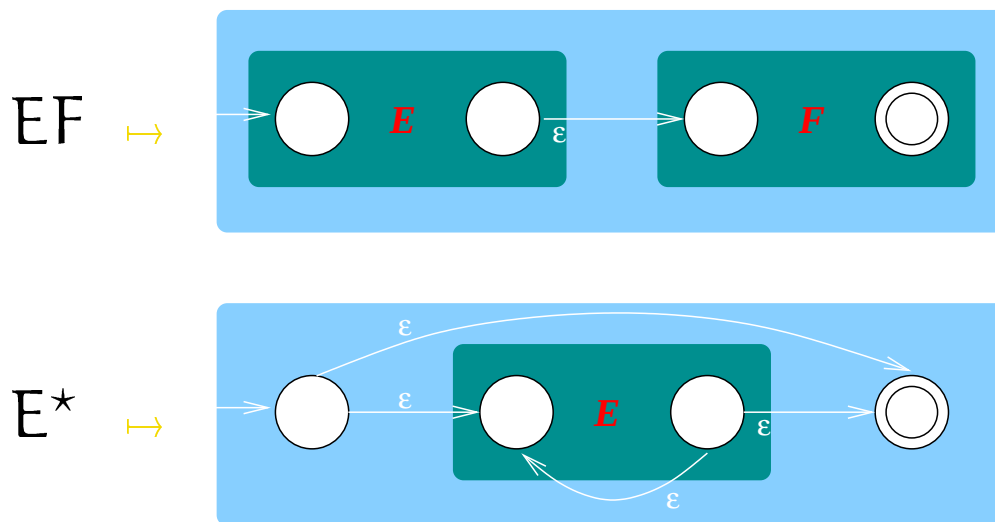
02140 Languages and Parsing, MF, Fall 2004 – p.19/22

## Regular expressions to $\epsilon$ -NFAs



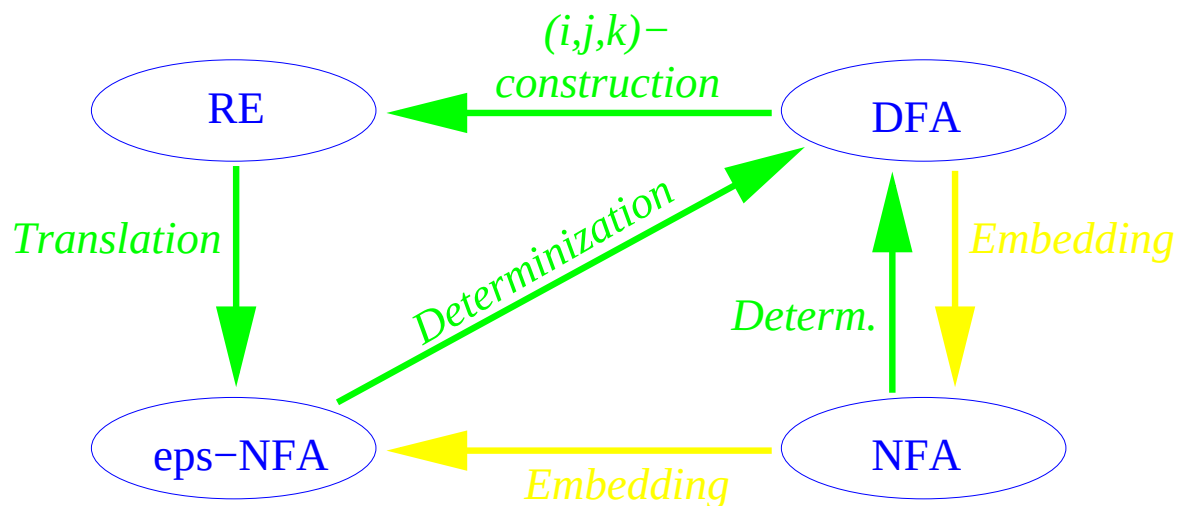
02140 Languages and Parsing, MF, Fall 2004 – p.20/22

## Regular expressions to $\epsilon$ -NFAs



02140 Languages and Parsing, MF, Fall 2004 – p.21/22

## Kleene's Theorem



Regular expressions, DFA, NFA, and  $\epsilon$ -NFA define the same class of languages: the *regular languages*.

02140 Languages and Parsing, MF, Fall 2004 – p.22/22