Regular Expressions and Automata

Introduction

Regular expressions: convenient to specify a text pattern

Finite automata: efficient to match a text pattern

Both are equivalent

Outline

- Regular Expressions to NFA
  - Illustration
  - Algorithm
- NFA to Regular Expressions

Regular Expressions to NFA

Regex and NFA:

- To say they are equivalent, the **langauge** produced by the regex $\boldsymbol{L}(\mathbf{R})$ must be equal to the **language** produced by the NFA $\boldsymbol{L}(\mathbf{N})$.

Given a Regex, we need to convert the following parts into NFA

- Basis Regular Expressions:
  - Empty set
  - Empty string
  - Input symbol
- Regular Operations:
  - Concatenation
  - Union
  - Kleene-Closure

Basis Regular Expressions:

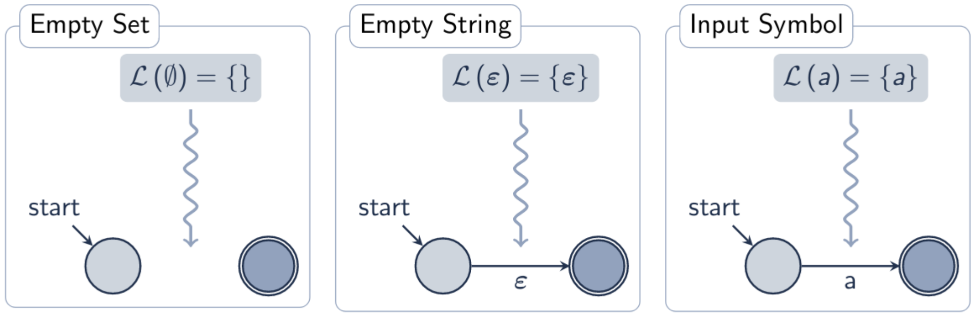


Figure 1: Basis Regular Expressions -> NFA

Regular Operations:

Before handling operations, convert the basis regular expressions into a NFA using the conversions shown in figure 1.

Example:

Given REGEX: $e_1e_2e_3$*, where e_i are expressions

Convert each e_i to NFAs so that they are N_1, N_2, N_3

Then handle the concatenation of N_1 and N_2 and the Kleene Closure of N_3

Concatenation

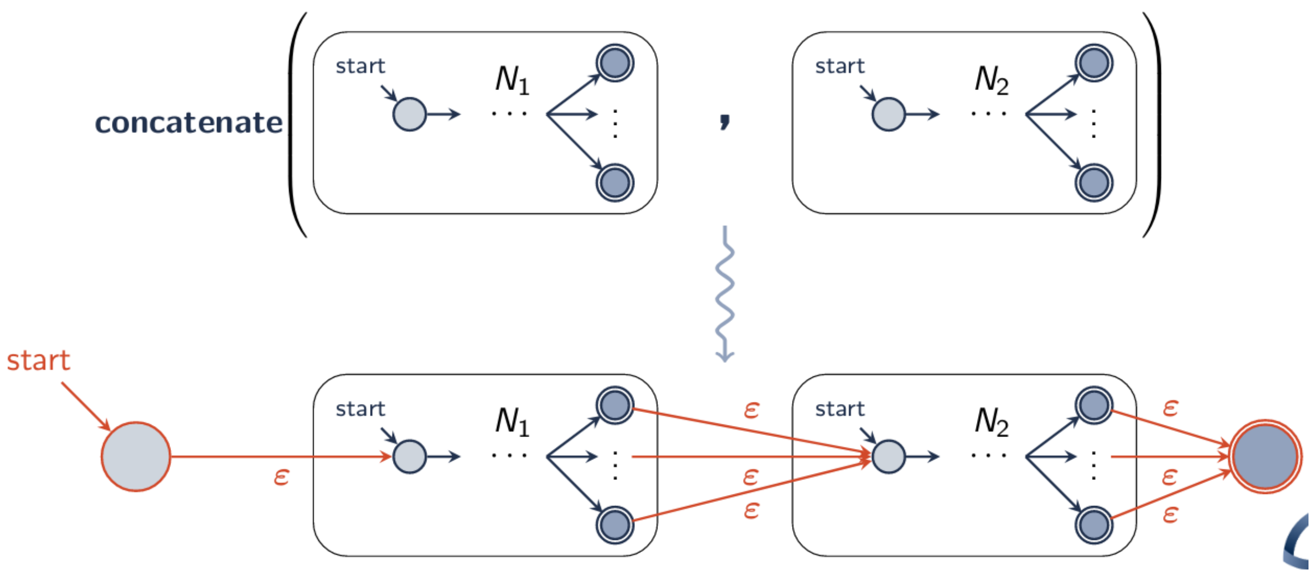


Figure 2: Example Conatenation: N_1 and N_2 are basis expressions that were converted to NFAs

Union

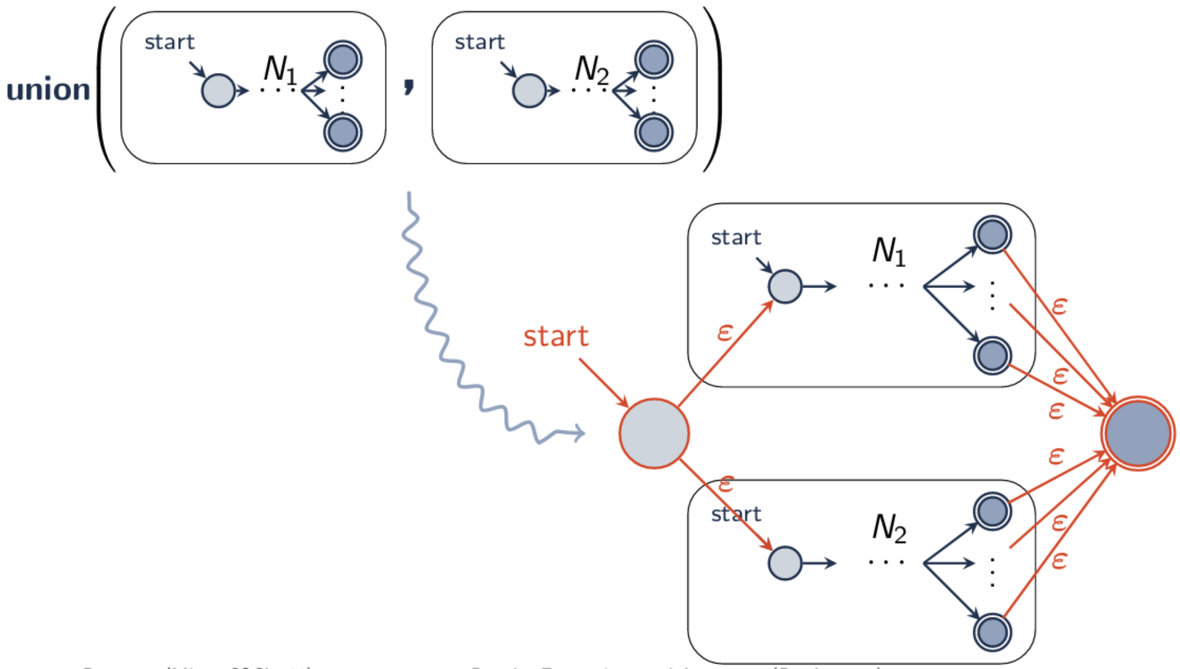


Figure 3: Example Union

Kleene

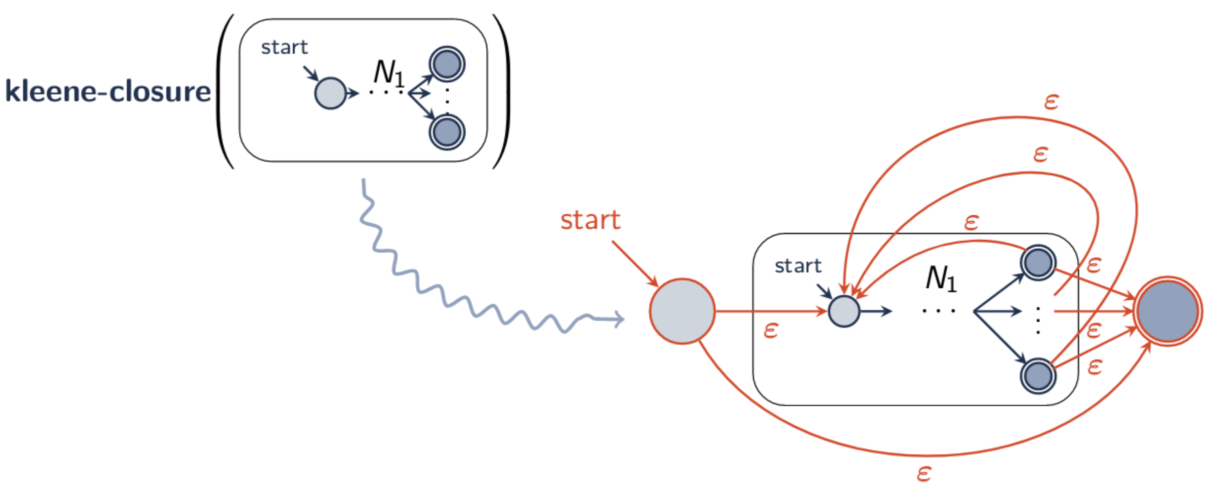
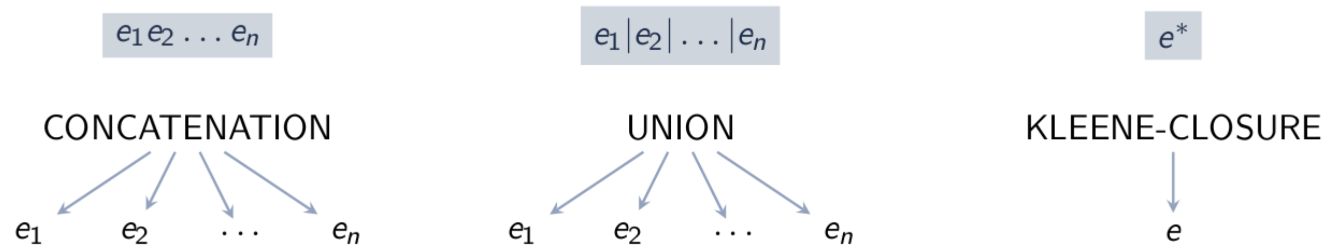


Figure 4: Example Kleene

Abstract Syntax of Regular Expressions



Concatenation → **(:concatenation e-1 . . . e-n)**

Union → **(:union e-1 . . . e-n)**

Kleene → **(:kleene e)**

McNaughton-Yamada-Thompson Algorithm

Input: A regular expression R

Output: The equivalent NFA N, where $L(\mathrm{R}) = L(\mathrm{N})$

Overview: Recursive construction:

> Base case: If R is a basis element ($\emptyset$, ϵ, a) construct the equivalent NFA directly
>
> Recursive case: Otherwise, recurse on the children of R and combine the results according to the operation of R

Algorithm 1: MYT

```
Input:  R ;                                              // Regex
Output: (Q, E, s, a) ;               // states, edges, start, accept
if R is ∅, ε, or symbol then // Base Case
    return MYT-base(R);
else if car(R) = KLEENE-CLOSURE then
    return kleene-closure(MYT(child(R)));
else if car(R) = CONCATENATION then
    return MYT-concatenate(cdr(R));
else if car(R) = UNION then
    return MYT-union(cdr(R));
else // Malformed Regex
    ERROR;
```

Procedure MYT-base

Input: R

Output: (Q, E, s, a)

```
s ← newstate();
a ← newstate();
Q ← {s, a};
if R = ∅ then // empty set ∅
    E ← ∅;
else // empty string ε or symbol σ
    E ← {s -R-> a};
```

$$E \leftarrow \left\{ s \xrightarrow{R} a \right\};$$

Procedure MYT-concatenate

Input: c

Output: (Q, E, s, a)

```
function f(M, R) is
    concatenate(M, MYT(R));
if ∅ = c then
    return MYT-base(ε);
else
    return fold-left(f, MYT(car(c)), cdr(c));
```

Procedure MYT-union

Input: c

Output: (Q, E, s, a)

```
function f(M, R) is
    union(M, MYT(R));
if ∅ = c then
    return MYT-emptyset(∅);
else
    return fold-left(f, MYT(car(c)), cdr(c));
```

NFA to Regular Expressions

Input: NFA N

Output: Equivalent regular expression R,

$L(\mathrm{N}) = L(\mathrm{R})$

Approach: Construction generalized NFA: an NFA with regular expressions on its edges 1. Convert N to an initial GNFA 2. Iteratively remove (rip) states from the GNFA 3. When the GNFA has only two states (start and accept), the edge between them is the equivalent regular expression

Generalized NFA (GNFA)

A generalized NFA $\tilde{N}$ = (Q, Σ, δ, q_{start}, q_{accept}):

- Q is the finite set of states
- Σ is the input alphabet
- δ: (Q {q~accept}) × (Q {q_{start}}) → REGEX
- q_{start} ∈ Q is the start state
- q_{accept} ∈ Q is the accept state

Ripping a state from the GNFA

Ripping is taking a section of the GNFA and converting it into a regex

- Given: $state_i$ → $state_{rip}$ → $state_j$
- Convert: $state_i$ → $state_j$ where the transition is the regex

Ripping state $\tilde{q}$

- Predecessor state q_i
- Successor state q_j

Four types of edges:

- Predecessor E(q_i, $\tilde{q}$)
- Successor E($\tilde{q}$, q_j)
- Loop E($\tilde{q}$, $\tilde{q}$)
- Bypass E(q_i, q_j)

All predecessor/successor pairs

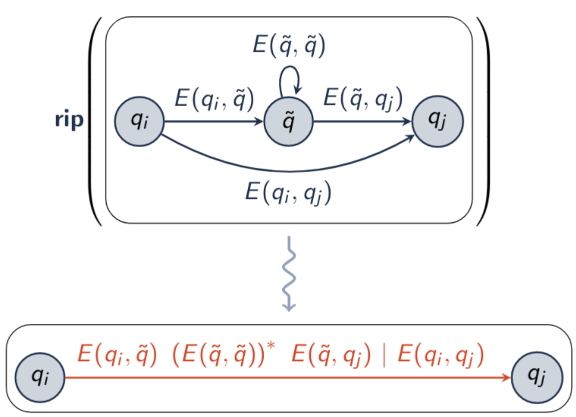


Figure 5: Example rip of a GNFA

NFA to Regular Expression Algorithm

Algorithm 2: NFA to Regex

Input: $N = (Q, \Sigma, E, q_0, F)$; // states, alphabet, edges, start, accept

Output: R; // Regular Expression

/* Construct the initial GNFA */

$N' \leftarrow \text{NFA-to-GNFA}(N)$;

/* Call Convert() subroutine on the GNFA */

$R \leftarrow \text{Convert}(N')$;

Algorithm 3: NFA-to-GNFA

Input: $N = (Q, \Sigma, E, q_0, F)$; // states, alphabet, edges, start, accept

Output: $N' = (Q', \Sigma, E', q_{\text{start}}, q_{\text{accept}})$; // states, alphabet, edges, start, accept

$Q' \leftarrow Q \cup \{q_{\text{start}}, q_{\text{accept}}\}$; // add new start, accept states

$$E' \leftarrow \underbrace{\left\{q_{\text{start}} \xrightarrow{\varepsilon} q_0\right\}}_{\text{edge from new start}} \cup \underbrace{\bigcup_{q \in F} \left\{q \xrightarrow{\varepsilon} q_{\text{accept}}\right\}}_{\text{edges to new accept}};$$

forall $q_i \in Q$ **do** // Merge multiple edges between nodes into union edges

    forall $q_j \in Q$ **do**

        $e \leftarrow \left\{a \xrightarrow{\sigma} b \in E \mid a = q_i \wedge b = q_j\right\}$; // set of edges from q_i to q_j

        if $|e| \leq 1$ **then**

            $E' \leftarrow E' \cup e$;

        else if $|e| > 1$ **then**

            $\ell \leftarrow \left(\bigcup_{a \xrightarrow{\sigma} b \in e} \{\sigma\}\right)$; // set of edge labels from q_i to q_j

            $r \leftarrow \textbf{regex}\,(\ell_0 \cup \ldots \cup \ell_n)$;

            $E' \leftarrow E' \cup \left\{q_i \xrightarrow{r} q_j\right\}$;

Function Convert(Q,E)

if $|Q| = 2$ **then**

  $R \leftarrow E(q_{\text{start}}, q_{\text{accept}})$; // Extract label of edge from GNFA start to accept

  return R;

else

  $\tilde{q} \leftarrow$ any state in $(Q \setminus \{q_{\text{start}}, q_{\text{accept}}\})$;

  $Q' \leftarrow Q \setminus \{\tilde{q}\}$;

  $E' \leftarrow E \setminus \{\tilde{q} \rightarrow \tilde{q}\}$;

  forall q_i **where** $E(q_i, \tilde{q}) \neq \emptyset$ **do** // predecessors of $\tilde{q}$

    forall q_j **where** $E(\tilde{q}, q_j) \neq \emptyset$ **do** // successors of $\tilde{q}$

      $r \leftarrow$ **regex** $(E(q_i, \tilde{q})\ (E(\tilde{q}, \tilde{q}))^*\ E(\tilde{q}, q_j)\ \cup\ E(q_i, q_j))$;

      $E' \leftarrow E' \setminus \{(q_i \rightarrow \tilde{q}), (\tilde{q} \rightarrow q_j), (q_i \rightarrow q_j), \}$;

      $E' \leftarrow E' \cup \left\{q_i \xrightarrow{r} q_j\right\}$;

  return Convert(Q', E');

Equivalence