ε-NFAs and Regular Expressions

COMPSCI 3331

Outline

- ε-NFAs
- Transforming ε-NFAs to (standard) NFAs.
- Regular expressions: definitions

ε-NFAs: Motivation

- We want to extend NFAs to allow transitions without "consuming any input".
- Called ε-**NFAs**.
- New transitions are ε-transitions.
- Why have ε-transitions?
  - can make the definition of a machine easier to understand.
  - ε-NFAs will be a crucial "intermediate step"

ε-NFA: Example

- Temperatures: "Maybe we see a '-' sign"
- Format: '+'/'-' sign (maybe), then number, then "℃".

ε-NFAs

An ε-NFA is a 5-tuple $M = (Q, \Sigma, \delta, q_0, F$ where

- Q is a finite set of states,
- Σ is a finite alphabet,
- $\delta : Q \times (\Sigma \cup \{\varepsilon\}) \rightarrow 2^Q$ is the transition function.
- $q_0 \in Q$ is the start state.
- $F \subseteq Q$ is the set of final states.

ε-NFAs

The only difference between NFAs and ε-NFAs is δ:

$$\delta : Q \times (\Sigma \cup \{\varepsilon\}) \to 2^Q.$$

- $\delta(q, \varepsilon)$ defines the set of states that the ε-NFA can go to on empty input ε.

We want to be able to translate an ε-NFA into an NFA without ε-transitions.

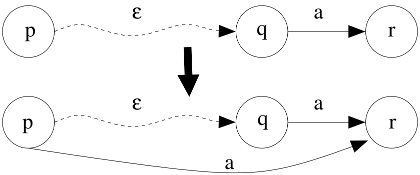
Removing ε-transitions

- An ε-path from q_1 to q_2 is a sequence of ε-transitions from q_1 to q_2.
- How do we replace ε-paths?

Removing ε-transitions

Algorithm A: remove ε-transitions from $M = (Q, \Sigma, \delta, q_0, F)$

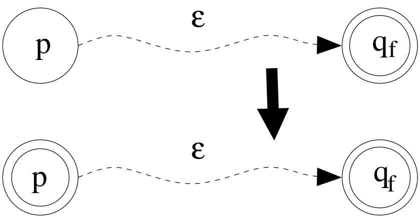
for all $q \in Q$ **do**
  for all $p \in Q$ **do**
    if there is an ε-path from p to q **then**
      for all $a \in \Sigma$ and all $r \in \delta(q, a)$ **do**
        add the state r to the set $\delta(p, a)$
      end for
    end if
  end for
end for



Problem with Algorithm A: Final states

Before removing ε-transitions, update final states:

$F' = \emptyset$
for all $q_f \in F$ **do**
  for all $p \in Q$ **do**
    if there is an ε-path from p to q_f **then**
      add the state p to F'
    end if
  end for
end for
$F = F \cup F'$



Removing ε-transitions (1 of 2)

Let $M = (Q, \Sigma, \delta, q_0, F)$ be a ε-NFA. Let $cl_\varepsilon : Q \to 2^Q$ be defined by

$$\begin{aligned} cl_\varepsilon(q) &= \{r \in Q \ : \ \exists n \geq 0, q_0, q_1, \ldots, q_n \in Q \text{ such that} \\ &\qquad q_0 = q, q_n = r, \text{ and } q_{i+1} \in \delta(q_i, \varepsilon) \ \forall 0 \leq i \leq n-1\}. \end{aligned}$$

Let $M' = (Q, \Sigma, \delta', q_0, F')$ be an NFA without ε-transitions defined by

$$\delta'(q,a) = \delta(q,a) \cup \bigcup_{q' \in cl_\varepsilon(q)} \delta(q', a)$$

and

$$F' = F \cup \{q \ : \ cl_\varepsilon(q) \cap F \neq \emptyset\}.$$

Removing ε-transitions (2 of 2)

Theorem. Let $M = (Q, \Sigma, \delta, q_0, F)$ be an ε-NFA and $M' = (Q, \Sigma, \delta', q_0, F')$ be the NFA without ε-transitions defined using cl_ε. Then $L(M) = L(M')$.

From last slide ...

$$\delta'(q,a) = \delta(q,a) \cup \bigcup_{q' \in cl_\varepsilon(q)} \delta(q',a)$$

and

$$F' = F \cup \{q \ : \ cl_\varepsilon(q) \cap F \neq \emptyset\}.$$

Regular Expressions

- Regular expressions are a **textual** representation of languages.
- Regular expressions define exactly the regular languages, i.e., they are equivalent to DFAs and NFAs.

Regular Expressions: Definition

Let Σ be an alphabet. Regular languages are defined recursively.

- $\emptyset$ is a regular expression, ε is a regular expression and a is a regular expression for all $a \in \Sigma$.
- if r_1, r_2 are regular expressions, then so are
  (a) $r_1 r_2$ (i.e., the concatenation of r_1 and r_2);
  (b) $r_1 + r_2$;
  (c) r_1^*.

Language defined by a Regular Expression

Each regular expression r defines a language, denoted by $L(r)$.

- $L(\emptyset) = \emptyset$, $L(\varepsilon) = \varepsilon$; and $L(a) = \{a\}$ for all $a \in \Sigma$.
- if r_1, r_2 are regular expressions, then
  - (a) $L(r_1 r_2) = L(r_1)L(r_2)$;
  - (b) $L(r_1 + r_2) = L(r_1) \cup L(r_2)$;
  - (c) $L(r_1^*) = (L(r_1))^*$.

r^*

$$L^* = \{\varepsilon\} + L + L^2 + L^3 + L^4 + \cdots$$

So $\varepsilon \in L^*$ **for all languages** L.

- $\{\varepsilon\}^* = \{\varepsilon\}$.
- $\emptyset^* = \{\varepsilon\}$.
- $a^* = \varepsilon + a + a^2 + a^3 + \cdots = \{a^i \ : \ i \geq 0\}$.
- $(a^* b^*)^* = (a + b)^*$
- a regular expression identity : for all r_1, r_2,

$$(r_1^* r_2^*)^* = (r_1 + r_2)^*$$

Applications of Regular Expressions

- Regular expressions will be familiar to you because of their applications:
  - regular expressions are built into software, especially under Linux: `grep`, `vi` (COMPSCI 2211)
  - Built into many languages like python.
  - Used in lexical analysis in compiler design.
- However, regexp in most applications has some additional power: `[^a]`, bracketing (`(\1)`), etc.

Next Time: regular language representations

- transform a regular expression into an automaton:

  regular expression $\rightarrow \varepsilon$-NFA.
- transform a DFA into a regular expression.
- Therefore, regular expressions define exactly the regular languages.