Lecture 4 – Regular expressions, Kleene's theorem, string substitution

NTIN071 Automata and Grammars

Jakub Bulín (KTIML MFF UK)

Spring 2024

** Adapted from the Czech-lecture slides by Marta Vomlelová with gratitude. The translation, some modifications, and all errors are mine.*

Recap of Lecture 3

- Nondeterministic finite automata (NFA): can ‘guess’ the right path to accepting, computation described by a state tree.
- ϵ-transitions: allow to change states without reading any input
- Subset construction: every NFA and ϵ-NFA is equivalent to a DFA (but can be easier to design, much smaller).
- Regular languages are closed under set operations (union, intersection, complement, difference)
- And under string operations (concatenation, iteration and positive iteration, reverse, left and right quotient)

1.8 Regular expressions

Regular expressions (RE)

- an algebraic description of languages
- declarative: express the form of the words we want to accept
- can describe all, and only, regular languages
- can be viewed as a programming language, a user/friendly description of a finite automaton

Example

- `grep` command in UNIX.
- Python module `re`
- lexical analysis, e.g. `Flex` (description via 'tokens' $\leftrightsquigarrow$ RE)

Note: syntax analysis needs a stronger tool, context-free grammars

The definition

A regular expression α over (finite, nonempty) Σ, $\alpha \in \mathsf{RegE}(\Sigma)$ and the matching language $L(\alpha)$, are defined inductively:

expression	**language**	**note**
$\emptyset$	$L(\emptyset) = \emptyset$	empty expression
ϵ	$L(\epsilon) = \{\epsilon\}$	empty string
$\mathbf{a}$	$L(\mathbf{a}) = \{a\}$	for all $a \in \Sigma$
$(\alpha + \beta)$	$L((\alpha + \beta)) = L(\alpha) \cup L(\beta)$	union (`grep`, `re` use '|')
$(\alpha\beta)$	$L((\alpha\beta)) = L(\alpha)L(\beta)$	concatenation
α^*	$L(\alpha^*) = L(\alpha)^*$	iteration (Kleene star)

Examples, notation

Example

- The language of alternating 0s and 1s can be expressed as:
  - $(\mathbf{01})^* + (\mathbf{10})^* + \mathbf{1}(\mathbf{01})^* + \mathbf{0}(\mathbf{10})^*$
  - $(\epsilon + \mathbf{1})(\mathbf{01})^*(\epsilon + \mathbf{0})$
- $L((\mathbf{0}^*\mathbf{10}^*\mathbf{10}^*\mathbf{1})^*\mathbf{0}^*) = \{w \in \{0,1\}^* \mid |w|_1 \equiv 0 \pmod 3\}$

We often omit parentheses:

- priority of operators: iteration $*$ $>$ concatenation $>$ union $+$
- associativity of concatenation, union $+$
- outer parentheses

We could define, and will sometimes use, positive iteration α^+

Kleene's theorem

Theorem (Kleene's theorem)

A language is regular, iff it is matched by some regular expression.

We will prove it by giving two constructions:

1. from RE to ϵ-NFA (which can be converted to a DFA)
2. from a DFA to a RE (but we could start from a ϵ-NFA)

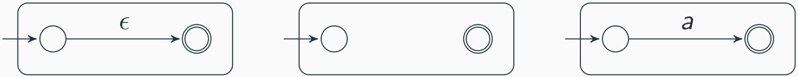
For 2. we also mention a better algorithm: state eliminiation

RE to ϵ-NFA

By induction on the structure of α, construct a ϵ-NFA E s.t. $L(\alpha) = L(E)$ with three additional properties:

1. Exactly one accepting state.
2. No incoming edges into the initial state.
3. No outgoing edges from the accepting state.

Induction base: α is the empty string ϵ, empty set $\emptyset$, or a letter **a**

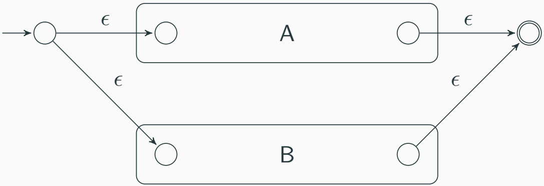


Induction step: $\alpha + \beta$, $\alpha\beta$, α^* (next slide) $\square$

RE to ϵ-NFA: Induction step

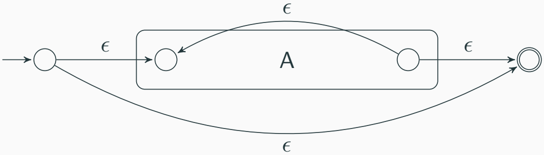
Let A, B be ϵ-NFA constructed for α, β.

Addition $\alpha + \beta$



Concatenation $\alpha\beta$

Iteration α^*



DFA to RE

Assume the states are $Q = \{1, \ldots, n\}$ and the start state is $q_0 = 1$.

Construct a RE $R_{ij}^{(k)}$ matching words that transition from state i into state j and all intermediate states (if any) have index $\leq k$.

Then we set $\alpha = \sum_{j \in F_A} R_{1j}^{(n)}$ (from start to some accepting state)

Iteratively construct $R_{ij}^{(k)}$ for $k = 0, \ldots, n$ (finite induction).

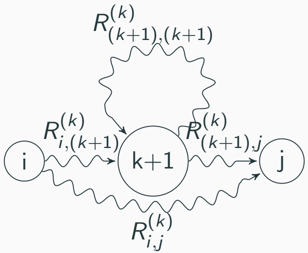
Induction base: $k = 0$

- If $i \neq j$, set $R_{ij}^{(0)} = \mathbf{a_1} + \ldots + \mathbf{a_m}$ where $a_1, \ldots, a_m$ are symbols on edges from i into j ($R_{ij}^{(0)} = \emptyset$ or $R_{ij}^{(0)} = \mathbf{a}$ for $m = 0, 1$).
- If $i = j$, $R_{ii}^{(0)} = \epsilon + \mathbf{a_1} + \ldots + \mathbf{a_m}$ where a_i's are on loops on i.

DFA to RE: Induction step

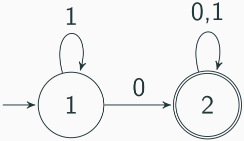
Once we have $R_{ij}^{(k)}$ for all $i, j \in Q$, we can construct $R_{ij}^{(k+1)}$:

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



- paths $i \rightsquigarrow j$ not going through $k+1$: already in $R_{ij}^{(k)}$
- paths $i \rightsquigarrow j$ going through $k+1$ one or more times: $i \rightsquigarrow k+1$ (first visit), loop on $k+1$, finally (last visit) $k+1 \rightsquigarrow j$ $\square$

Example



Apply the construction, simplify:

$$\alpha = R_{12}^{(2)} = \mathbf{1}^*\mathbf{0}(\mathbf{0}+\mathbf{1})^*$$

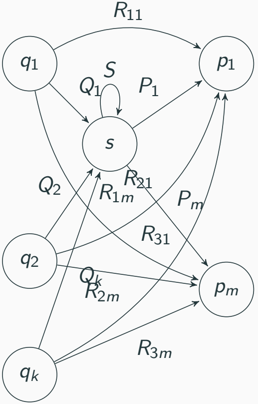
$R_{11}^{(0)}$	$\epsilon + \mathbf{1}$	$=$
$R_{12}^{(0)}$	$\mathbf{0}$	$=$
$R_{21}^{(0)}$	$\emptyset$	$=$
$R_{22}^{(0)}$	$(\epsilon + \mathbf{0} + \mathbf{1})$	$=$
$R_{11}^{(1)}$	$\epsilon + \mathbf{1} + (\epsilon + \mathbf{1})(\epsilon + \mathbf{1})^*(\epsilon + \mathbf{1})$	$=\mathbf{1}^*$
$R_{12}^{(1)}$	$\mathbf{0} + (\epsilon + \mathbf{1})(\epsilon + \mathbf{1})^*\mathbf{0}$	$=\mathbf{1}^*\mathbf{0}$
$R_{21}^{(1)}$	$\emptyset + \emptyset(\epsilon + \mathbf{1})^*(\epsilon + \mathbf{1})$	$=\emptyset$
$R_{22}^{(1)}$	$\epsilon + \mathbf{0} + \mathbf{1} + \emptyset(\epsilon + \mathbf{1})^*\mathbf{0}$	$=\epsilon + \mathbf{0} + \mathbf{1}$
$R_{11}^{(2)}$	$\mathbf{1}^* + \mathbf{1}^*\mathbf{0}(\epsilon + \mathbf{0} + \mathbf{1})^*\emptyset$	$=\mathbf{1}^*$
$R_{12}^{(2)}$	$\mathbf{1}^*\mathbf{0} + \mathbf{1}^*\mathbf{0}(\epsilon + \mathbf{0} + \mathbf{1})^*(\epsilon + \mathbf{0} + \mathbf{1})$	$=\mathbf{1}^*\mathbf{0}(\mathbf{0}+\mathbf{1})^*$
$R_{21}^{(2)}$	$\emptyset + (\epsilon + \mathbf{0} + \mathbf{1})(\epsilon + \mathbf{0} + \mathbf{1})^*\emptyset$	$=\emptyset$
$R_{22}^{(2)}$	$\epsilon + \mathbf{0} + \mathbf{1} + (\epsilon + \mathbf{0} + \mathbf{1})(\epsilon + \mathbf{0} + \mathbf{1})^*(\epsilon + \mathbf{0} + \mathbf{1})$	$=(\mathbf{0}+\mathbf{1})^*$

State elimination algorithm

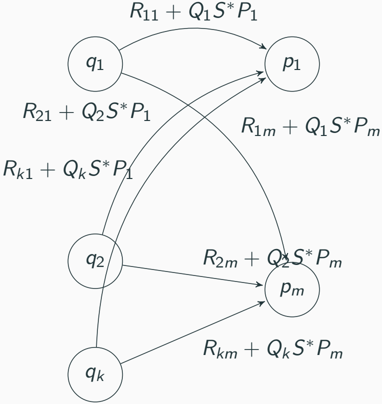
State elimination: the idea

Idea: Allow edges labelled by RE, iteratively remove nodes. (More efficient, avoids duplicity.)

State s selected for elimination



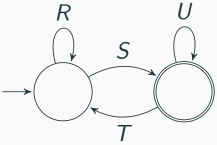
After s is eliminated.



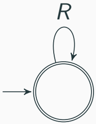
State elimination: the algorithm

For every accepting $q \in F$ eliminate all states $p \in Q \setminus \{q, q_0\}$.

- for $q \neq q_0$: $\mathrm{RegE}(q) = (R + SU^*T)^*SU^*$



- for $q = q_0$: $\mathrm{RegE}(q) = R^*$

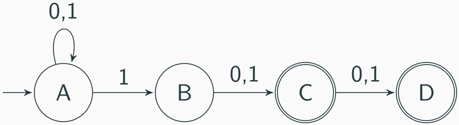


Finally, union over all accepting states: $\mathrm{RegE}(A) = \sum_{q \in F} \mathrm{RegE}(q)$

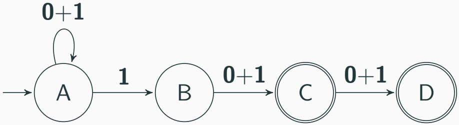
(Elimination order: first nonaccepting and noninitial states.)

State elimination: an example

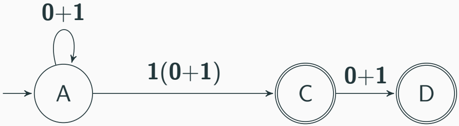
The original automaton:



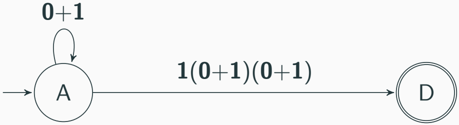
Replace letters by RE:



Eliminate B:



Eliminate C:



$$(\mathbf{0}+\mathbf{1})^*\mathbf{1}(\mathbf{0}+\mathbf{1}) + (\mathbf{0}+\mathbf{1})^*\mathbf{1}(\mathbf{0}+\mathbf{1})(\mathbf{0}+\mathbf{1})$$

Algebraic description of regular languages

Let $\mathsf{RL}(\Sigma)$ denote the smallest set of languages over Σ that:

- contains $\emptyset$ and $\{x\}$ for any letter $x \in \Sigma$, and
- is closed under union, concatenation, and iteration.

That is, for $A, B \in \mathsf{RL}(\Sigma)$ also $A \cup B, A.B, A^* \in \mathsf{RL}(\Sigma)$. Note that:

- $\{\epsilon\} \in \mathsf{RL}(\Sigma)$ since $\{\epsilon\} = \emptyset^*$
- $\Sigma \in \mathsf{RL}(\Sigma)$ since $\Sigma = \bigcup_{x \in \Sigma} \{x\}$ (a finite union)
- $\Sigma^* \in \mathsf{RL}(\Sigma)$
- any finite language over Σ is in $\mathsf{RL}(\Sigma)$.

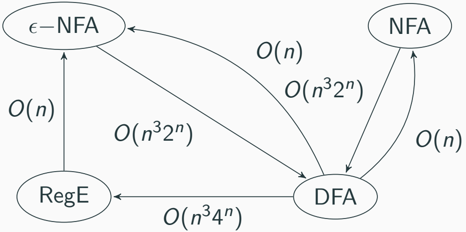
Theorem (A restatement of Kleene's Theorem)

A language over Σ is regular, iff it is in $\mathsf{RL}(\Sigma)$.

Some properties to simplify RE (will not be tested)

$$\begin{aligned}
L.\emptyset = \emptyset.L &= \emptyset \\
\{\epsilon\}.L = L.\{\epsilon\} &= L \\
(L^*)^* &= L^* \\
(L_1 \cup L_2)^* &= L_1^*(L_2.L_1^*)^* = L_2^*(L_1.L_2^*)^* \\
(L_1.L_2)^R &= L_2^R.L_1^R \\
\partial_w(L_1 \cup L_2) &= \partial_w(L_1) \cup \partial_w(L_2) \\
\partial_w(\Sigma^* - L) &= \Sigma^* - \partial_w L.
\end{aligned}$$

Converting between representations



- NFA or ϵ-NFA to DFA: $O(n^3 2^n)$
  - ϵ-closure in $O(n^3)$ (search n states $\times$ n^2 arcs)
  - subset construction, DFA with up to 2^n states; for each state need $O(n^3)$ time to compute transitions.
- DFA to NFA or ϵ-NFA: $O(n)$
  - a simple modification of the transition table
- DFA to RE: $O(4^n)$
- RE to ϵ-NFA: $O(n)$

String substitution

String substitution and homomorphism

A (string) substitution is a mapping $\sigma\colon \Sigma^* \to \mathcal{P}(Y^*)$ where

- Σ and Y are finite alphabets, $Y = \bigcup_{x\in\Sigma} Y_x$
- for each $x \in \Sigma$, $\sigma(x)$ is a language over Y_x
- $\sigma(\epsilon) = \{\epsilon\}$ and $\sigma(u.v) = \sigma(u).\sigma(v)$

For a language $L \subseteq \Sigma^*$, $\sigma(L) = \bigcup_{w\in L} \sigma(w) \subseteq Y^*$. A substitution is ϵ-free if no $\sigma(x)$ contains ϵ.

A (string) homomorphism is defined similarly but each letter is mapped to a single word, $h\colon \Sigma^* \to Y^*$ where $h(x) \in Y_x^*$ for $x \in \Sigma$, $h(\epsilon) = \epsilon$ and $h(u.v) = h(u).h(v)$. Then $h(L) = \{h(w) \mid w \in L\}$. It is ϵ-free if $h(x) \neq \epsilon$ for all $x \in \Sigma$.

The inverse homomorphism applied to a language $L' \subseteq Y^*$:

$$h^{-1}(L') = \{w \in \Sigma^* \mid h(w) \in L'\}$$

Examples

Example (Substitution)

- If $\sigma(0) = \{a^i b^j, i, j \geq 0\}$ and $\sigma(1) = \{cd\}$, then $\sigma(010) = \{a^i b^j cda^k b^l \mid i, j, k, l \geq 0\}$.
- $\Sigma = \{f, l, s, c, d\}$, $L = L((fsl)(cfsl)^* d)$ where
  - $\sigma(f)$ is a dictionary of first names
  - $\sigma(l)$ are last names
  - $\sigma(s) = \{\text{' '}\}$ (space), $\sigma(c) = \{\text{','}\}$, $\sigma(d) = \{\text{'.'}\}$
- A document template with symbols to be replaced by fields of database entries.

Example (Homomorphism)

- Define $h(0) = ab$ and $h(1) = \epsilon$. Then $h(0011) = abab$ and for $L = \mathbf{10^*1}$ we have $h(L) = L((ab)^*)$.
- Replace special symbols with TeX code (e.g. $h(\mu) = \backslash mu$).

Preserving regularity

Theorem

Let $L \subseteq \Sigma^*$ be regular, $h\colon \Sigma^* \to Y^*$ a homomorphism, and $\sigma\colon \Sigma^* \to \mathcal{P}(Y^*)$ a substitution.

- *The language $h(L)$ is regular.*
- *If $\sigma(x)$ is regular for all $x \in \Sigma$, then $\sigma(L)$ is also regular.*

Moreover, if $L' \subseteq Y^*$ is regular, then $h^{-1}(L')$ is also regular.

Proof for homomorphism and substitution

Homomorphism $\leftrightsquigarrow$ substitution with $\sigma(x)$ one-element (regular).

Structural induction on a RE α such that $L = L(\alpha)$.

- **Induction base:** $\emptyset$, ϵ, $\mathbf{a}$ ... easy
- **Induction step:**

$$\begin{aligned} \sigma(L(\alpha+\beta)) &= \sigma(L(\alpha)) \cup \sigma(L(\beta)) \\ \sigma(L(\alpha\beta)) &= \sigma(L(\alpha)).\sigma(L(\beta)) \end{aligned}$$

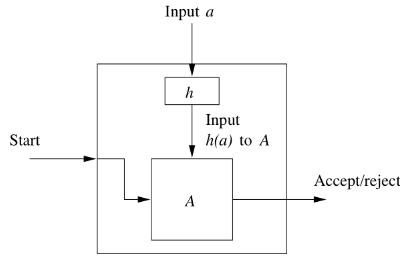
For iteration, decompose into an infinite union of powers:

$$\begin{aligned} \sigma(L(\alpha)^*) &= \sigma(L(\alpha)^0) \cup \sigma(L(\alpha)^1) \cup \dots \\ &= \sigma(L(\alpha))^0 \cup \sigma(L(\alpha))^1 \cup \dots = \sigma(L(\alpha))^* \end{aligned}$$

(Alternative view: take the tree of the RE α and replace every leaf x with a tree for a RE for $\sigma(x)$.) $\square$

Proof for inverse homomorphism & an example

Given a DFA $A = (Q, Y, \delta, q_0, F)$ recognizing L', construct a DFA recognizing $h^{-1}(L')$: $B(Q, \Sigma, \delta_B, q_0, F)$ with $\delta_B(q, a) = \delta^*(q, h(a))$ That is, for a letter $a \in \Sigma$ do what A does for the word $h(a)$. Easy to show (by induction on $|w|$) that $\delta_B^*(q_0, w) = \delta^*(q_0, h(w))$.



$\square$

Example (Inverse homomorphism)

$L' = L((\mathbf{00} + \mathbf{1})^*)$, $h(a) = 01$, and $h(b) = 10$: $h^{-1}(L') = (\mathbf{ba})^*$.
$[h(L((\mathbf{ba})^*)) \in \mathbf{L}'$ is obvious, other words generate an isolated 0.]

Decision properties of regular languages

Testing emptiness

Given a representation of a regular language L, is $L = \emptyset$?

FA: is any final state reachable from the initial state? $O(n^2)$

RE: convert to $\epsilon-$NFA (in $O(n)$ time) and check reachability

or directly:

basis: $\emptyset$ is empty, ϵ and **a** are not

induction:

- $\alpha = (\alpha_1 + \alpha_2)$: empty iff both $L(\alpha_1)$ and $L(\alpha_2)$ are empty
- $\alpha = (\alpha_1 \alpha_2)$ empty iff either $L(\alpha_1)$ or $L(\alpha_2)$ is empty
- $\alpha = (\alpha_1^*)$ never empty, includes ϵ

Testing membership

Given a regular language L and a word w, is $w \in L$?

- DFA: run the automaton; if $|w| = n$, with a suitable representation (constant time transitions) it is in $O(n)$
- NFA with s states: running time $O(ns^2)$, each letter processed by taking the previous set of states
- ϵ-NFA: first compute the ϵ-closure, then for each letter, process it and compute the ϵ-closure of the result
- RE of size s: convert to an ϵ-NFA with at most $2s$ states and then simulate, $O(ns^2)$

Summary of finite automata

- Finite Automata: DFA, reduced DFA, NFA, ϵ-NFA
- Regular Expressions
- Regular languages: closed under set operations, string operations, substitution, homomorphism, inverse hom.
- all FA and RE describe the same class of languages
- Key theorems
  - Mihyll–Nerode (implicit DFA via congruences on words)
  - Kleene (regular languages iff matched by RE)
  - Pumping lemma
- (optional) 2-way automata
- (optional) Automata with output
  - Moore machine
  - Mealy machine.

Summary of Lecture 4

- regular expressions
- Kleene's theorem (two variants)
- constructions: RE to ϵ-NFA, DFA to RE
- state elimination algorithm
- string substitution, homomorphism, inverse homomorphism
- decision properties

Appendix: Visit every state

Visit every state

Example (visit every state)

Given a DFA A, let L consist of all $w \in \Sigma^*$ that are accepted and, moreover, during the computation every state is visited, i.e.:

- $\delta^*(q_0, w) \in F$
- for every $q \in Q$ there is a prefix x_q of w s.t. $\delta^*(q_0, x_q) = q$

We will show that this language is regular.

Construct L from $M = L(A)$ using operations preserving regularity:

- define an alphabet of 'transitions': $T = \{[paq] \mid \delta(p, a) = q\}$
- define a homomorphism $h([paq]) = a$ for all p, q, a
- $L_1 = h^{-1}(M)$ is regular (inverse homomorphism), consists of accepting sequences of transitions

Visit every state: proof continues

- start at q_0: $L_2 = L_1 \cap L(E_1.T^*)$, $E_1 = \{[q_0aq] \mid a \in \Sigma, q \in Q\}$
- adjacent states must be equal: define non-matching pairs

  $$E_2 = \{[paq][rbs] \mid q \neq r, p, q, r, s \in Q, a, b \in \Sigma\}$$

  and set $L_3 = L_2 - L(T^*.E_2.T^*)$ (remove if at least one non-matching pair of adjacent states)
- L_3 already ends in accepting state: we started from $M = L(A)$
- all states: for $q \in Q$ let E_q be the RE that is the sum of all the symbols in T not containing q, set $L_4 = L_3 - \bigcup_{q \in Q}\{L(E_q^*)\}$
- from a sequence of transitions back to the word: $L = h(L_4)$

$\square$