Regular Expressions

We’ve seen regular languages are closed under
▶ Union
▶ Concatenation
▶ Kleene Star

Starting from automata for the languages, we’ve constructed automata that correspond to these operations.

We can also use notation that directly describes and performs operations on languages.

Definition of Regular Expressions

$R$ is a regular expression if it is

▶ $a$, for some symbol $a \in \Sigma$ represents $\{a\}$
▶ $\epsilon$ represents $\{\epsilon\}$
▶ $\emptyset$
▶ $(R_1 \cup R_2)$ where $R_1$ and $R_2$ are regular expressions sometimes denoted $R_1 + R_2$ or $R_1 \mid R_2$
▶ $(R_1 \circ R_2)$ where $R_1$ and $R_2$ are regular expressions sometimes written simply $R_1R_2$
▶ $(R_1^*)$ where $R_1$ is a regular expression

inductive definition (by structural induction)

Some definitions have only two base cases; $\epsilon = \emptyset^*$ is derived
More shorthands: $R^+ = RR^*$

Examples of Regular Expressions

Consider alphabet $\Sigma = \{0, 1\}$. Write $\Sigma$ for $0 \cup 1$.

Single 1: $0^*1^*$
At least a 1: $\Sigma^*1\Sigma^*$
Length $\equiv 1 \mod 3$: $\Sigma(\Sigma\Sigma\Sigma)^*$
Every 0 followed by a 1: $1^*(01)^* \cup (101)^*$
Contains no 00: like above, but can end in 0: $(101)^*(0 + \epsilon)$
$(0^*1)^*$: does not end in 0
Real numbers with optional sign: $(+|−|\epsilon)(D^*D|D^+D^*|
(must include decimal point, otherwise int)

Used in lexical analysis (compiler)
Recognize email addresses, URLs, etc.

Regular Expression Identities

There are multiple regular expressions describing a given language.

Basic Identities:
$R \cup \emptyset = R$
$R\epsilon = \epsilon R = R$

Others:
$(R^*)^* = R^*$
$\epsilon \cup RR^* = R^*$

etc.

Equivalence with Finite Automata

A language is regular if and only if a regular expression describes it.
(Kleene’s Theorem)

Proof: by construction
(1) Construct automaton (NFA) from regular expression
(2) Construct regular expression for automaton

For (1), we start with automata for the three base cases

We can apply the construction discussed for NFAs.
Closure under Union

Add new initial state with $\varepsilon$-transitions to both initial states

![Figure 1.46: Construction of an NFA $N$ to recognize $A_1 \cup A_2$](image)

Closure under Kleene Star

Add $\varepsilon$-transitions from all accept states to initial state, and new initial (and accepting) state with $\varepsilon$-transitions to original one.

![Figure 1.50: Construction of $N$ to recognize $A^*$](image)

Alternative Constructions (cont’d.)

Union $R_1 \cup R_2$: merge initial and final states

Kleene Star: merge initial and accept state, create new initial and accept state with $\varepsilon$-transitions to loop

Correctness proof: invariants are maintained

Converting Automata to Regular Expressions

Generalized Nondeterministic Finite Automaton (GNFA)

- transitions labeled with regular expressions
- single initial state, no incoming transitions
- single accept state (not initial), no outgoing transitions
- textbook: transitions between all other states (labeled $\emptyset$ if needed)
CHAPTER 1 / REGULAR LANGUAGES

Taking the machine from $q_i$ to $q_j$ either directly or via $q_{rip}$. We illustrate this process.

### Derivative Rules

- $\partial_a \emptyset = \emptyset$
- $\partial_a \varepsilon = \emptyset$
- $\partial_a a = \varepsilon$
- $\partial_a b = \emptyset$
- $\partial_a (R + S) = \partial_a R + \partial_a S$
- $\partial_a (RS) = (\partial_a R)S - \partial_a S \cdot R$ if $\varepsilon \notin R$
- $\partial_a (RS) = (\partial_a R)S + \partial_a S \cdot R$ if $\varepsilon \in R$
- $\partial_a R^* = \partial_a R \cdot R^*$

### Determining Membership

Given a regular expression $R$ and a string $u$, find whether $u \in R$.

Construct automaton for $R$ (“compile” regex), run on $u$.

What is the complexity? worst-case exponential in size of $R$

DFA may be exponential in size of NFA or, if we backtrack in NFA, can spend exponential time.

ReDoS (denial of service attack): if $R$ controlled by user:

- nested repetitions (applied to complex expression)
- a match is also a suffix of another match, e.g. $a...ax \in (a^+)^+$?

$\Rightarrow$ check libraries for evil regexes do lazy construction of DFA if regex user-supplied

### Example: Strings with no aba

Introduce new accepting state 4

Eliminate 2, then 3

$R = (b(a^+bb)^*(a^+)a^+b)$

### Derivative of Regular Expression

(Brzoziowski, 1964)

Q: Given a language $L$ and a symbol $a$, what are the suffixes of strings in $L$ that start with $a$?

$\partial_a L = \{v \mid av \in L\}$

Example: $\partial_a a^+b = a^+b$, $\partial_a a^+b = \varepsilon$

How could we use the derivative?

- check string membership: take derivative for successive symbols $a_1, a_2, \ldots, a_n$, check if result accepts $\varepsilon$
- directly construct DFA from regular expression!

### Constructing DFA with Derivatives

Every distinct regular expression obtained is a state.

A state is accepting if regex contains $\varepsilon$.

Construct DFA for $(a^+b)^*$

$\partial_a (a^+b)^* = \partial_a (a^+b) \cdot (a^+b)^* = a^b(a^+b)^*$

$\partial_b (a^+b)^* = \partial_b (a^+b) \cdot (a^+b)^* = \varepsilon(a^+b)^* = (a^+b)^*$

$\partial_a (a^b(a^+b)^*) = \partial_a (a^b(a^+b)^*) = a^b(a^+b)^*$

$\partial_b (a^b(a^+b)^*) = \partial_b (a^b(a^+b)^*) = (a^+b)^*$

### Derivating States

Successively eliminate all states except initial and final.

When eliminating state $q_{rip}$, consider all paths $q_i \rightarrow q_{rip} \rightarrow q_j$.

Augment transition $q_i \rightarrow q_j$ with regular expression $R_1R_2R_3$.

Finally, only initial and accept states with overall regex left.

Correctness proof: by induction over number of states.

### Example: GNFA with one fewer state

Introduce new accepting state 4

Eliminate 2, then 3

$R = (b(a^+bb)^*(a^+)a^+b)$

### Constructing DFA with Derivatives

Every distinct regular expression obtained is a state.

A state is accepting if regex contains $\varepsilon$.

Construct DFA for $(a^+b)^*$

$\partial_a (a^+b)^* = \partial_a (a^+b) \cdot (a^+b)^* = a^b(a^+b)^*$

$\partial_b (a^+b)^* = \partial_b (a^+b) \cdot (a^+b)^* = \varepsilon(a^+b)^* = (a^+b)^*$

$\partial_a (a^b(a^+b)^*) = \partial_a (a^b(a^+b)^*) = a^b(a^+b)^*$

$\partial_b (a^b(a^+b)^*) = \partial_b (a^b(a^+b)^*) = (a^+b)^*$

### Constructing DFA with Derivatives

Every distinct regular expression obtained is a state.

A state is accepting if regex contains $\varepsilon$.

Construct DFA for $(a^+b)^*$

$\partial_a (a^+b)^* = \partial_a (a^+b) \cdot (a^+b)^* = a^b(a^+b)^*$

$\partial_b (a^+b)^* = \partial_b (a^+b) \cdot (a^+b)^* = \varepsilon(a^+b)^* = (a^+b)^*$

$\partial_a (a^b(a^+b)^*) = \partial_a (a^b(a^+b)^*) = a^b(a^+b)^*$

$\partial_b (a^b(a^+b)^*) = \partial_b (a^b(a^+b)^*) = (a^+b)^*$

### Constructing DFA with Derivatives

Every distinct regular expression obtained is a state.

A state is accepting if regex contains $\varepsilon$.

Construct DFA for $(a^+b)^*$

$\partial_a (a^+b)^* = \partial_a (a^+b) \cdot (a^+b)^* = a^b(a^+b)^*$

$\partial_b (a^+b)^* = \partial_b (a^+b) \cdot (a^+b)^* = \varepsilon(a^+b)^* = (a^+b)^*$

$\partial_a (a^b(a^+b)^*) = \partial_a (a^b(a^+b)^*) = a^b(a^+b)^*$

$\partial_b (a^b(a^+b)^*) = \partial_b (a^b(a^+b)^*) = (a^+b)^*$

### Constructing DFA with Derivatives

Every distinct regular expression obtained is a state.

A state is accepting if regex contains $\varepsilon$.

Construct DFA for $(a^+b)^*$

$\partial_a (a^+b)^* = \partial_a (a^+b) \cdot (a^+b)^* = a^b(a^+b)^*$

$\partial_b (a^+b)^* = \partial_b (a^+b) \cdot (a^+b)^* = \varepsilon(a^+b)^* = (a^+b)^*$

$\partial_a (a^b(a^+b)^*) = \partial_a (a^b(a^+b)^*) = a^b(a^+b)^*$

$\partial_b (a^b(a^+b)^*) = \partial_b (a^b(a^+b)^*) = (a^+b)^*$