INF 332: Languages & Automata

Chapter 9: Regular Expressions

Yliès Falcone

ylies.falcone@univ-grenoble-alpes.fr — www.ylies.fr

Univ. Grenoble-Alpes
-
Laboratoire d'Informatique de Grenoble - www.liglab.fr

Outline Chap. 9 - Regular Expressions

Outline Chap. 9 - Regular Expressions

Motivations

We want a notation more **concise** than automata to describe finite-state languages.

Example (Unix `grep`)

Writing a finite automaton to implement `grep` on Unix/Linux is unthinkable.

Example (Lexical analyzers)

Tools like `Lex` or `Flex` require us to specify the tokens of a programming language.

Example (String validation)

Checking email addresses, dates of birth, etc. in web forms.

Motivations (cont.)

Example (Regular expression for a valid email address)

According to RFC 5322[a]

```
  (?:[a-z0-9!#$%&'*+/=?^_`{|}~-]+(?:\.[a-z0-9!#$%&'*+/=?^_`{|}~-]+)*
  |  "(?:[\x01-\x08\x0b\x0c\x0e-\x1f\x21\x23-\x5b\x5d-\x7f]
     |  \\[\x01-\x09\x0b\x0c\x0e-\x7f])*")
@ (?:(?:[a-z0-9](?:[a-z0-9-]*[a-z0-9])?\.)+[a-z0-9](?:[a-z0-9-]*[a-z0-9])?
  |  \[(?:(?:25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)\.){3}
      (?:25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?|[a-z0-9-]*[a-z0-9]:
         (?:[\x01-\x08\x0b\x0c\x0e-\x1f\x21-\x5a\x53-\x7f]
         |  \\[\x01-\x09\x0b\x0c\x0e-\x7f])+)
    \])
```

[a] www.regular-expressions.info/email.html

- Automata describe languages in an *operational* way: as machines that process input step by step.
- Regular expressions describe languages in a *declarative/algebraic* way.

Key intuition

Every regular expression corresponds to some automaton, and vice versa. The two notations are different views of the same class of languages.

Outline Chap. 9 - Regular Expressions

Outline Chap. 9 - Regular Expressions

Regular expressions: syntax

Let Σ be an alphabet.

Definition (Regular expressions: syntax)

The set of *regular expressions over* Σ is defined inductively as follows:

- ϵ and $\emptyset$ are regular expressions over Σ.
- If $a \in \Sigma$, then a is a regular expression over Σ.
- If e and e' are regular expressions over Σ, then $e + e'$ is a regular expression (union).
- If e and e' are regular expressions over Σ, then $e \,\underline{\cdot}\, e'$ is a regular expression (concatenation).
- If e is a regular expression over Σ, then $e^{\underline{*}}$ is a regular expression (Kleene star).

Notation

The set of all regular expressions is denoted by RE.

Example (Regular expressions over $\Sigma = \{a, b\}$)

- $\emptyset$
- b
- $b \,\underline{\cdot}\, a$
- $a \,\underline{\cdot}\, \emptyset$
- $(\emptyset)^{\underline{*}}$
- $a^{\underline{*}}$
- $a + b$
- $a \,\underline{\cdot}\, (b + a)^*$
- $(b \underline{\cdot} a \underline{\cdot} b)^{\underline{*}} \,\underline{\cdot}\, b$

Outline Chap. 9 - Regular Expressions

Regular Expressions: Semantics

Regular expressions describe *languages*.

Definition (Regular Expressions: Semantics)

- The semantics is given by the mapping $\mathcal{L} : RE \rightarrow \mathcal{P}(\Sigma^*)$ which associates a (unique) language $\mathcal{L}(e)$ with every regular expression e.
- The mapping $\mathcal{L}$ is defined *inductively*:
  - $\mathcal{L}(\epsilon) = \{\epsilon\}$,
  - $\mathcal{L}(\emptyset) = \emptyset$,
  - $\mathcal{L}(a) = \{a\}$,
  - $\mathcal{L}(e + e') = \mathcal{L}(e) \cup \mathcal{L}(e')$,
  - $\mathcal{L}(e \underline{\cdot} e') = \mathcal{L}(e) \cdot \mathcal{L}(e')$,
  - $\mathcal{L}(e^{\underline{*}}) = \mathcal{L}(e)^*$.

Vocabulary

A language L is regular iff there exists a regular expression e such that $\mathcal{L}(e) = L$.

Example (Regular language)

The languages over $\{a, b\}$ denoted by the following regular expressions are regular:

- $(a)^{\underline{*}}$ – the language of words containing only a's
- $(a \underline{\cdot} b)^{\underline{*}}$ – the language of words formed by a finite repetition of the factor $a \cdot b$.

Convention and Notation

- From now on, we will no longer explicitly distinguish between
  - $\underline{\cdot}$ and $\cdot$, on the one hand;
  - $\underline{*}$ and $*$, on the other hand.
- We also want to be able to write expressions such as
  - $a + b + c$ instead of $(a + b) + c$, and
  - $a + b^*$ instead of $(a + (b^*))$.
- To avoid ambiguity, we allow the use of parentheses and adopt the following precedence rules, in decreasing order:
  1. $*$
  2. $\cdot$
  3. $+$
- We also write ee' instead of $e \cdot e'$.

Example (Convention and Notation)

The expressions

- $e_1 + e_2^*$ and $e_1 + (e_2)^*$, on the one hand,
- $e_1 + e_2 \cdot e_3$ and $e_1 + (e_2 \cdot e_3)$, on the other hand,

denote the same sets.

Examples of Regular Expressions

Let $\Sigma = \{a, b\}$.

Example (Words containing only a's)

Example (Words consisting of repetitions of the factor ab)

Example (Words with an even number of a's)

Example (Words with an odd number of b's)

Example (Words with an even number of a's or an odd number of b's)

Notation – Operator $^+$ (exponent)

Operator $^+$ (exponent)

Let e be a regular expression. We write e^+ for $e \cdot e^*$.

The regular expression e^+ denotes the language of words that are formed by concatenating at least one word from the language denoted by e.

Example (Regular expression with operator $^+$ (exponent))

Let $\Sigma = \{a, b, c, d\}$, and consider the regular expression $e = ab + cd$.
Then the regular expression e^+ denotes the language

$$\{ab, cd, abab, abcd, cdab, cdcd, \ldots\}$$

Property

If e is a regular expression such that $\epsilon \in L(e)$, then $L(e^+) = L(e^*)$.

Notation – Operator $^?$ (superscript)

Operator $^?$ (superscript)

Let e be a regular expression. We write $e?$ for $\epsilon + e$.

The regular expression $e?$ denotes the language of words that consist of at most one occurrence of a word in the language denoted by e.

Example (Regular expression with operator $^?$ (superscript))

Let $\Sigma = \{a, b, c\}$, and consider the regular expression $e = ab?c$.
Then e denotes the language

$$\{ac, abc\}.$$

Outline Chap. 9 - Regular Expressions

Equivalence between Regular Expressions

Definition (Equivalence between two regular expressions)

Two regular expressions e_1 and e_2 are said to be *equivalent* when

$$L(e_1) = L(e_2).$$

(That is, when these regular expressions denote the same language.)

Notation

When e_1 and e_2 are equivalent, we write $e_1 \equiv e_2$.

Remark The relation $\equiv$ on regular expressions is indeed an equivalence relation, since language equality is an equivalence relation. □

Equivalence of Regular Expressions: Basic Identities

Classical identities

Regular expression	Equivalent regular expression	Remark
$e + \emptyset$	e	trivial
$e \cdot \epsilon$	e	trivial
$e \cdot \emptyset$	$\emptyset$	trivial
$(e + f) + g$	$e + (f + g)$	associativity
$(e \cdot f) \cdot g$	$e \cdot (f \cdot g)$	associativity
$e \cdot (f + g)$	$(e \cdot f) + (e \cdot g)$	distributivity
$(e + f) \cdot g$	$(e \cdot g) + (f \cdot g)$	distributivity
$e + f$	$f + e$	commutativity

Equivalence of Regular Expressions: More Identities

Classical identities

Regular expression	Equivalent regular expression	Remark
e^*	$\epsilon + e \cdot e^*$	unrolling
e^*	$\epsilon + e^* \cdot e$	unrolling
$(\emptyset)^*$	ϵ	Kleene star
$e + e$	e	idempotence
$(e^*)^*$	e^*	idempotence
$e + ee^*$	e^*	absorption
$e^* e + e$	e^*	absorption

Equivalence between regular expressions

Let Σ be an alphabet such that $a \in \Sigma$ and e a regular expression over Σ.

Example (Equivalent regular expressions)

- The expressions $(a + \epsilon)^*$ and a^* are equivalent.
  - $L((a+\epsilon)^*) \subseteq L(a^*)$. Let $w \in L((a+\epsilon)^*)$. By the semantics of regular expressions, either (i) $w = \epsilon$, or (ii) $w = w_1 \cdot w_2 \cdots w_n$ with $w_i \in L(a + \epsilon)$.
    Case (i): $w = \epsilon$, and thus $w \in L(a^*)$ by the semantics of a^* (Kleene closure of $L(a)$).
    Case (ii): w is formed by concatenating words a and ϵ, and can therefore be written as $w = w_1' \cdots w_m'$ with $m \leq n$ and $w_i = w_i'$. Thus $w \in \{a\}^* = L(a^*)$.
  - $L((a+\epsilon)^*) \supseteq L(a^*)$. We have $L(a^*) = L(a)^* = (\{a\})^* \subseteq (\{a\} \cup X)^*$, for any language X, in particular for $X = \{\epsilon\}$.
- The expressions $(e + \epsilon)^*$ and e^* are equivalent. The proof follows the same principle as above, reasoning on $L(e)$ instead of $L(a) = \{a\}$.
- The expressions $\epsilon + e + ee^*e$ and e^* are equivalent, for any regular expression e.
  - $\epsilon, e, ee^*e \in e^*$ by definition of e^*.
  - Every word of e^* can be expressed either as ϵ, as a single e, or as a word of the form ee^*e.

Is equivalence between regular expressions decidable?

Simplification of regular expressions

Principle of simplifying regular expressions

If e and e' are two equivalent regular expressions (i.e. $e \equiv e'$, meaning $L(e) = L(e')$), then we can substitute e with e' in any regular expression r without changing the language denoted by r.

Example (Simplification of regular expressions)

Consider the regular expression $r = (a + \epsilon)^* + b^* + c \cdot d^*$.
Since $L\big((a+\epsilon)^*\big) = L(a^*)$, r can be simplified to $a^* + b^* + c \cdot d^*$.

Some useful facts for simplification

Let e_1 and e_2 be two regular expressions.

- If $L(e_1) \subseteq L(e_2)$, then $L(e_1 + e_2) = L(e_2)$. So $e_1 + e_2$ can be replaced by e_2 in any regular expression without changing the denoted language.
- $L(e \cdot \epsilon) = L(\epsilon \cdot e) = L(e)$.

Can we automatically determine whether $e_1 + e_2$ can be replaced by e_2? In other words, is $L(e_1) \subseteq L(e_2)$ decidable?

Simplification of regular expressions: examples

Example (Simplification of regular expressions)

Regular expression	Simplified regular expression
$e^* + e$	e^*
$e^+ + e$	e^+
$e^+ + \epsilon$	e^*
$(e + \epsilon)^*$	e^*
$a + ab^*$	ab^*
$e + ee^*e$	e^+
$\epsilon + e + ee^*e$	e^*

Remark See the exercises for more examples of simplification, and for the proofs of equivalence between these regular expressions. □

Regular expressions and closure properties

Observation

The syntax of regular expressions directly encodes the closure properties of regular languages.

- $e + f$ corresponds to closure under **union**.
- ef corresponds to closure under **concatenation**.
- e^* corresponds to closure under the **Kleene star**.

Remark

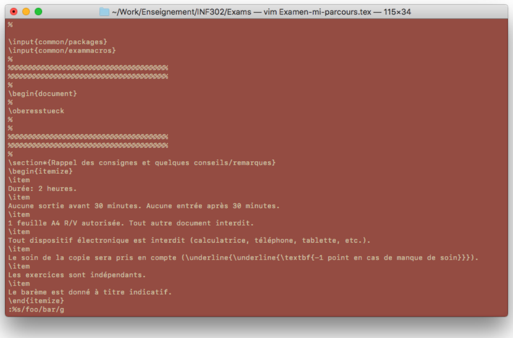
We have already shown that finite-state (regular) languages are closed under these operations. Regular expressions provide a **compact algebraic notation** for these closures.

Outline Chap. 9 - Regular Expressions

UNIX commands

Many UNIX commands allow specifying character strings using regular expressions.

- Text editors: `vi(m)`, `emacs`, `nano`



UNIX commands

- Search for a string in a text: grep

```
grep 'exam' /Users/prof/teaching/*.*
```

Displays the lines containing exam in all files (with any extension) of the directory /Users/prof/teaching/.

- String transformation in a text/file: sed

```
sed 's/2024/2025/' old.tex > new.tex
```

Replaces all occurrences of 2024 in old.tex by 2025 and writes the result into new.tex.

- File search: find

```
find . -name '*exam*' -print
```

Searches in the current directory and its subdirectories (.) for files whose names match the pattern *exam*.

UNIX commands (cont'd)

- Expression evaluation: expr

```
$expr "$string" : "reg_expression"
```

Compares $string against reg_expression. Returns the length of the longest matching prefix.

- Data filtering and processing: awk

```
pattern { action }
```

```
BEGIN { print "START" }
      { print         }
END   { print "STOP"  }
```

Adds one line with START at the beginning and one line with STOP at the end of a file.

```
BEGIN { print "File\tOwner"}
{ print $8, "\t", $3}
END { print " - DONE -" }
```

Script FileOwner that prints the owner of each file.

```
ls -l | awk -f FileOwner
```

Using the FileOwner script on the command line.

Outline Chap. 9 - Regular Expressions

Summary

Summary

- Regular expressions
  - syntax,
  - semantics.
- Manipulations of regular expressions
  - equivalence,
  - simplification.
- Practical applications: UNIX commands using regular expressions.