Formal languages, formal grammars, and regular expressions

LING 570

Fei Xia

Unit #1

- Formal grammar, language and regular expression
- Finite-state automaton (FSA)
- Finite-state transducer (FST)
- Morphological analysis using FST

Regular expression

- Two concepts:
  - Regular expression in *formal language theory*
  - Regular expression (or **pattern**) in *pattern matching/programming languages*:
    - Ex: /^\d+(\.\d+)\1/
- Both concepts describe a set of strings.
- The two concepts are closely related, but the latter is often more *<u>expressive</u>* than the former.

Outline

- Formal languages
  - Regular languages
  - Context-free languages
  - ...

- Regular expression in formal language theory

- Formal grammars
  - Regular grammars
  - Context-free grammars

- “Regular expression” in pattern matching

Formal languages

Definition of formal language

- An alphabet is a finite set of symbols:
  - Ex: $\Sigma = \{a, b, c\}$
- A string is a finite sequence of symbols from a particular alphabet juxtaposed:
  - Ex: the string "baccab"
  - Ex: *empty string* ϵ
- A formal language is a set of strings defined over some alphabet.
  - Ex1: {aa, bb, cc, aaaa, abba, acca, baab, bbbb, ….}
  - Ex2: $\{a^n b^n \mid n > 0\}$
  - Ex3: the *empty set* ϕ

Definition of regular languages

- The class of <u>regular languages</u> over an alphabet Σ is formally defined as:
  - The empty set, ϕ, is a regular language
  - $\forall$ a $\in \Sigma \cup \{\epsilon\}$, {a} is a regular language.
  - If L1 and L2 are regular languages, then so are:
    - (a) $L_1 \bullet L_2 = \{x\ y \mid x \in L_1;\ y \in L_2\}$ (concatenation)
    - (b) $L_1 \bigcup L_2$ (union or disjunction)
    - (c) $L_1^* = \{x_1\ x_2 \ldots x_n \mid x_i \in L_1,\ n \in N\}$ (Kleene closure)
  - There are no other regular languages.

Kleene star

Another way to define L*:

- $L^1 = L$
- $L^n = L^{n-1} \bullet L$
- $L^* = \{\epsilon\} \cup L^1 \cup L^2 \cup \ldots$

Examples:

- $L = \{a, bc\}$
- $L^2 = \{aa, abc, bca, bcbc\}$
- $L^* = \{\epsilon, a, bc, aa, abc, bca, bcbc, aaa, \ldots.\}$

Properties

- Regular languages are closed under
  - Concatenation
  - Union
  - Kleene closure

- Regular languages are also closed under:
  - Intersection: $L_1 \cap L_2$
  - Difference: $L_1 - L_2$
  - Complementation: $\Sigma^* - L_1$
  - Reversal

Are the following languages regular?

- {a, aa, aaa, ….}
- Any finite set of strings

- {xy | $x \in \Sigma^*$, and y is the reverse of x}
- {xx | $x \in \Sigma^*$}

- $\{a^n b^n \mid n \in N\}$
- $\{a^n b^n c^n \mid n \in N\}$

➔ To prove a language is not regular or context-free, use pumping lemma.

Regular expression

Definition of Regular Expression (as in formal language theory)

- The set of regular expressions is defined as follows:
  - (1) Every symbol of Σ is a regular expression
  - (2) ϵ is a regular expression
  - (3) If $\mathbf{r_1}$ and $\mathbf{r_2}$ are regular expressions, so are
    $(\mathbf{r_1})$, $\mathbf{r_1}\ \mathbf{r_2}$, $\mathbf{r_1} \mid \mathbf{r_2}$, $\mathbf{r_1}^*$
  - (4) Nothing else is a regular expression.

Examples

- Let r1=a, r2=b

- ab matches the RegEx r1 r2, and (r1 | r2)*, but not (r1 | r2).

Examples

- ab^*c
- $a\ (0|1|2|..|9)^*\ b$
- $(CV \mid CCV)^+\ C?C?$: C is a consonant, V is a vowel

Other operations that we can use:

- $a^+ = a\ a^*$
- $a? = (a \mid \epsilon)$

Relation between regular language and Regex

- They are equivalent:
  - With every regular expression we can associate a regular language.
  - Conversely, every regular language can be obtained from a regular expression.

- Examples:
  - Regular expression = ab*c
  - Regular language = {ac, abc, abbc, ….}.
    = {ab*c}

Formal grammars

Definition of formal grammar

A formal grammar is a concise description of a formal language. It is a (N, Σ, P, S) tuple:

- A finite set *N* of *nonterminal symbols*
- A finite set Σ of *terminal symbols* that is disjoint from *N*
- A finite set *P* of *production rules*, each of the form:
  $(\Sigma \cup N)^* N (\Sigma \cup N)^* \rightarrow (\Sigma \cup N)^*$
- A distinguished symbol S $\in$ N that is the *start symbol*

Chomsky hierarchy

The left-hand side of a rule must contain at least one non-terminal.
α, β, $\gamma \in$ (N $\bigcup$ Σ)*, A,B $\in$ N, a $\in \Sigma$

- Type 0: unrestricted grammar: no other constraints.
- Type 1: Context-sensitive grammar:
  The rules must be of the form: α A $\beta \rightarrow \alpha\ \gamma\ \beta$
- Type 2: Context-free grammar (CFGs):
  The rules must be of the form: A $\rightarrow \alpha$
- Type 3: Regular grammar: The rules are of the forms:
  right regular grammar: A $\rightarrow$ a, A $\rightarrow$ aB, or A $\rightarrow \epsilon$
  $left$ regular grammar: A $\rightarrow$ a, A $\rightarrow$ Ba, or A $\rightarrow \epsilon$

Are there other kinds of grammars?

Ex: CFG vs. Reg Grammar

- Let G1 be a CFG and G2 be a Regular grammar, both with the alphabet {a, b} and the start symbol S.

- Rules for G1: $\{S \rightarrow a\ S\ b,\ S \rightarrow \epsilon\}$

- Rules for G2:

  $\{S \rightarrow a\ S,\ S \rightarrow b\ S1,\ S1 \rightarrow bS1,\ S1 \rightarrow \epsilon,\ S \rightarrow \epsilon\}$

- $L(G1) = \{a^n\ b^n\}$, $L(G2) = \{a^*\ b^*\}$
- L(G1) is a subset of L(G2).

CFG vs. Reg Grammar (cont)

- CFG has less restriction on the FORM of the production rules than regular grammar; that's why type 0 grammar is called “unrestricted”.
- Thus, CFG is more “powerful” in expressing the constraints on the strings generated by the grammar than regular grammar:
  - In L(G1), the numbers of a and b are the same.
  - L(G2) does not have that constraint.
- As a result, L(G1) is a subset of L(G2).

Strings generated from a grammar

- The rules are:

  $S \rightarrow x \mid y \mid z \mid S + S \mid S - S \mid S * S \mid S/S \mid (S)$

- What strings can be generated?

- A grammar is ambiguous if there exists at least one string which has multiple parse trees.

- Is this grammar ambiguous?

Languages generated by grammars

- Given a grammar G, L(G) is the set of strings that can be generated from G.

- Ex: $G = (N, \Sigma, P, S)$

  $N = \{S\}$, $\Sigma = \{a, b, c\}$

  $P = \{\ S \rightarrow a\,S\,b,\ S \rightarrow c\ \}$

What is L(G)?

$L(G) = \{a^n\ c\ b^n\}$

The relation between regular grammars and regular languages

- The regular grammars describe exactly all regular languages.

- All the following are equivalent:
  - Regular language: alphabet, operations
  - Regular expression: alphabet, operations
  - Regular grammar: terminals, non-terminals, production rules
  - Finite state automaton (FSA): alphabet, states, edges

Relation between grammars and languages

Chomsky hierarchy	Grammars	**Languages**	**Minimal automaton**
Type-0	Unrestricted	Recursively enumerable	Turing machine
Type-1	Context-sensitive	Context-sensitive	Linear-bounded
Type-2	Context-free	Context-free	Nondeterministic pushdown
Type-3	Regular	Regular	Finite state

Relation between grammars and languages (from wikipedia page)**

Chomsky hierarchy	Grammars	**Languages**	**Minimal automaton**
Type-0	Unrestricted	Recursively enumerable	Turing machine
n/a	(no common name)	Recursive	Decider
Type-1	Context-sensitive	Context-sensitive	Linear-bounded
n/a	Indexed	Indexed	Nested stack
n/a	Tree-adjoining	Mildly context-sensitive	Thread
Type-2	Context-free	Context-free	Nondeterministic pushdown
n/a	Deterministic context-free	Deterministic context-free	Deterministic pushdown
Type-3	Regular	Regular	Finite state

How about human languages?

- Are they formal languages?
  - What is the alphabet?
  - What is a string?

- What type of formal languages are they?

Ex: crossing dependency in Dutch: $N_1\ N_2\ V_1\ V_2$

English: John believes Mary bought a car.

Dutch: John Mary believes bought a car.

Outline

- Formal language
  - Regular language
- Regular expression in formal language theory
- Formal grammar
  - Regular grammar
- **Patterns in pattern matching → J&M-ed2 2.1**

Patterns in Perl or Python

[ab]	a|b
.	match any character
^	the starting position in a string
$	the ending position in a string
(..)	defines a marked subexpression
a*	match “a” zero or more times
a+	match “a” one or more time
a?	match “a” zero or one time
a{n,m}	“a” appears n to m times

Special symbols in the patterns

\s match any whitespace char, [\t\n\r\f\v]

\d match any digit, [0-9]

\w match any letter or digit, [a-zA-Z0-9_]

\S match any non-whitespace char

…

\+, \-, \., \?, *, …

Examples

Integer: `(\+|\-)?\d+`

Real number: `(\+|\-)?\d+\.\d+`

Scientific notation: `(\+|\-)? \d+ (\.\d+)?e (\+|\-)?\d+`

Integer, real number, or scientific notation:

`(\+|\-)? \d+ (\.\d+)? (e (\+|\-)?\d+)?`

Numbered groups

/^(.*)\1$/ $\Leftrightarrow$ $\{ xx \mid x \in \Sigma^* \}$

/^(.+)a(.+)\1\2$/ $\Leftrightarrow$ $\{xayxy \mid x, y \in \Sigma^*\}$

➔ The extra power comes from the ability to refer to "numbered groups".