Regular Expressions and Languages

Prof. Licia Sbattella

aa 2007-08

Translated and adapted by L. Breveglieri

REGULAR EXPRESSIONS AND LANGUAGES

REGULAR LANGUAGES are the simplest family of formal languages. This family can be defined in different ways;

- algebraically
- by means of generating grammars
- by means of recognizer machines

$$\boxed{\begin{array}{c} \Sigma = \{a_1, a_2, ...a_i\} \\ \cdot \quad \bigcup \quad * \\ \{a_1\}, \{a_2\}, ... \{a_i\} \ \emptyset \end{array}}$$

REGULAR EXPRESSION (regexp) is a string *r* over the letters of the terminal alphabet Σ and containing a few metasymbols, according to the following rules (where *s* and *t* represent regular expressions):

metasymbols

$$\boxed{\cdot \quad \bigcup \quad * \quad \emptyset \quad (\ )}$$

$$\boxed{\begin{array}{lll} 1.\ r = \emptyset & 3.\ r = (s \bigcup t) & 5.\ r = (s)^* \\ 2.\ r = a,\ a \in \Sigma & 4.\ r = (s.t) \text{ o } r = (st) & \end{array}}$$

Pay attention: U is often denoted | (vertical bar)

PRECEDENCE OF OPERATORS: star (∗), concatenation (.), union (U)

In the regexp it is permitted to use ε, because it holds: $\boxed{\varepsilon = \emptyset^*}$

THE MEANING OF A REGEXP *e* is a language L_e over the alphabet Σ according to the following table:

A LANGUAGE IS REGULAR
if it is the meaning
(= if it is generated)
by a regular expression

	regexp	language L_r
1.	ε	$\{\varepsilon\}$
2.	$a \in \sum$	$\{a\}$
3.	$s \bigcup t$ o $s \mid t$	$L_s \bigcup L_t$
4.	$s.t$ o st	$L_s.L_t$
5.	$s*$	$L_s^{\ *}$

EXAMPLE 1: L_e is the (unary) language of the multiples of three

$$e = (111)*$$

$$L_e = \{\varepsilon, 111, 111111, ...\} = \{1^n \mid n \bmod 3 = 0\}$$

$$e_1 = 11(1)* \qquad L_e \neq L_{e_1}$$

$$L_{e_1} = \{11, 111, 1111, 11111, ...\} = \{111^n \mid n \ \geq 0\}$$

EXAMPLE 2: Let Σ $=$ { $+$, $-$, *d* }, where *d* denotes a decimal digit 0,1,…,9. Define the regexp *e* that generates the language of integers, possibly signed.

$$\boxed{e = (+ \cup - \cup \varepsilon) dd\,*}$$

$$\boxed{L_e = \{+, -, \varepsilon\}\{d\}\{d\}^*}$$

EXAMPLE 3: define a language over the binary alphabet { a, b }, such that the number of letters *a* in each string is odd and that each string contains at least one letter *b*.

$$\boxed{\begin{array}{l} A_p b A_d \mid A_d b A_p \\ A_p = b^*(ab^*ab^*)^* \quad A_d = b^*ab^*(ab^*ab^*)^* \end{array}}$$

FAMILY OF REGULAR LANGUAGES (REG):
is the collection of all regular languages.

FAMILY OF FINITE LANGUAGES (FIN):
is the collection of all languages of finite cardinality
(= that contain only finitely many strings).

Caution: REG and FIN are sets of sets of strings,
not sets of strings themselves.

EVERY FINITE LANGUAGE IS REGULAR, because it is the (finite) union of finitely many strings, and each string is in turn the concatenation of finitely many letters or in general of alphabetical symbols.

$$\boxed{(x_1 \cup x_2 \cup \ldots \cup x_k) = (a_{1_1} a_{1_2} \ldots a_{1_n} \cup \ldots \cup a_{k_1} a_{k_2} \ldots a_{k_m})}$$

But the family of regular languages also contains many (actually infinitely many) languages of infinite cardinality (= that contain infinitely many strings).

$$\boxed{FIN \subset REG}$$

SUBEXPRESSION OF A REGEXP

1. consider e regexp that is fully parenthesized
2. number every alphabetical symbol occurring in the regexp
3. isolate the subexpressions and put them into evidence

$$e = (a \cup (bb))^{*}(c^{+} \cup (a \cup (bb)))$$

$$e_N = (a_1 \cup (b_2 b_3))^{*}(c_4{}^{+} \cup (a_5 \cup (b_6 b_7)))$$

$$\begin{array}{lll}
(a_1 \cup (b_2 b_3))^{*} & & c_4{}^{+} \cup (a_5 \cup (b_6 b_7)) \\
a_1 \cup (b_2 b_3) & c_4{}^{+} & a_5 \cup (b_6 b_7) \\
a_1 \quad b_2 b_3 & c_4 & a_5 \quad b_6 b_7 \\
\quad b_2 \quad b_3 & & \qquad b_6 \quad b_7
\end{array}$$

DIFFERENT PARENTHESIZATIONS (and hence interpretations) OF A REGEXP

$$f = (a \bigcup bb)^{*}(c^{+} \bigcup a \bigcup bb)$$

$$(a \bigcup bb)^{*}(c^{+} \bigcup (a \bigcup bb))$$

$$(a \bigcup bb)^{*}((c^{+} \bigcup a) \bigcup bb)$$

$$(a_1 \bigcup b_2 b_3)^{*} \qquad c_4{}^{+} \bigcup a_5 \bigcup b_6 b_7$$

$$a_1 \bigcup b_2 b_3 \qquad c_4{}^{+} \qquad a_5 \bigcup b_6 b_7 \qquad c_4{}^{+} \bigcup a_5 \qquad b_6 b_7$$

$$a_1 \quad b_2 b_3 \qquad c_4 \qquad a_5 \quad b_6 b_7$$

$$b_2 \quad b_3 \qquad\qquad b_6 \quad b_7$$

CHOICE: the union and iteration operators that occur in a regexp correspond to some (sometimes infinitaly many) possible alternatives.

By choosing one of the available alternatives, a new regexp that defines a smaller language than the original one, is obtained.

$e_k, 1 \leq k \leq n,$	choice for the union	$e_1 \bigcup ... \bigcup e_k \bigcup ... e_n$
e	choice for the iteration	e^*, e^+, e^n
ε	choice for the iteration	e^*

Given a regexp e_1, one can always DERIVE another regexp e_2 by replacing a subexpression (short s.e.) of the regxexp e_1 with one of the possible choices.

DERIVATION RELATION between two regexps *e’* and *e’’*

$e' \Rightarrow e''$ If the two regexp *e*’ and *e*’’ can be factored as

$$e' = \alpha\beta\gamma \qquad e'' = \alpha\delta\gamma$$

with β s.e. of e', δ s.e. of e'', δ is a choice of β

DERIVATION STEPS can be applied in sequence, two or more times:

$e_0 \overset{n}{\Rightarrow} e_n$ if $e_0 \Rightarrow e_1, e_1 \Rightarrow e_2, \;\; .... \;, \;\; e_{n-1} \Rightarrow e_n$

$e_0 \overset{+}{\Rightarrow} e_n$ e_0 derives e_n $\text{in } n \geq 1$ steps

$e_0 \overset{*}{\Rightarrow} e_n$ e_0 derives e_n $\text{in } n \geq 0$ steps

VARIOUS EXAMPLES

One-step derivation and multiple-step derivation:

$$a^* \bigcup b^+ \Rightarrow a^* \qquad a^* \bigcup b^+ \Rightarrow b^+$$

$$a^* \bigcup b^+ \Rightarrow a^* \Rightarrow \varepsilon \qquad a^* \bigcup b^+ \overset{2}{\Rightarrow} \varepsilon \qquad a^* \bigcup b^+ \overset{+}{\Rightarrow} \varepsilon$$

$$a^* \bigcup b^+ \Rightarrow b^+ \Rightarrow bbb \qquad a^* \bigcup b^+ \overset{2}{\Rightarrow} bbb \qquad a^* \bigcup b^+ \overset{+}{\Rightarrow} bbb$$

Of the regexp that are obtained by derivation from the initial regexp *e*, some may contain letters of the alphabet and also metasymbols, some may contain only letters of the alphabet. The latter constitute the language generated by the initial regexp *e*.

THE LANGUAGE *L*(*r*) DEFINED
(or GENERATED) BY A GIVEN REGEXP *r* IS:

$$\boxed{L(r) = \left\{ x \in \Sigma^* \mid r \overset{*}{\Rightarrow} x \right\}}$$

Two regexps are EQUIVALENT if they define (generate) the same language.

THE LANGUAGE DEFINED BY A DERIVED REGEXP IS CONTAINED
IN THE LANGUAGE DEFINED BY THE DERIVING REGEXP

There may exist several derivation orders that generate the same string, which are however substantially equivalent.

EXAMPLES:

$1.(ab)^* \Rightarrow abab$	$5.a^*(b \cup c \cup d) f^+ \Rightarrow aaa(b \cup c \cup d) f^+$
$2.(ab \cup c) \Rightarrow ab$	$6.a^*(b \cup c \cup d) f^+ \Rightarrow a^* c f^+$
$3.a(ba \cup c)^* d \Rightarrow ad$	$7.a^*(b \cup c \cup d) f^+ \overset{+}{\Rightarrow} aaacf^+$ in 2 steps
$4.a(ba \cup c)^* d \Rightarrow a(ba \cup c)(ba \cup c)d$	$8.a^*(b \cup c \cup d) f^+ \overset{+}{\Rightarrow} aaacff$ in 3 steps

AMBIGUITY OF REGULAR EXPRESSIONS

A string (phrase) may be obtained by two derivations, that differ not only in the ordere of choices, but in a more substantial way.

Examples:

$$(a \bigcup b)^* a(a \bigcup b)^*$$

$$(a \bigcup b)^* a(a \bigcup b)^* \Rightarrow (a \bigcup b)a(a \bigcup b)^* \Rightarrow aa(a \bigcup b)^* \Rightarrow aa\varepsilon \Rightarrow aa$$

$$(a \bigcup b)^* a(a \bigcup b)^* \Rightarrow \varepsilon a(a \bigcup b)^* \Rightarrow \varepsilon a(a \bigcup b) \Rightarrow \varepsilon aa \Rightarrow aa$$

A regexp *f* is AMBIGUOUS if and only if, the corresponding marked regexp *f'* generates two marked strings *x* and *y* such that, if the indices are removed, the underlying ordinary strings are identical.

$$f\,' = (a_1 \bigcup b_2)^* a_3 (a_4 \bigcup b_5)^*$$

defines a regular language over the (marked) alphabet

$$\{a_1, b_2, a_3, a_4, b_5\}$$

$a_1 a_3$, $a_3 a_4$ exhibit the ambiguity phenomenon

EXAMPLE (ambiguity)

$(aa \mid ba)^* a \mid b(aa \mid ba)^*$ is ambiguous

in fact, the marking

$$(a_1 a_2 \mid b_3 a_4)^* a_5 \mid b_6 (a_7 a_8 \mid b_9 a_{10})^*$$

generates the two strings

$$b_3 a_4 a_5 \qquad b_6 a_7 a_8$$

that project ambiguously onto the same phrase: baa

OTHER OPERATORS

Basic operators: union, concatenation, star.
Derived operators: power, cross.

$$e^h = \underbrace{ee...e}_{h \text{ times}} \qquad e^+ = ee^*$$

EXAMPLE: floating point fractional numbers, with or without sign and exponent

$\Sigma = \{+, -, \bullet, E, d\}$

$r = s.c.e$

$s = (+ \cup - \cup \varepsilon)$ sign ±, optional

$c = (d^+ \bullet d^* \cup d^* \bullet d^+)$ constant, integer or fractional

$e = (\varepsilon \cup E(+ \cup - \cup \varepsilon)d^+)$ exponent, optional, preceded by E

$(+ \cup - \cup \varepsilon)(d^+ \bullet d^* \cup d^* \bullet d^+)(\varepsilon \cup E(+ \cup - \cup \varepsilon)d^+)$

$+dd \bullet E - ddd \qquad +12 \bullet E - 341 \quad 12.10^{-341}$

OTHER OPERATORS

REPETITION: from *k* to *n* > *k* times
OPTIONALITY:
INTERVAL OF AN ORDERED SET:

$$[a]_k^n = a^k \cup a^{k+1} \cup \ldots \cup a^n$$

$$[a] = (\varepsilon \cup a)$$

$$(0 \ldots 9) \quad (\mathrm{a} \ldots \mathrm{z}) \quad (\mathrm{A} \ldots \mathrm{Z})$$

By admitting in a regexp also the presence of the set-theoretic operations INTERSECTON, COMPLEMENT and DIFFERENCE, one obtains the so-called EXTENDED REGULAR EXPRESSIONS.

EXAMPLE (intersection) – allows to express the request that the phrases of the language satisfy two conditions (both, not either one).

alphabet $\{a, b\}$

string contains bb $(a \mid b)^* bb (a \mid b)^*$

string length is even $((a \mid b)^2)^*$

$$(a \mid b)^* bb (a \mid b)^* \cap ((a \mid b)^2)^*$$

the same result of intersection
but obtained in a more
complicated way
(*bb* is sourrounded by two strings
both of either even or odd length)

↓

$$((a \mid b)^2)^* bb ((a \mid b)^2)^* \mid (a \mid b)((a \mid b)^2)^* bb (a \mid b)((a \mid b)^2)^*$$

EXMPLE (complement and intersection)

The regexp $r = (a\ b)^*$ generates the strings that

- do not start by *b*
- do not end by *a*
- do not contain either the substring *aa* or the substring *bb*

$$r' = \neg(b(a \bigcup b)^* \bigcup (a \bigcup b)^* a \bigcup (a \bigcup b)^* (aa \bigcup bb)(a \bigcup b)^*)$$

De Morgan theorem:

$$r' = \neg b(a \bigcup b)^* \bigcap \neg(a \bigcup b)^* a \bigcap \neg((a \bigcup b)^* (aa \bigcup bb)(a \bigcup b)^*)$$

EXAMPLE: how to transform an extended regexp over the alphabet { *a*, *b*, *c* } into an equivalent regexp, such that it uses only the basic operators U, $*$ and .

$$\neg((a \mid b)(\neg\emptyset))$$

$$= \ \neg((a \mid b)(a \mid b \mid c)^*) = \varepsilon \mid c(a \mid b \mid c)^*$$

CLOSURE OF THE REG FAMILY WITH RESPECT TO OPERATIONS / 1

Let θ be an operator, that applied to one language (operand), or to a pair of languages, produces another language (result).

A FAMILY OF LANGUAGES IS SAID TO BE CLOSED WITH RESPECT TO AN OPERATOR θ, IF THE RESULT LANGUAGE BELONGS TO THE SAME FAMILY AS THE OPERAND LANGUAGE(S).

PROPERTY: the family REG of the regular languages is CLOSED with respect to the following operators: concatenation, union, star, and therefore is closed also w.r.t. the derived operators, e.g. cross, repetition, optionality, etc (this follows from the definition of regexp itself).

This implies that any two regular languages L_1 and L_2 can be combined by means of the above mentioned operators, and yet the resulting language remains in the family of the regular languages.

CLOSURE OF THE REG FAMILY WITH RESPECT TO OPERATIONS / 2

AN EVEN STRONGER PROPERTY: the family REG of the regular languages is the *smallest* family of languages such that both the following properties hold:

- REG contains all finite languages
- REG is closed w.r.t. concatenation, union and star

Later the LIB family will be defined (of the so-called context-free languages), which is closed w.r.t. concatenation, union and star as well, but is not the smallest such family, because the following strict containment holds:

$$\boxed{REG \subset LIB}$$

Moreover, REG is closed w.r.t. INTERSECTION, COMPLEMENT (and DIFFERENCE) and MIRRORING (this is difficult to prove directly, but is esay by using the concepts and tools of automata theory).

LINGUISTIC ABSTRACTION

Linguistic abstraction transforms the phrases of a real, effective language, and gives them a simpler form, called abstract representation.

To do so, the symbols of the effective alphabet are discarded and replaced by those of the abstract alphabet.

AT THE ABSTRACT LEVEL, THE TYPICAL STRUCTURES OF MOST ARTIFICIAL LANGUAGES CAN BE OBTAINED BY THE COMPOSITON OF FEW ELEMENTARY PARADIGMS, AND BY THE BASIC LANGUAGE OPERATIONS SUCH AS CONCATENATION, UNION AND ITERATION.

From the abstract language to the effective language → choice of the actual lexical elements (e.g. keywords, identifiers, ...)

COMPILER DESIGN MAKES REFERENCE TO THE ABSTRACT LANGUAGE TO PROCESS, RATHER THAN TO THE EFFECTIVE ONE.

Artificial languages (e.g. programming languages) contain few abstract structures, among which lists play a relevant role; lists can be esaily modeled by regexps.

EFFECTVE AND ABSTRACT LISTS

A list is a sequence of a number of elements, not fixed in advance.

A list can be generated by the regexp e^+ or e^*, if the empty list is admitted.

The element e can be a teminal (= alphabetical symbol) or a compound object of some kind (for instance, a list itself …).

LISTS WITH SEPARATORS AND START-MARKER OR END-MARKER

$$\boxed{ie(se)^* f \qquad\qquad i[e(se)^*]f}$$

EXAMPLES:

$$\overbrace{begin}^{i}\,\overbrace{istr_1}^{e}\overset{s}{;}\overbrace{istr_2}^{e}\overset{s}{;}\ldots\overset{s}{;}\overbrace{istr_n}^{e}\,\overbrace{end}^{f}$$

$$\overbrace{procedure\ STAMPA(}^{i}\,\overbrace{par_1}^{e}\overset{s}{,}\overbrace{par_2}^{e}\overset{s}{,}\ldots\overset{s}{,}\overbrace{par_n}^{e}\overset{f}{)}$$

$$\overbrace{array\ MATRICE\,'['}^{i}\,\overbrace{\mathrm{int}_1}^{e}\overset{s}{,}\overbrace{\mathrm{int}_2}^{e}\overset{s}{,}\ldots\overset{s}{,}\overbrace{\mathrm{int}_n}^{e}\overbrace{']'}^{f}$$

SUBSTITUTION (REPLACEMENT): operation that replaces the terminal characters of the source language with the phrases of the target (or destination) language.

$$L \subseteq \Sigma^* \quad L_\mathrm{b} \subseteq \Delta^*$$

$x \in L \quad x = a_1 a_2 ... a_\mathrm{n}$ and for some $a_i = b$

Substituting the language L$_b$ to the letter *b* in the string *x* produces a language over the alphabet (Σ \ { b }) U Δ, defined as follows:

$$\left\{ y \mid y = y_1 y_2 ... y_n \wedge (\text{if } a_i \neq b \text{ then } y_i = a_i \text{ else } y_i \in L_b \right\}$$

NESTED LISTS (sometimes called PRECEDENCE LISTS)

Lists may contain atomic objects as elements, but also other lists, of lower level.

$$lista_1 = i_1 lista_2 (s_1 lista_2)^* f_1$$

$$lista_2 = i_2 lista_3 (s_2 lista_3)^* f_2$$

$$\ldots$$

$$lista_k = i_k e_k (s_k e_k)^* f_k$$

EXAMPLES

$$\text{livello 1: } begin \quad istr_1; istr_2; \ldots; istr_n \quad end$$

$$\text{livello 2: } STAMPA(\text{var}_1, \text{var}_2, \ldots, \text{var}_n)$$

$$3 + \underbrace{5 \times 7 \times 4}_{\text{monomio1}} - \underbrace{8 \times 2 \div 5}_{\text{monomio2}} + 8 + 3$$

padre, madre, figlio e figlia
un padre forte, severo e giusto, una madre amorevole e fedele

un libro come lista di capitoli separati da pagine bianche,
chiusa tra due copertine
un capitolo come lista di sezioni

Bibliography

- S. Crespi Reghizzi, *Linguaggi Formali e Compilazione*, Pitagora Editrice Bologna, 2006
- Hopcroft, Ullman, *Formal Languages and their Relation to Automata*, Addison Wesley, 1969
- A. Salomaa – *Formal Languages*, Academic Press, 1973
- D. Mandrioli, C. Ghezzi – *Theoretical Foundations of Computer Science*, John Wiley & Sons, 1987
- L. Breveglieri, S. Crespi Reghizzi, *Linguaggi Formali e Compilatori: Temi d'Esame Risolti,* web site (eng + ita)