2.8 On limitations of regular languages

For cardinality reasons there must be (many) non-regular languages: languages are uncountable, but regular expressions are countable.

Could we find a concrete, *interesting* example of a non-regular language? Easily.

The basic limitation of regular languages: finite state automata only have a limited "memory". Thus they cannot solve problems that require remembering arbitrarily large numbers.

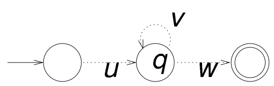
Example: the parenthesis language

$$L_{\text{match}} = \{(^k)^k \mid k \geq 0\}.$$

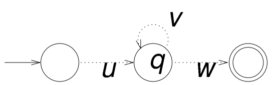
Formalization: "the pumping lemma".

Lemma 2.6 (The pumping lemma) Let A be a regular language. Then there exists an $n \geq 1$, such that every $x \in A$, $|x| \geq n$, can be (somehow) split into parts $x = uvw$ such that $|uv| \leq n$, $|v| \geq 1$, and $uv^iw \in A$ for all $i = 0, 1, 2, \ldots$

Proof. Let M be a deterministic finite state automaton that recognizes A, and let n be the number of states in M. Examine the sequence of states M goes through with input $x \in A$, $|x| \geq n$. Since M moves from one state to another with each input symbol, it must visit one state (at least) twice — in fact, already within the first n symbols of x. Let q be the first state visited twice.



Let u be the prefix of x the machine M has handled when it first enters q, v the part of x M handles before its first return to q and w the remaining part of x. Then we have $|uv| \leq n$, $|v| \geq 1$, ja $uv^iw \in A$ for all $i = 0, 1, 2, \ldots$ $\square$



Example. Examine the parenthesis language (denote '(' = a, ')' = b):

$$L = L_{\text{match}} = \{a^k b^k \mid k \geq 0\}.$$

Suppose that L were regular. Then according to the pumping lemma there should be some $n \geq 1$ so that all strings in L that are longer than n symbols can be pumped. We choose $x = a^n b^n$, so that $|x| = 2n > n$. According to the lemma x may be split as $x = uvw$, $|uv| \leq n$, $|v| \geq 1$; thus we must have

$$u = a^i,\ v = a^j,\ w = a^{n-(i+j)}b^n, \quad i \leq n-1,\ j \geq 1.$$

But for example by "pumping 0 times" we obtain

$$uv^0w = a^i a^{n-(i+j)} b^n = a^{n-j} b^n \notin L.$$

Thus L cannot be regular.

3. GRAMMARS AND PRODUCTION OF STRINGS

A grammar = a transformation system for producing strings ("words" of a language) from a certain initial string by repeatedly rewriting substrings according to given rules.

A grammar is *context-free* if in each rewriting step one so called *nonterminal symbol* may be replaced by some substituted string, and the substitution can always be carried out without regard for the surrounding structure of the string.

Applications: describing structured texts (e.g. BNF-syntax descriptions of programming languages, DTD/Schema definitions of XML), more generally describing structured "objects" (e.g. syntactic pattern recognition).

Context-free grammars can also describe (produce) certain non-regular languages.

Example: A context-free grammar for the language L_{match} (initial symbol S):

(i) $S \to \varepsilon$,
(ii) $S \to (S)$.

E.g. the derivation of the string ((())):

$$S \Rightarrow (S) \Rightarrow ((S)) \Rightarrow (((S))) \Rightarrow (((\varepsilon))) = ((())).$$

T–79.148 Introduction to Theoretical Computer Science

Another example: a simplified grammar for arithmetic expressions of a programming language similar to C:

$$\begin{array}{lcl} E & \to & T \mid E+T \\ T & \to & F \mid T*F \\ F & \to & a \mid (E). \end{array}$$

E.g. production of the expression $(a+a)*a$:

$$\begin{array}{llllll} \underline{E} & \Rightarrow \underline{T} & \Rightarrow & \underline{T}*F & \Rightarrow & \underline{F}*F \\ & \Rightarrow (\underline{E})*F & \Rightarrow & (\underline{E}+T)*F & \Rightarrow & (\underline{T}+T)*F \\ & \Rightarrow (\underline{F}+T)*F & \Rightarrow & (a+\underline{T})*F & \Rightarrow & (a+\underline{F})*F \\ & \Rightarrow (a+a)*\underline{F} & \Rightarrow & (a+a)*a. & & \end{array}$$

Harri Haanpää 2003–2004

T–79.148 Introduction to Theoretical Computer Science

Definition 3.1 A *context-free grammar* is a 4-tuple

$$G = (V, \Sigma, P, S),$$

where

- V is the alphabet of the grammar;
- $\Sigma \subseteq V$ is the set of *terminals* in the grammar; its complement $N = V - \Sigma$ is the set of *nonterminals*;
- $P \subseteq N \times V^*$ is the set of *rules* or *productions*;
- $S \in N$ is the *initial symbol*.

The production $(A, \omega) \in P$ is usually denoted by $A \to \omega$.

Harri Haanpää 2003–2004

T–79.148 Introduction to Theoretical Computer Science

The string $\gamma \in V^*$ *produces* or *derives directly* the string $\gamma' \in V^*$ in grammar G, denoted by

$$\gamma \underset{G}{\Rightarrow} \gamma'$$

if we may write $\gamma = \alpha A \beta$, $\gamma' = \alpha\omega\beta$ ($\alpha, \beta, \omega \in V^*$, $A \in N$), and G contains the production $A \to \omega$.

If the grammar G is clear from the context, we may write $\gamma \Rightarrow \gamma'$.

The string $\gamma \in V^*$ *produces* or *derives* the string $\gamma' \in V^*$ in grammar G, denoted by

$$\gamma \underset{G}{\Rightarrow}^* \gamma'$$

if there is a sequence $\gamma_0, \gamma_1, \ldots, \gamma_n$ ($n \geq 0$) of strings over V such that

$$\gamma = \gamma_0 \underset{G}{\Rightarrow} \gamma_1 \underset{G}{\Rightarrow} \cdots \underset{G}{\Rightarrow} \gamma_n = \gamma'.$$

As a special case $n = 0$ we have $\gamma \underset{G}{\Rightarrow}^* \gamma$ for all $\gamma \in V^*$. Again, if G is obvious from the context, we may write $\gamma \Rightarrow^* \gamma'$.

Harri Haanpää 2003–2004

A string $\gamma \in V^*$ is a *sentence derivation* of G, if $S \underset{G}{\Rightarrow}^* \gamma$.

A sentence derivation $x \in \Sigma^*$ of G that only contains terminal symbols is a *sentence* of G.

The language *produced* or *described* by G consists of the sentences of G:

$$L(G) = \{x \in \Sigma^* \mid S \underset{G}{\Rightarrow}^* x\}.$$

A formal language $L \subseteq \Sigma^*$ is *context-free*, if it can be produced by a context-free grammar.

For example the parenthesis language $L_{\text{match}} = \{(^k)^k \mid k \geq 0\}$ is produced by the grammar

$$G_{\text{match}} = (\{S, (,)\}, \{(,)\}, \{S \to \varepsilon, S \to (S)\}, S).$$

The language L_{expr} of simple arithmetic expressions is produced by the grammar

$$G_{\text{expr}} = (V, \Sigma, P, E),$$

where

$$\begin{aligned} V &= \{E, T, F, a, +, *, (,)\}, \\ \Sigma &= \{a, +, *, (,)\}, \\ P &= \{E \to T,\ E \to E + T,\ T \to F,\ T \to T * F, \\ &\quad\ F \to a,\ F \to (E)\}. \end{aligned}$$

Another grammar for producing L_{expr} is

$$G'_{\text{expr}} = (V, \Sigma, P, E),$$

where

$$\begin{aligned} V &= \{E, a, +, *, (,)\}, \\ \Sigma &= \{a, +, *, (,)\}, \\ P &= \{E \to E + E,\ E \to E * E,\ E \to a,\ E \to (E)\}. \end{aligned}$$

Note: Although G'_{expr} appears simpler than the grammar G_{expr}, it is unfortunately structurally ambiguous, which is often an undesirable property.

Established notation

Nonterminals: $A, B, C, \ldots, S, T$.

Terminals: letters $a, b, c, \ldots, s, t$;
numbers $0, 1, \ldots, 9$;
special signs; bolded or underlined reserved words (**if**, **for**, **end**, $\ldots$).

Arbitrary symbols (when no distinction is made between terminals and nonterminals): X, Y, Z.

Strings of terminals: u, v, w, x, y, z.

Mixed strings: $\alpha, \beta, \gamma, \ldots, \omega$.

Productions with a common left side A may be written together: instead of

$$A \to \omega_1,\ A \to \omega_2,\ \dots A \to \omega_k$$

we write

$$A \to \omega_1 \mid \omega_2 \mid \dots \mid \omega_k.$$

A grammar is ofter presented simply as a collection of rules

$$\begin{array}{lcccccc} A_1 & \to & \omega_{11} & \mid & \dots & \mid & \omega_{1k_1} \\ A_2 & \to & \omega_{21} & \mid & \dots & \mid & \omega_{2k_2} \\ \vdots & & & & & & \\ A_m & \to & \omega_{m1} & \mid & \dots & \mid & \omega_{mk_m}. \end{array}$$

Then nonterminals are determined by the notation conventions above or by observing that they appear on the left side of productions; other symbols are terminals. The *initial symbol* is the nonterminal that appears as *the left side of the first rule*; in this case A_1.

Some constructions

Let $L(T)$ be the set of strings that can be derived from the nonterminal T. Let us have a collection P of productions, where the nonterminal A does not appear and for which $L(B)$ can be derived from B and similarly $L(C)$ from C.

By adding one of the following productions to P we obtain new languages:

production	language
$A \to B \mid C$	union $L(A) = L(B) \cup L(C)$
$A \to BC$	catenation $L(A) = L(B)L(C)$, and
$A \to AB \mid \epsilon$ (left recursion) or $A \to BA \mid \epsilon$ (right recursion)	Kleene closure $L(A) = L(B)^*$

Central embedding is a construction typical of context-free grammars; it often (but not always) makes a language non-regular. By adding the production

$A \to BAC \mid \epsilon$ we obtain

$$L(A) = \bigcup_{i=0}^{\infty} L(B)^i L(C)^i.$$

3.2 Regular languages and context-free grammars

We have seen that context-free grammars can describe some non-regular languages such as L_{match} and L_{expr}. We show that all regular languages can also be described by context-free grammars. Thus context-free languages are a proper superclass of regular languages.

A context-free grammar is *right linear*, if all its productions are of the form $A \to aB$ or $A \to \varepsilon$, and *left linear*, if all its productions are of the form $A \to Ba$ or $A \to \varepsilon$.

It turns out that all regular languages and no others can be produced by left or right linear grammars. Therefore such grammar are also called *regular*. We prove this only for right linear grammars.

Theorem 3.1 Every regular language can be produced by a right linear grammar.

Proof. Let L be a regular language over the alphabet Σ and let $M = (Q, \Sigma, \delta, q_0, F)$ be the (deterministic or nondeterministic) finite state automaton that recognizes L. We design a grammar G_M, for which $L(G_M) = L(M) = L$.

The set of terminals of G_M is the same as the input alphabet of M, and the set of nonterminals will have one symbol A_q for each state q in M. The initial symbol of the grammar is A_{q_0}, and its productions represent the transitions of M:

(i) for each final state $q \in F$ of M the production $A_q \to \varepsilon$ is included;

(ii) for each transition $q \xrightarrow{a} q'$ in M (that is $q' \in \delta(q, a)$) the production $A_q \to aA_{q'}$ is included.

To verify the correctness of the construction we denote the set of strings of terminals that can be derived from A_q by

$$L(A_q) = \{x \in \Sigma^* \mid A_q \underset{G_M}{\Rightarrow}{}^* x\}.$$

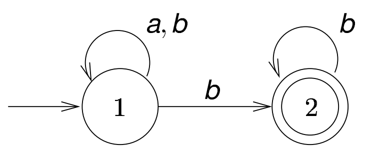
By induction over the length of x we may show that for all q we have

$$x \in L(A_q) \text{ iff } (q, x) \underset{M}{\vdash}{}^* (q_f, \varepsilon) \text{ for some } q_f \in F.$$

In particular we have

$$\begin{aligned} L(G_M) = L(A_{q_0}) &= \{x \in \Sigma^* \mid (q_0, x) \underset{M}{\vdash}{}^* (q_f, \varepsilon) \\ &\quad \text{for some } q_f \in F\} \\ &= L(M) = L. \quad \Box \end{aligned}$$

Example. Automaton:



The corresponding grammar:

$$\begin{aligned} A_1 &\to aA_1 \mid bA_1 \mid bA_2 \\ A_2 &\to \varepsilon \mid bA_2. \end{aligned}$$

Theorem 3.2 Every language produced by a right linear grammar is regular.

Proof. Let $G = (V, \Sigma, P, S)$ be a right linear grammar. Construct a nondeterministic finite state automaton $M_G = (Q, \Sigma, \delta, q_S, F)$ that recognizes $L(G)$ as follows:

The states of M_G represent the nonterminals of G:

$$Q = \{q_A \mid A \in V - \Sigma\}.$$

The initial state of M_G is the state q_S that represents the initial symbol S.

The input alphabet of M_G is the set Σ of terminals of G.

The transition function δ of M_G:n emulates the productions of G so that for every production $A \to aB$ the automaton has the transition $q_A \xrightarrow{a} q_B$ (i.e., $q_B \in \delta(q_A, a)$).

The final states of M_G are those state whose corresponding nonterminals in G have an ε-production:

$$F = \{q_A \in Q \mid A \rightarrow \varepsilon \in P\}.$$

The correctness of the construction can again be verified by induction on the length of strings produced by G and accepted by M_G. $\square$