Aalto University
School of Science

CS-C2160 Theory of Computation

Lecture 5: Limitations of Regular Languages; Context-Free Grammars and Languages

Pekka Orponen
Aalto University
Department of Computer Science

Spring 2021

Topics

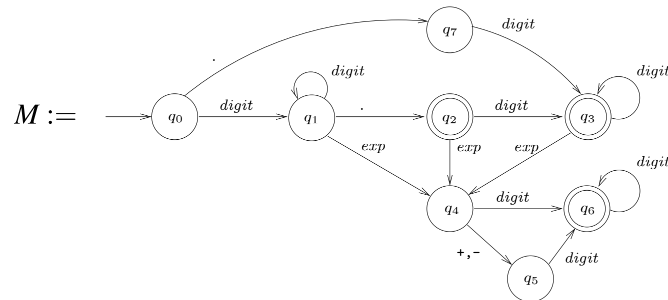
- Limitations of regular languages
  - The pumping lemma
- Context-free grammars and languages
  - Generating strings with grammars
  - Context-free grammars
  - Some useful constructions
- Connection to regular languages
  - Right- and left-linear grammars

Material:

- In Finnish: Sections 2.8 and 3.1–3.2 in Finnish lecture notes
- In English: Sections 1.4 and 2.1 (up to "ambiguity") in the Sipser book, and right/left-linear grammars on these slides

Recap: Regular languages

$M :=$



$\downarrow$ recognises $\mathcal{L}(M)$

$\{.256, 1., 3.14, 2.3E{-}10, \ldots\}$

$\uparrow$ describes $\mathcal{L}(r)$

$$r := (dd^*.d^* \cup .dd^*)(e(+\cup-\cup\varepsilon)dd^* \cup \varepsilon) \cup (dd^*e(+\cup-\cup\varepsilon)dd^*)$$

Limitations of Regular Languages

4.3 The pumping lemma

- Due to cardinality reasons, there must be a large number of languages that are not regular: there are uncountably many languages but only countably many regular expressions.[1]
- Can we find a concrete, interesting example of a language that is not regular? Yes, easily.
- A fundamental limitation is that finite automata only have a bounded amount of "memory" (= the states). Therefore, they cannot solve problems that require "remembering" arbitrarily large numbers.
- For instance, the "parentheses language"

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

is not regular.
- Such results can be established precisely using the "pumping lemma for regular languages".

[1]We shall discuss countable and uncountable sets in Lecture 9.

Lemma 4.3 (Pumping lemma for regular languages)

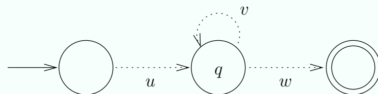
Let A be a regular language. Then there is a number $p \geq 1$ such that every $x \in A$ with $|x| \geq p$ can be divided in three parts, $x = uvw$, satisfying the conditions (i) $|uv| \leq p$, (ii) $|v| \geq 1$, and (iii) $uv^iw \in A$ for all $i = 0, 1, 2, \ldots$

Proof

Let M be a deterministic finite automaton that recognises A, i.e. has $\mathcal{L}(M) = A$. Let p be the number of states in M. Study the sequence of states visited when M is run on any input $x \in A$ with $|x| \geq p$. The initial state is visited first and then $|x|$ other states, one for each symbol in x. Thus at least $p+1$ states are visited, and since M has only p states, some state(s) are visited more than once. In fact, the revisit to some state happens already during the first p symbols of x. Let q be the first revisited state.

Let now:

- u be the prefix of x already processed by M's first visit to q,
- v the substring of x processed next before M's first revisit to q,
- w the remaining suffix of x.



Obviously $|uv| \leq p$ and $|v| \geq 1$. Consider what happens on any string of the form uv^iw, with $i \geq 0$: (i) M processes u similarly as when accepting $x = uvw$, and enters q, (ii) M processes v^i by executing the same loop as when accepting $x = uvw$, but now i times instead of once, and again enters q, (iii) processes w and enters the same accepting state as when accepting $x = uvw$. Thus M accepts uv^iw as well, and so $uv^iw \in \mathcal{L}(M) = A$.

Example:

Consider the language of balanced parentheses (for clarity, replace '(' = a and ')' = b):

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

Suppose (towards an eventual contradiction) that L were regular.
In that case, by the pumping lemma, there should be some $p \geq 1$ so that all the strings in L that have at least p symbols can be "pumped".
Choose one such string, say $x = a^p b^p$. Now $x \in L$ and $|x| = 2p > p$.
The lemma says that x can be divided in three parts, $x = uvw$, so that $|uv| \leq p$, $|v| \geq 1$ and $uv^iw \in L$ for all $i = 0, 1, 2, \ldots$. In detail:
$u = a^m$, $v = a^n$, $w = a^{p-(m+n)} b^p$ for some m, n s.t. $m + n \leq p$ and $n \geq 1$.
But when "pumping" the middle part zero times we get

$$uv^0w = a^m a^{p-(m+n)} b^p = \mathbf{a^{p-n} b^p} \notin \mathbf{L} \text{ as } n \geq 1.$$

From the contradiction, we conclude that the assumption must be wrong and in fact L is not regular.

Note

- There are also non-regular languages that can be "pumped".
  ☞ Lemma 4.3 can **not** be used to show that a language **is** regular.
- As an example, the language

$$L = \{c^r a^k b^k \mid r \geq 1, k \geq 0\} \cup \{a^k b^l \mid k, l \geq 0\}$$

  is *not* regular, even though the strings in it can be "pumped".
- Brainteaser: Show that (i) L satisfies the conditions of Lemma 4.3, (ii) L is not regular.

Context-Free Grammars and Languages

5.1 Generating strings with grammars

- A grammar = a system for generating strings from an initial string by repeatedly substituting substrings with others according to some rules.
- A grammar is *context-free* if (i) each substitution replaces only one symbol (variable) with some new substring and (ii) this substitution can always be done independently of the context, i.e., of the other symbols around the substituted variable.
- Applications: Description of structural text (e.g., BNF descriptions for programming languages, DTD/Schema descriptions in XML) and other structural data (e.g. structural pattern recognition, structured data mining).
- Some examples of describing the syntax of programmig languages:
  - grammar for Python 3
  - grammar for Java SE7

- Context-free grammars can generate (i.e., describe) languages that are not regular.

Example:

A context-free grammar for language L_{match} (S is the start variable):

$$\begin{aligned} S &\rightarrow \varepsilon \\ S &\rightarrow (S) \end{aligned}$$

Generating string $((()))$ according to this grammar:

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

Example:

A (simplified) grammar for generating arithmetic expressions in a programming language resembling C:

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

Generating string $(a+a)*a$ according to this grammar:

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

5.2 Context-free grammars

Definition 5.1

A *context-free grammar* is a tuple

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

where

- V a finite set of symbols,
- $\Sigma \subseteq V$ is the set of *terminals* and its complement $N = V - \Sigma$ is the set of *variables* (also called non-terminals),
- $P \subseteq N \times V^*$ is the finite set of *rules* (also called productions), and
- $S \in N$ is the *start symbol* of the grammar.

- A rule $(A, \omega) \in P$ is usually written as $A \to \omega$.

Note

The Sipser book has a slightly different, but equivalent, definition.

- Let G be a context-free grammar.
- A string $\gamma \in V^*$ *yields* (or "derives in a single step") a string $\gamma' \in V^*$, denoted by
  $$\gamma \underset{G}{\Rightarrow} \gamma'$$
  if
  - the grammar G contains a rule $A \to \omega$ such that
  - $\gamma = \alpha A \beta$ and $\gamma' = \alpha\omega\beta$ for some $\alpha, \beta \in V^*$.
- A string $\gamma \in V^*$ *derives* a string $\gamma' \in V^*$, denoted by
  $$\gamma \underset{G}{\Rightarrow}^* \gamma'$$
  if there is a finite sequence of strings $\gamma_0, \gamma_1, \ldots, \gamma_n$ over V $(n \geq 0)$ such that
  $$\gamma = \gamma_0, \quad \gamma_0 \underset{G}{\Rightarrow} \gamma_1 \underset{G}{\Rightarrow} \ldots \underset{G}{\Rightarrow} \gamma_n, \quad \text{and} \quad \gamma_n = \gamma'.$$
- As the special case $n = 0$, relation $\gamma \underset{G}{\Rightarrow}^* \gamma$ holds for any $\gamma \in V^*$.
- If the grammar G is clear from the context, we simply write $\gamma \Rightarrow \gamma'$ and $\gamma \Rightarrow^* \gamma'$ instead of $\gamma \underset{G}{\Rightarrow} \gamma'$ and $\gamma \underset{G}{\Rightarrow}^* \gamma'$, respectively.

- A grammar G *generates* (or *describes*) the language
  $$\mathcal{L}(G) = \{x \in \Sigma^* \mid S \underset{G}{\Rightarrow}^* x\}.$$
- That is, the language generated by a grammar consists of all the variable-free strings that can be derived from the start variable. These terminal strings are also called the *words* of the language.

Definition 5.2

A language $L \subseteq \Sigma^*$ is *context-free* if it can be generated by some context-free grammar.

Example:

The "parentheses language" $L_{\text{match}} = \{(^k)^k \mid k \geq 0\}$ is generated by the grammar:

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

Example:

Our earlier grammar for generating simple arithmetic expressions is formally:

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

where

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

Example:

Another grammar for generating simple arithmetic expressions 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

Even though the grammar G'_{expr} looks simpler than the grammar G_{expr}, it has the disadvantage of being *ambiguous*, which is usually an unwanted structural feature. (More on this topic in Lecture 6.)

Some notational conventions

- Variables: $A,B,C,\ldots,S,T$.
- Terminal symbols:
  - letters $a,b,c,\ldots,s,t$
  - digits $0,1,\ldots,9$
  - special symbols such as parentheses
  - boldface or underlined atomic strings or "reserved names": **if**, **for**, **end**, ...
- Symbols in general (when one does not want to distinguish terminals and variables): X,Y,Z.
- Variable-free (i.e., only terminal symbol) strings u,v,w,x,y,z.
- Strings in general (i.e., mixed terminal/variable): $\alpha,\beta,\gamma,\ldots,\omega$.

- Rules that have the same symbol A on the left-hand side,

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

can be presented together as:

$$A \to \omega_1 \mid \omega_2 \mid \ldots \mid \omega_k$$

- A grammar is usually represented simply as a set of rules:

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

In this case, the variables are deduced by the above conventions or by the fact that they appear as the left-hand sides of the rules; other symbols are then terminal symbols. The start symbol is the left-hand side of the *first rule*, e.g. A_1 in the above rule set.

5.3 Some useful constructions

- Let $\mathcal{L}(T)$ be the set of terminal strings that can be derived from a variable T (i.e., "the language generated by T").
- Let P be a set of rules which (i) does not contain a variable A, and (ii) in which one can generate $\mathcal{L}(B)$ from variable B and $\mathcal{L}(C)$ from variable C.
- One can obtain new languages by adding new rules to P:

rule	language
$A \to B \mid C$	union $\mathcal{L}(A) = \mathcal{L}(B) \cup \mathcal{L}(C)$
$A \to BC$	concatenation $\mathcal{L}(A) = \mathcal{L}(B)\mathcal{L}(C)$
$A \to AB \mid \varepsilon$ (left recursion) or $A \to BA \mid \varepsilon$ (right recursion)	Kleene star $\mathcal{L}(A) = (\mathcal{L}(B))^*$

- Using these constructions it is easy to prove the following:

Theorem 5.1

The class of context-free languages is closed under union, concatenation and star operations. In other words,, if the languages $L_1, L_2 \subseteq \Sigma^\star$ are context-free, then so are $L_1 \cup L_2$, $L_1 L_2$ and $L_1^\star$.

- *Centre embedding* is a characteristic context-free grammar construction:

  Adding rule

$$A \to BAC \mid \varepsilon$$

  results in the language

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

- Grammars containing this construction usually — but not always — generate non-regular languages (cf. grammar G_{match}).

Connection to Regular Languages

5.4 Right- and left-linear grammars

- Context-free grammars can generate non-regular languages, e.g. the languages L_{match} and L_{expr}.
- We next show that also every regular language can be generated by some context-free grammar. Thus, context-free languages are a proper super-class of regular languages.
- We say that a context-free grammar is
  - *right-linear* if all the rules in it are of form $A \to aB$ or $A \to \varepsilon$, and
  - *left-linear* if all the rules in it are of form $A \to Ba$ or $A \to \varepsilon$.
- It turns out that both right- and left-linear grammars generate exactly the class of regular languages. (Hence they are also known as *regular grammars*.)
- In the following, we prove this for right-linear grammars.

Theorem 5.2

Each regular language can be generated by some right-linear context-free grammar.

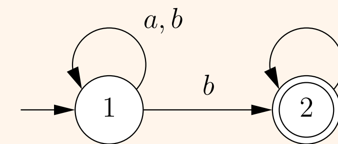
Proof

Let L be a regular language over an alphabet Σ and let $M = (Q, \Sigma, \delta, q_0, F)$ be a (deterministic or nondeterministic) finite automaton that recognises it. We construct a right-linear context-free grammar G_M such that $\mathcal{L}(G_M) = \mathcal{L}(M) = L$.
The set of terminal symbols of G_M is Σ and its variable set contains a variable A_q for each state q of M. The start variable of the grammar is A_{q_0} and its rules correspond to the transitions of M:

- For each accept state $q \in F$ of M there is a rule $A_q \to \varepsilon$.
- For each transition $q \xrightarrow{a} q'$ in M (that is, $q' \in \delta(q, a)$) there is a rule $A_q \to aA_{q'}$.

Example:

Automaton:



The corresponding grammar:

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

For establishing the correctness of the construction, let us again denote the set of terminal symbol strings that can be derived from a variable A_q by

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

By induction on the length of a string x one can then show that for all states q it holds that:

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

In particular,

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

Theorem 5.3

If a language can be generated by a right-linear context-free grammar, then it is regular.

Proof

Let $G = (V, \Sigma, P, S)$ be a right-linear context-free grammar. We construct a nondeterministic finite automaton $M_G = (Q, \Sigma, \delta, q_S, F)$ that recognises $\mathcal{L}(G)$ as follows:

- The states of M_G correspond to the variables of G:

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

- The initial state of M_G is the state q_S corresponding to the start variable of G.
- The alphabet of M_G is the set Σ of terminal symbols in G.

- The transition function δ of M_G simulates the rules of G so that for each rule $A \to aB$, the automaton has the transition $q_A \xrightarrow{a} q_B$ (that is, $q_B \in \delta(q_A, a)$).
- The accept states of M_G are the ones that correspond to variables having an ε-rule in G:

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

By induction on the length of a string x one can again show that for all variables A it holds that:

$$x \in \mathcal{L}(A) \text{ iff } (q_A, x) \vdash_M^* (q_f, \varepsilon) \text{ for some } q_f \in F.$$

In particular,

$$\begin{aligned} \mathcal{L}(M_G) = \mathcal{L}(q_S) &= \{x \in \Sigma^* \mid (q_S, x) \vdash_M^* (q_f, \varepsilon) \text{ for some } q_f \in F\} \\ &= \mathcal{L}(S) = L. \end{aligned}$$