Lec 08. More on regular language & Context-free grammar

Eunjung Kim

When L is regular, so is $Prefix(L)$?

Given two strings $x, w \in \Sigma^*$, x is a prefix of w if $w = xy$ for some $y \in \Sigma^*$. For a language $L \subseteq \Sigma^*$, let $Prefix(L) = \{x \in \Sigma^* : x \text{ is a prefix of } w \in L\}$.

If L is regular, $Prefix(L)$ is regular

Let M be an DFA with $L = L(M)$. Notice that

$w \in L$ can be written as $w = xy$ if and only if $\hat{\delta}(q_0, x) = q$ for some state $q \in Q$ such that ....

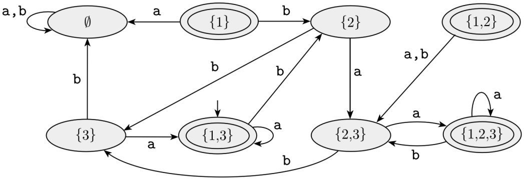

Figure 1.43, Sipser 2012

When L is regular, so is $Prefix(L)$?

Let M be an DFA with $L = L(M)$. Then

$$Prefix(L) = \bigcup_{q \in Q \text{ such that....}} L_q.$$

where $L_q = \{x \in \Sigma^* : \hat{\delta}(q_0, x) = q\}$.

- If L_q is regular, then $Prefix(L)$ is regular (why?)
- Is L_q regular?
- $properPrefix(L)$ be the set of all proper prefixes of some $w \in L$; x is a proper prefix of w if $w = xy$ for some $y \in \Sigma^+$.
- Is $properPrefix(L)$ regular?

When L is regular, so is $Suffix(L)$?

Given two strings $x, w \in \Sigma^*$, x is a suffix of w if $w = yx$ for some $y \in \Sigma^*$.
For a language $L \subseteq \Sigma^*$, let $Suffix(L) = \{x \in \Sigma^* : x \text{ is a suffix of } w \in L\}$.

If L is regular, $Suffix(L)$ is regular

Let *reverse*(L) be the set of all strings each of which is a reversal w^R of some string $w \in$L.

- If L is regular, *reverse*(L) is regular as well; homework.
- $Suffix(L)$ can be obtained by applying ???? and ???? operations on L.

QUOTIENT L/a FOR $a \in \Sigma$

Given a language L over Σ and a symbol $a \in \Sigma$, the quotient of L by a denoted as L/a is the language

$$\{x \in \Sigma^* : xa \in L\}.$$

Is L/a regular?

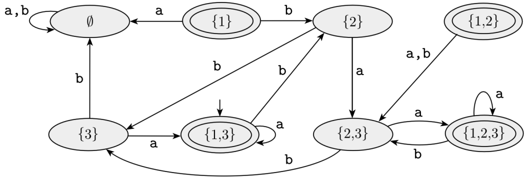

Figure 1.43, Sipser 2012

- For a state $q \in Q$, if $x \in L_q$ satisfies $xa \in L$ for some x, then for all $y \in L_q$ we have $ya \in L$.
- That is, $L_q \subseteq L/a$ or $L_q \cap L/a = \emptyset$.
- How to tell if $L_q \subseteq L/a$?

The language $a \setminus L$ for $a \in \Sigma$

Given a language L over Σ and a symbol $a \in \Sigma$, the language $a \setminus L$ is defined as

$$\{x \in \Sigma^* : ax \in L\}.$$

Is $a \setminus L$ regular?

Idea: Express $a \setminus L$ using the operations we examined so far to immediately conclude.

More exotic language P_s

- Fix a DFA M and a state $s \in Q$.
- Let P_s be the set of all string $w \in L$ such that the accepting computation history of w visits the state s.
- Is P_s regular?

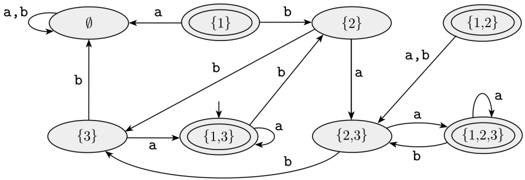

Figure 1.43, Sipser 2012

More exotic language P_s

First approach.

- For any string w, $w \in P_s$ if and only if it can be written as $w = xy$ with $\hat{\delta}(q_0, x) = s$ and $\hat{\delta}(s, y) \in F$.
- That is $P_s = L_s \cdot A_s$, where L_q and A_q are defined for all $q \in Q$ as

$$L_q = \{x \in \Sigma^* : \hat{\delta}(q_0, x) = q\}.$$

$$A_q = \{x \in \Sigma^* : \hat{\delta}(q, x) \in F\}.$$

- Is any one of L_q and A_q regular?

More exotic language P_s

Second approach: use Myhill-Nerode Theorem.

Myhill-Nerode Theorem

L is regular if and only if the number of equivalence classes of $\equiv_L$ is finite.

Idea: use the DFA M recognizing L to identify the equivalence relation $\equiv_{P_s}$, (or a refinement of it) of finite index.

- For $T \subseteq Q$ and $q \in T$, let $L_{T,q}$ be the set of all strings w such that the computation history of w on M visits precisely the states in T and end in q.
- $\Sigma^* = \dot{\bigcup}_{T \subseteq Q, q \in T} L_{T,q}$ (disjoint union).
- We want to argue that any strings $x, y \in L_{T,q}$ are indistinguishable by P_s.
- This shows that P_s has finitely many equivalent classes, thus regular.

More exotic language P_s

Second approach: use Myhill-Nerode Theorem

Myhill-Nerode Theorem, in action

P_s is regular if for any $T \subseteq Q$ and $q \in T$,

- any $x, y \in L_{T,q}$ are indistinguishable by P_s, or equivalently
- for any $x, y \in L_{T,q}$ and for any $z \in \Sigma^*$, $xz \in P_s$ if and only if $yz \in P_s$.

What are the key property of z which will make $xz \in P_s$ (or not) for $x \in L_{T,q}$?

More exotic language P_s

Second approach: use Myhill-Nerode Theorem

Myhill-Nerode Theorem, in action

P_s is regular if for any $T \subseteq Q$ and $q \in T$,

- any $x, y \in L_{T,q}$ are indistinguishable by P_s, or equivalently
- for any $x, y \in L_{T,q}$ and for any $z \in \Sigma^*$, $xz \in P_s$ if and only if $yz \in P_s$.

What are the key property of z which will make $xz \in P_s$ (or not) for $x \in L_{T,q}$?

1. whether $\hat{\delta}(q, z) \in F$ or not: this determines whether $xz \in L$ or not.
2. whether $s \in T$ or not.
3. whether s is visited during the computation history of $\hat{\delta}(q, z)$ or not.

A bit more exotic language

Fix a DFA M. The set of all strings w such that the (accepting) computation history of w visits all the state of M, is it regular?

Even More exotic language

- Why do we care about the second approach using Myhill-Nerode theorem when the first approach seems much simpler?

- Even more exotic language. Fix two states s_1, s_2 of a DFA M. Let P_{s_1,s_2} be the set of strings $w \in L$ whose computation history visits each of s_1, s_2 exactly once.

- Is P_{s_1,s_2} regular?

What we learned so far

- Finite (state) automata: a machine with limited memory.
- Nondeterministic FA has the extra feature of making multiple transitions in parallel and ϵ-transition. Conversions between DFA and NFA possible (no added power).
- Regular expression: describes the 'shape' of a regular language directly.
- Conversion between regular expression and NFA using Generalized NFA.
- The class of regular languages is closed under various operations.
- Pumping lemma as a tool to prove that a language is nonregular.
- Myhill-Nerode Theorem as a powerful characterization of regular languages.
- Büchi's Theorem as another characterization of regular language using logic.
- One can prove various properties of NFA/DFA and regular language combining the tools we learned.

Context-Free Language

Expressing palindromes

- A string w is a palindrome if and only if $w = w^R$.
- The set of palindromes (e.g. over $\{0, 1\}$) is not regular, so one cannot use a regular expression or NFA to describe the language.
- Recursive definition:
  1. Base case: ϵ, 0 and 1 are palindromes.
  2. Induction: if w is a palindrome, then $0w0$ and $1w1$ are palindromes.
- Any word generated in this way is a palindrome.
- Conversely, any palindrome can be generated in this way: a word of the form $x \cdot w \cdot y$ with $x, y \in \Sigma$ is a palindrome (if and) only if $x = y$ and w is a palindrome.

Expressing palindromes

- Recursive definition:
  1. Base case: ϵ, 0 and 1 are palindromes.
  2. Induction: if w is a palindrome, then $0w0$ and $1w1$ are palindromes.

- Definition by rules.
  1. $S \to \epsilon$.
  2. $S \to 0$.
  3. $S \to 1$.
  4. $S \to 0S0$.
  5. $S \to 1S1$.

- Any word generated using the above rules is a palindrome.

- Conversely, any palindrome can be generated using the above rules.

Context-free grammars (CFG)

Definition of Context-free grammar

There are four component of CFG $G = (V, \Sigma, R, S)$.

1. A finite set of nonterminals, often called the variables and denoted by V.
2. A finite set of terminals (equivalently, alphabet) Σ.
3. A finite set of rules R (often called substitution rules/ production rules) in the form

   $$X \to \gamma,$$

   where
   - X is a variable; $X \in V$.
   - γ is a string of terminal and nonterminal symbols; $\gamma \in (\Sigma \cup V)^*$.
4. A unique nonterminal symbol, often denoted as S, called the start symbol.

A quadruple $G = (V, \Sigma, R, S)$ is a context-free grammar (CFG) if the four components are as above.

Expressing palindromes

- Consider the grammar $G_{pal} = (\{S,\}, \{0,1\}, R, S)$, where R is the five production rules below.
  1. $S \to \epsilon$.
  2. $S \to 0$.
  3. $S \to 1$.
  4. $S \to 0S0$.
  5. $S \to 1S1$.
  6. (or equivalently, we write all the bodies of rules sharing the same head) $S \to \epsilon \mid 0 \mid 1 \mid 0S0 \mid 1S1$.
- Observe: a word over $\{0,1\}$ is a palindrome if and only if it can be derived from S, that is, by a recursively substituting a variable using one of the substitution rules.
- In other words, the language of palindromes over $\{0,1\}$ is precisely the language of the grammar G_{pal}, denoted as $L(G_{pal})$.

Expressing MSO-formulae on strings using "rules"

Recall that we defined MSO-formula on Σ-strings as a well-formed strings such that ....

- Definition by rules.
  1. $\varphi \to x = x \mid x \in X \mid x < x \mid P_a(x)$ (for each $a \in \Sigma$).
  2. $\varphi \to (\varphi)$.
  3. $\varphi \to \varphi \wedge \varphi \mid \varphi \vee \varphi$.
  4. $\varphi \to \exists x\ \varphi \mid \forall x\ \varphi \mid \exists X\ \varphi \mid \forall X\ \varphi$
  5. $x \to x_1 \mid x_2 \mid x_3 \mid \cdots$.
  6. $X \to X_1 \mid X_2 \mid X_3 \mid \cdots$.
- Any string generated using the above rules is an MSO-formula.
- Conversely, any MSO-formula can be generated using the above rules.

Derivation

Consider a CFG $G = (V, \Sigma, R, S)$, $u, v, w \in (\Sigma \cup V)^*$ (a string of terminals and nonterminals) and a variable (nonterminal) $A \in V$.

Yield, derive, derivation

- We say that uAv yields uwv, written $uAv \Rightarrow_G uwv$, if G has the rule $A \rightarrow w$; put another way, uwv is obtained by substituting a variable in the string uAv by the body of a rule whose head is the said variable.
- We say that u derives v, written $u \Rightarrow_G^* v$. if $u = v$ or there is a sequence $u_1, \ldots, u_k$ for some $k \geq 1$ such that

$$u \Rightarrow_G u_1 \Rightarrow_G \cdots \Rightarrow_G u_k \Rightarrow_G v$$

  and the sequence is called a derivation.

We omit the subscript G in $\rightarrow_G$ and $\Rightarrow_G$ if the CFG under consideration is clear in the context.

Derivation by example

We want to describe, as CFG,

- all arithmetic expressions with $+$ and $\times$
- over the variables of the form $(a \cup b)(a \cup b \cup 0 \cup 1)^*$.

Consider the following CFG $G_{ari} = (\{E, I\}, \{a, b, 0, 1, +, \times, (,)\}, R, E)$, where R consists of the following rules

1. $E \to I$
2. $E \to E + E$
3. $E \to E \times E$
4. $E \to (E)$
5. $I \to a$
6. $I \to b$
7. $I \to Ia$
8. $I \to Ib$
9. $I \to I0$
10. $I \to I1$

Context-free language

For a CFG $G = (V, \Sigma, R, S)$, the language of G, denoted by $L(G)$ is the set of all strings consisting of terminals (only) that have derivations from the start symbol, i.e.

$$L(G) = \{w \in \Sigma^* \mid S \Rightarrow_G^* w\}.$$

A language is said to be context-free if it is the language of a context-free grammar. A context-free languages is often abbreviated as CFL.

Some remarks on CFG

- In general, the rule of a grammar has the form $u \to v$ with both u and v are strings of terminals and nonterminals.
- A grammar is context-free if the head u is a nonterminal (variable) in all the rules; we do not need to consider the context.
- Different restrictions on the grammar define the hierarchy of formal languages.

Class	Languages	Automaton	Rules	Word Problem	Example
type-0	recursively enumerable	Turing machine	no restriction	undecidable	Post's corresp. problem
type-1	context sensitive	linear-bounded TM	$\alpha \to \gamma$ $\lvert\alpha\rvert \leq \lvert\gamma\rvert$	PSPACE-complete	$a^n b^n c^n$
type-2	context free	pushdown automaton	$A \to \gamma$	cubic	$a^n b^n$
type-3	regular	NFA / DFA	$A \to a$ or $A \to aB$	linear time	$a^* b^*$

Figure 1: Chomsky Hierarchy

Figure 1, Lecture note on 15-411: Compiler Design, CMU, 2023