UMC205: Automata and Computability

Naman Mishra

January 2024

Contents

Chapter 0

The Course

Lecture 01: Tue 02 Jan '24

Instructor: Prof. Deepak D'Souza
Lecture hours: Tuesdays and Thusdays 11:30–12:50
Course Website: https://www.csa.iisc.ac.in/~deepakd/atc-2024/

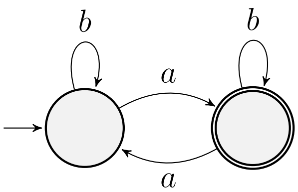
Grading

- Mid-semester Exam: 20%
- Assignments + Quizzes + Seminar: 40%
- End-semester Exam: 40%

Course Overview

We will be studying different kinds of "automata" or "state machines".

A *finite state automata* which accepts the language with an odd number of a's



A language is *regular* if it is accepted by a finite state automata. It may be deterministic or non-deterministic, the set of languages accepted by both are the same.

A *pushdown automata* is a finite state automata with a stack.

A *context-free grammar* is a language that is accepted by a pushdown automata.

Chapter 1

Languages

Lecture 02: Thu 04 Jan '24

Definition 1.1. An *alphabet* is a non-empty finite set of symbols or "letters".

A *string* or *word* over an alphabet A is a finite sequence of letters from A. Equivalently, a string is a map from a prefix (possibly empty) of $\mathbb{N}$ to A. The length of a string s, notated $\#s$, is the cardinality of its domain. The empty string is denoted ϵ.

The set of all strings over A is denoted A^*.

Example. $A = \{a, b, c\}$ and $\Sigma = \{0, 1\}$ are both alphabets. $aaba$ is a string over $A = \{a, b, c\}$.

Proposition 1.2. Let A be an alphabet. Then A^* is countably infinite.

Proof. Let $n = \#A$. Let $f : A \to \{1, \ldots, n\}$ be a numbering of A. Replacing each letter in a string with its number gives a representation of the string as a natural number in base $n + 1$. This gives an injection $A^* \to \mathbb{N}$ and so A^* is countable. Infiniteness is obvious.

Alternatively, consider the strings in their Lexicographic order. $\square$

Definition 1.3 (Language)**.** A *language* over an alphabet A is a subset of A^*.

Example. Let $A = \{a, b, c\}$. Then $\{abc, aaba\}$, $\{\epsilon, b, aa, bb, aab, aba, bbb, \ldots\}$, $\{\epsilon\}$, $\{\}$ are all languages over A.

Definition 1.4 (Concatenation)**.** Let u, v be strings over an alphabet A. Then $u \cdot v$ or simply uv is the string obtained by appending v to the end of u.

For two languages L_1, L_2 over A, define their concatenation

$$L_1 \cdot L_2 \coloneqq \{uv \mid u \in L_1, v \in L_2\}.$$

We will also write ua where u is a string and a is a letter to mean u concatenated with the string of length 1 consisting of the letter a.

Definition 1.5 (Lexicographic order)**.** Let $(A, <)$ be a totally ordered alphabet. We say $u < v$ for $u, v \in A^*$ if either $\#u < \#v$ or $\#u = \#v$ and $u = pxu'$, $v = pyv'$ for some $p, u', v' \in A^*$ and $x, y \in A$ with $x < y$.

This is called the *lexicographic order* on A^*.

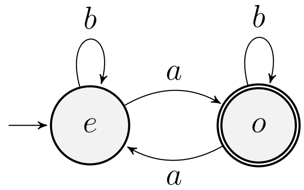
Proposition 1.6. Let A be an alphabet. Then the set of all languages over A is uncountable.

Proof. Diagonilization. Let $\Phi\colon A^* \to 2^{A^*}$ be a map. Then define $L_\Phi = \{s \in A^* \mid s \notin \Phi(s)\}$. Then L_Φ is a language over A that is not in the image of Φ. □

Definition 1.7 (Concatenations)**.** Let L_1, L_2 be languages over an alphabet A. Then $L_1 \cdot L_2$ is the language $\{uv \mid u \in L_1, v \in L_2\}$.

Chapter 2

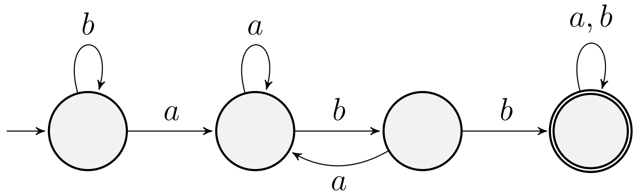
Finite-State Automata



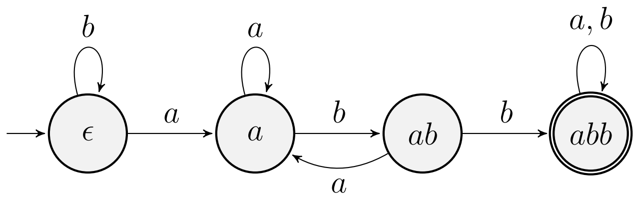
This is a deterministic finite state automaton. The machine starts at state e, and as it reads a (finite) string consiting of a's and b's, it moves to the state specified by the arrows from the current state. The state o is an "accepting state". Each string whose evaluation ends at state o is said to be accepted by the automaton.

Each state represents a property of the input string read so far. In this case, State e corresponds to an even number of a's read, and State o corresponds to an odd number of a's read. This can be proven by induction to conclude that the automaton accepts the language $\{w \in \{a,b\}^* \mid \#_a(w)\}$.

Example. Let $A = \{a, b\}$. Consider the DFA



We label the nodes as ϵ, a, ab and abb



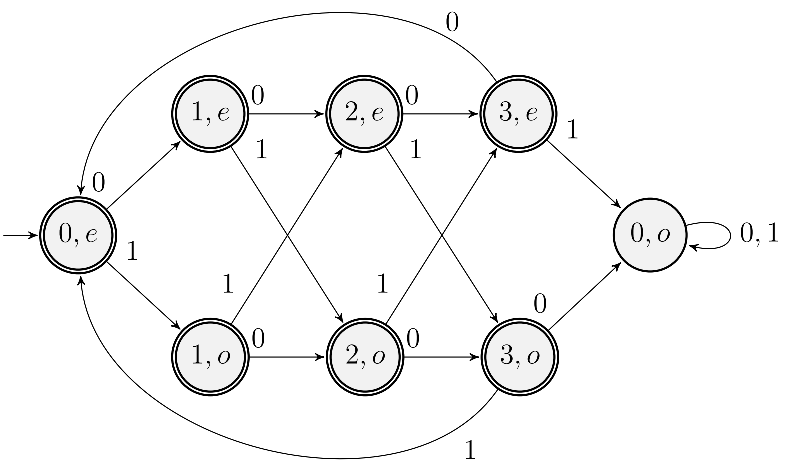
and consider the property corresponding to each state.

- State abb: the string seen so far contains abb.

Every other state will have the property that the string seen so far does not contain abb, in addition to some other property.

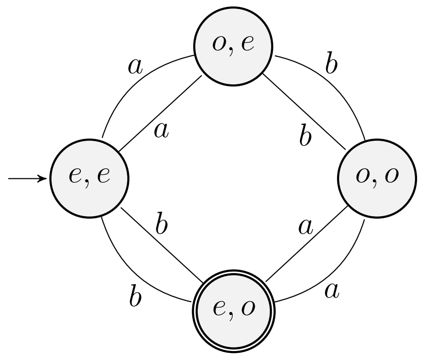
- State a: the string seen so far ends in a.
- State ab: the string seen so far ends in ab.
- State ϵ: every other string seen so far.

Here is another example of a DFA, which accepts strings over $\{0, 1\}$ that have even parity of 1s in each completed length 4 block.



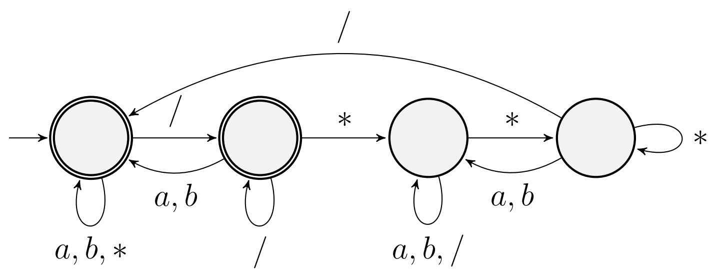
Exercise 2.1. Give a DFA that accepts strings over the alphabet $\{a, b\}$ containing an even number of a's and an odd number of b's.

Solution.



> **Exercise 2.2.** Give a DFA that accepts strings over $\{a, b, /, *\}$ which don't end inside a C-style comment, *i.e.*, comments of the form `/* ... */`.

Solution.



Lecture 03: Tue 09 Jan '24

2.1 A good way to construct DFAs

Suppose we have to construct a DFA for a language L over an alphabet A.

- Think of a finite number of properties of strings that you might want to keep track of. For example, "number of a's seen so far is even".
- Identify an initial property that is true of the empty string, say p_0.
- Make sure there is a rule to update the properties which are being tracked for a string wa, based purely on the properties for w and the last input a.
- The properties should imply membership in L or non-membership in L.

2.2 DFAs Formally

Definition 2.3 (DFA)**.** A *deterministic finite-state automaton* $\mathcal{A}$ over an alphabet A is a tuple (Q, s, δ, F) where

- Q is a finite set of *states*,
- $s \in Q$ is the *start state*,
- $\delta : Q \times A \to Q$ is the *transition function*,
- $F \subseteq Q$ is the set of *final states*.

For example, the first example in this chapter can be written as

$$\begin{aligned} A &= \{a, b\} \\ Q &= \{e, o\} \\ s &= e \\ F &= \{o\} \end{aligned}$$

and

$$\delta = \begin{matrix} (e, a) \mapsto o, & (o, a) \mapsto e, \\ (e, b) \mapsto e, & (o, b) \mapsto o. \end{matrix}$$

We further define $\hat{\delta} : Q \times A^* \to Q$ as the extension of δ to strings.

$$\begin{aligned} \hat{\delta}(q, \epsilon) &= q \\ \hat{\delta}(q, wa) &= \delta(\hat{\delta}(q, w), a) \end{aligned}$$

Definition 2.4 (Language of a DFA)**.** The *language of a DFA* $\mathcal{A}$ is

$$L(\mathcal{A}) = \left\{ w \in A^* \mid \hat{\delta}(s, w) \in F \right\}$$

2.3 Regular Languages

Definition 2.5 (Regular Language)**.** A language L is *regular* if there exists a DFA $\mathcal{A}$ over A such that $L(\mathcal{A}) = L$.

For example, the exercises we have done so far. Another example is *any* finite language.

Theorem 2.6. The class of regular languages over an alphabet is countable.

Proof. We partition the set of all DFAs over A by their number of states. For each $n \in \mathbb{N}$, there are finitely many DFAs with n states. A countable union of finite sets is countable. Thus the set of all DFAs over A is countable. Since each regular language corresponds to at least one DFA, the set of all regular languages over A is countable. $\square$

However, we have seen that there are uncountably many languages over any alphabet. This immediately yields the following.

Corollary 2.7. There are uncountably many languages that are not regular.

Theorem 2.8 (Closure under set operations)**.** The class of regular languages is closed under union, intersection and complementation.

Proof. For complementation, simply invert the set of final states. That is, given $\mathcal{A} = (Q, s, \delta, F)$, let $\mathcal{A}' = (Q, s, \delta, Q \setminus F)$. Then $L(\mathcal{A}') = A^* \setminus L(\mathcal{A})$, since

$$\begin{aligned} w \in L(\mathcal{A}') &\iff \hat{\delta}(s, w) \in Q \setminus F \\ &\iff \hat{\delta}(s, w) \notin F \\ &\iff w \notin L(\mathcal{A}) \\ &\iff w \in A^* \setminus L(\mathcal{A}) \end{aligned}$$

For intersection and union, define the *product* of two DFAs.

Definition 2.9 (Product)**.** Given two DFAs $\mathcal{A} = (Q, s, \delta, F)$ and $\mathcal{B} = (Q', s', \delta', F')$ over the same alphabet A, the *product* of $\mathcal{A}$ and $\mathcal{B}$ is

$$\mathcal{A} \times \mathcal{B} = (Q \times Q', (s, s'), \Delta, F \times F')$$

where $\Delta((q, q'), a) = (\delta(q, a), \delta'(q', a))$.

Note that in the above definition, the extension of Δ to strings $\hat{\Delta}$ is given by

$$\hat{\Delta}((q, q'), w) = (\hat{\delta}(q, w), \hat{\delta}'(q', w))$$

This is easily proved by induction on the length of w (or structural induction on w).

$$\begin{aligned}\hat{\Delta}((q,q'),\epsilon) &= (q,q') \\ &= (\hat{\delta}(q,\epsilon),\hat{\delta}'(q',\epsilon))\end{aligned}$$

and if

$$\hat{\Delta}((q,q'),w) = (\hat{\delta}(q,w),\hat{\delta}'(q',w))$$

then

$$\begin{aligned}\hat{\Delta}((q,q'),wa) &= \Delta(\hat{\Delta}((q,q'),w),a) \\ &= \Delta((\hat{\delta}(q,w),\hat{\delta}'(q',w)),a) \\ &= (\delta(\hat{\delta}(q,w),a),\delta'(\hat{\delta}'(q',w),a)) \\ &= (\hat{\delta}(q,wa),\hat{\delta}'(q',wa)).\end{aligned}$$

Now let $\mathcal{A}$, $\mathcal{B}$ be DFAs over A. Then $L(\mathcal{A}\times\mathcal{B}) = L(\mathcal{A})\cap L(\mathcal{B})$, since

$$\begin{aligned}w \in L(\mathcal{A}\times\mathcal{B}) &\iff \hat{\Delta}((s,s'),w) \in F\times F' \\ &\iff (\hat{\delta}(s,w),\hat{\delta}'(s',w)) \in F\times F' \\ &\iff \hat{\delta}(s,w)\in F \wedge \hat{\delta}'(s',w)\in F' \\ &\iff w\in L(\mathcal{A}) \wedge w \in L(\mathcal{B})\end{aligned}$$

Since $X\cup Y = \overline{\overline{X}\cap\overline{Y}}$, closure under union follows from closure under complementation and intersection.

More directly, the DFA $(Q\times Q',(s,s'),\Delta,F\times Q'\cup F'\times Q)$ accepts the language $L(\mathcal{A})\cup L(\mathcal{B})$. $\square$

Lecture 04: Thu 11 Jan '24

2.3.1 Two Necessary Conditions for Regular Languages

In a given DFA $\mathcal{A}$ with n states, any path of length greater than n must have a loop. Let u be the string of symbols on the path from the start state to the beginning of the loop, let v be the (non-empty) string of symbols on the loop, and let w be the string of symbols on the path from the end of the loop to the final state.

Then if uvw is accepted by $\mathcal{A}$, then so is uv^kw for any $k\geq 0$.

Theorem 2.10 (Pumping Lemma)**.** For any regular language L, there exists a constant k, such that for any word $t \in L$ of the form xyz with $|y| \geq k$, there exist strings u, v and w such that

(i) $y = uvw$, $v \neq \epsilon$, and

(ii) $xuv^iwz \in L$ for each $i \geq 0$.

Proof. Let L be accepted by a DFA $\mathcal{A} = (Q, s, \delta, F)$ with n states. Let t be a word in L of the form xyz with $|y| \geq n$. Let $y = a_1a_2 \ldots a_m$. Let $q_0 = \widehat{\delta}(s, x)$ and $q_i = \widehat{\delta}(q_{i-1}, a_n)$ for $1 \leq i \leq m$. By the pigeonhole principle, there exist $i < j$ such that $q_i = q_j$. Then letting $u = a_1 \ldots a_i$, $v = a_{i+1} \ldots a_j$, and $w = a_{j+1} \ldots a_m$, we have that $y = uvw$, $v \neq \epsilon$, $\widehat{\delta}(s, xu) = \widehat{\delta}(s, xuv) = q_i$. Since

$$\widehat{\delta}(q, xy) = \widehat{\delta}(\widehat{\delta}(q, x), y),$$

(which we will prove in assignment 1) we have that

$$\widehat{\delta}(q_i, v) = q_i$$

and so by induction

$$\widehat{\delta}(q_i, v^k) = q_i$$

which gives

$$\begin{aligned}
\widehat{\delta}(s, xuv^kwz) &= \widehat{\delta}\Big(\widehat{\delta}\Big(\widehat{\delta}(s, xu), v^k\Big), wz\Big) \\
&= \widehat{\delta}\Big(\widehat{\delta}(q_i, v^k), wz\Big) \\
&= \widehat{\delta}(q_i, wz) \\
&= \widehat{\delta}\Big(\widehat{\delta}(s, xuv), wz\Big) \\
&= \widehat{\delta}(s, t).
\end{aligned}$$

$\square$

Proposition 2.11. The language $\{a^nb^n \mid n \geq 0\}$ is not regular.

Proof. Let $k \in \mathbb{N}$. Choose $t = a^kb^k = xyz$ where $x = \epsilon$, $y = a^k$, and $z = b^k$. Let $y = uvw$ for some non-empty v. Then $v = a^j$ for some $j \geq 1$. Then $xuv^2wz = a^{k+j}b^k$, which is not in the language. Therefore, the language is not regular. $\square$

Exercise 2.12. Show that $\left\{a^{2^n} \mid n \geq 0\right\}$ is not a regular language.

Solution. Let $k \in \mathbb{N}$. Choose $t = a^{2^k} = xyz$ where $x = \epsilon$, $y = a^{2^k} - 1$, and $z = a$. Let $y = uvw$ for some non-empty v. Then $v = a^j$ for some $1 \leq j < 2^k$. Then $xuv^2wz = a^{2^k+j}$, which is not in the language since $2^k < 2^k + j < 2^{k+1}$.

Exercise 2.13. Is the language $\{w \cdot w \mid w \in \{0,1\}^*\}$ regular?

Solution. Let $k \in \mathbb{N}$. Choose $t = 0^k 1^k 0^k 1^k = xyz$ where $x = 0^k$, $y = 1^k$, and $z = 0^k 1^k$. Let $y = uvw$ for some non-empty v. Then $v = 1^j$ for some $1 \leq j \leq k$. If j is odd, we are done. Otherwise, $xuv^2wz = 0^k 1^{k+m} 1^m 0^k 1^k$, where $j = 2m$. This is not in the language since the second half starts with a 1.

Lecture 05: Tue 16 Jan '24

Definition 2.14. Let $L \subseteq A^*$ be a language. The *Kleene closure* of L, denoted L^*, is defined as

$$L^* = \bigcup_{n \in \mathbb{N}} L^n$$

where $L^0 = \{\epsilon\}$ and $L^{n+1} = L^n \cdot L$.

In other words,

$$L^* = \left\{s \in A^* \mid \exists\, w \in L^{\mathbb{N}} \text{ and } n \in \mathbb{N} \text{ such that } s = w_0 \cdots w_n\right\}$$

Exercise 2.15. If $L \subseteq \{a\}^*$, show that L^* is regular.

Exercise 2.16. Show that there exists a language $L \subseteq A^*$ such that neither L nor its complement $A^* \setminus L$ contains an infinite regular subset.

Solution. Assignment 2.

Definition 2.17 (Ultimate periodicity)**.** A subset X of $\mathbb{N}$ is said to be *ultimately periodic* if there exist $n_0 \in \mathbb{N}$, $p \in \mathbb{N}^*$ such that for all $m \geq n_0$, $m \in X$ iff $m + p \in X$.

Proposition 2.18. A subset X being ultimately periodic is equivalent to either

- there exist $n_0 \in \mathbb{N}$, $p \in \mathbb{N}^*$ such that for all $m \geq n_0$, $m \in X \implies m + p \in X$, or
-

Definition 2.19. For a language $L \subseteq A^*$, define lengths(L) to be $\{\#w \mid w \in L\}$.

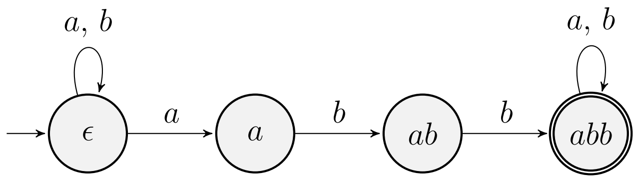
Theorem 2.20. If L is a regular language, then lengths(L) is ultimately periodic.

Lecture 06: Thu 18 Jan '24

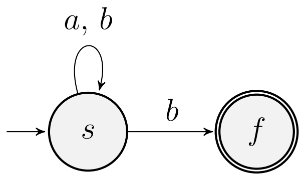
2.4 Non-deterministic Finite Automata

Examples.

- An NFA for "contains abb as a subword"



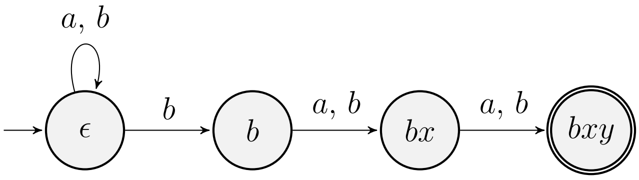
- An NFA for "last symbol is b"



Exercise 2.21. Give an NFA for the language of strings over $\{a, b\}$ in which the third-last symbol is a b.

Solution. We give an NFA $\mathcal{B}$ below, and claim that

$$L(\mathcal{B}) = \{ubxy \in \{a,b\}^* \mid u \in \{a,b\}^* \wedge x, y \in \{a,b\}\} =: L.$$



Suppose $w = ubxy \in L$. A valid path of $\mathcal{B}$ on w is as follows:

$$\epsilon \xrightarrow{u} \epsilon \xrightarrow{b} b \xrightarrow{x} bx \xrightarrow{y} bxy.$$

Now suppose $w \in L(\mathcal{B})$. Any path to (bxy) must pass through (b) and (bx) consecutively. Hence, $w = ubxy$ for some $u \in \{a, b\}^*$ and $x, y \in \{a, b\}$.

2.5 NFAs Formally

Definition 2.22. An NFA over an alphabet A is a tuple (Q, S, Δ, F) where

- Q is a finite set of states,
- $S \subseteq Q$ is a set of start states,
- $\Delta \colon Q \times A \to 2^Q$ is a transition function that returns a set of states for each state-symbol pair, and
- $F \subseteq Q$ is a set of final states.

We define the relation $p \xrightarrow{a} q$ which says there is a path from state p to state q labelled by w, as

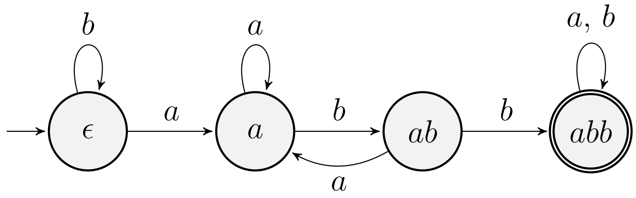
- $p \xrightarrow{\epsilon} p$ for all $p \in Q$, and
- $p \xrightarrow{ua} q$ if there is a state $r \in Q$ such that $p \xrightarrow{u} r$ and $q \in \Delta(r, a)$.

We define the language accepted by an NFA $\mathcal{A}$ as

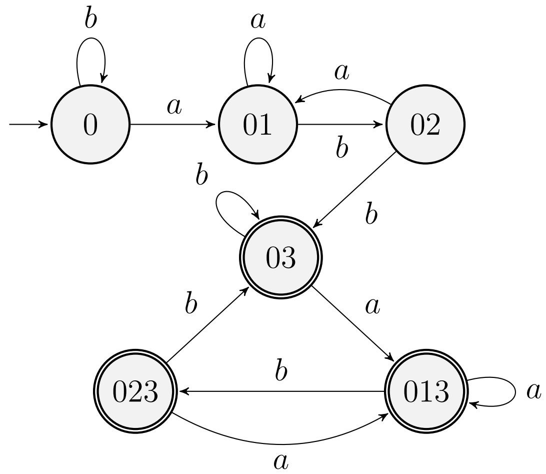
$$L(\mathcal{A}) := \left\{ w \in A^* \;\middle|\; s \xrightarrow{w} f \text{ for some } s \in S \text{ and } f \in F \right\}.$$

Exercise 2.23. Convert the "contains *abb* as a subword" NFA to a DFA.

Solution.



A more illustrative answer is as follows:



This illustrates the idea of the subset construction.

Theorem 2.24 (Closure under concatenation)**.** The class of regular languages is closed under concatenation.

Proof. Let A and B be regular languages accepted by DFAs $\mathcal{A}$ and $\mathcal{B}$ respectively, with a disjoint set of states.

We construct an NFA $\mathcal{W}$ for AB as follows:

- $Q = Q_{\mathcal{A}} \cup Q_{\mathcal{B}}$;
- $S = \{s_{\mathcal{A}}, s_{\mathcal{B}}\}$ if $\epsilon \in B$, and $S = \{s_{\mathcal{A}}\}$ otherwise;
- $F = F_{\mathcal{B}}$;
- $\Delta(q,a) = \begin{cases} \{\delta_{\mathcal{B}}(q,a)\} & \text{if } q \in Q_{\mathcal{B}}, \\ \{\delta_{\mathcal{A}}(q,a)\} & \text{if } q \in A \text{ and } \delta_{\mathcal{A}}(q,a) \notin F_A, \\ \{\delta_{\mathcal{A}}(q,a), s_{\mathcal{B}}\} & \text{otherwise.} \end{cases}$

This accepts the language AB. Thus AB is regular. □

Lecture 07: Tue 23 Jan '24

Theorem 2.25 (Closure under Kleene star)**.** The class of regular languages is closed under Kleene star.

Proof. Let L be a regular language accepted by a DFA $\mathcal{A} = (Q, s, \delta, F)$. We construct an NFA $\mathcal{A}' = (Q', S, \Delta, F')$ for L^* by

- $Q' = Q \cup \{Q\}$;
- $S = \{Q\}$;
- $F' = \{Q\}$;
- For $q \in Q$, $\Delta(q, a) = \begin{cases} \{\delta(q, a)\} & \text{if } \delta(q, a) \notin F, \\ \{\delta(q, a), Q\} & \text{otherwise.} \end{cases}$
- $\Delta(Q, a) = \Delta(s, a)$.

□

2.6 Regular Expressions

Consider the alphabet $\{a, b\}$. A regular expression over this language is built from a, b, ϵ, using operators $+$, $\cdot$, $*$, and parentheses.

Examples.

- $(a^*) \cdot b$ is "any number of a's followed by one b".
- $(a+b)^*abb(a+b)^*$ is "contains abb as a subword".
- $(a+b)^*b(a+b)(a+b)$ is "third last letter is b".
- $(b^*ab^*a)b^*$ is "has even number of as".
- (Exercise) Give a regular expression for "Every 4 bit block of the the form $4i$, $4i+1$, $4i+2$, $4i+3$ has an even number of 1s".
  (Answer) $(0000 + 0011 + \cdots + 1111)^* \cdot (\epsilon + 0 + 1 + 00 + \cdots + 111)$.

Definition 2.26. The syntax of regular expressions over an alphabet A is defined by

$$r ::= \varnothing \mid a \mid r + r \mid r \cdot r \mid r^*$$

with $a \in A$.

The semantics of regular expressions over an alphabet A is defined by

(i) $L(\varnothing) = \varnothing$.

(ii) $L(a) = \{a\}$.

(iii) $L(r_1 + r_2) = L(r_1) \cup L(r_2)$.

(iv) $L(r_1 \cdot r_2) = L(r_1) \cdot L(r_2)$.

(v) $L(r^*) = L(r)^* = \bigcup_{i=0}^{\infty} L(r)^i$.

We give precedence to $*$, $\cdot$, $+$, in that order.

Theorem 2.27 (Kleene's theorem)**.** The class of languages defined by regular expressions is exactly the class of regular languages.

We prove the two directions separately.

RE to NFA. Let L be a language corresponding to a regular expression r. We prove by induction on the structure of r that L is regular. For the base cases $r = \varnothing$ and $r = a$, we have $L = \varnothing$ and $L = \{a\}$, both of which are regular. The inductive cases follow from the closure properties of regular languages. □

Lecture 08: Thu 25 Jan '24

Proof. Let L be a regular language over an alphabet A, accepted by a DFA $\mathcal{A} = (Q, s, \delta, F)$. If the set of final states is empty, then $L = \varnothing$. Otherwise, we induct on the number of states of $\mathcal{A}$. For the base case, there is a DFA with one state which accepts all strings in L.

This is the regular expression $r = (a_1 + \cdots + a_k)^*$, where $A = \{a_1, \ldots, a_k\}$. Define L_{pq} to be $\left\{w \in A^* \mid \widehat{\delta}(p, w) = q\right\}$. Then $L(\mathcal{A}) = \bigcup_{f \in F} L_{sf}$. For $X \subseteq Q$, define

$$L_{pq}^X := \{w \in A^* \mid \widehat{\delta}(p, w) = q \text{ and } \widehat{\delta}(p, x) \in X \text{ for all non-empty proper prefixes } x \text{ of } w\}.$$

Claim: $L^{X+r}_{pq} = L^X_{pq} \cup L^X_{pr}(L^X_{rr})^* L^X_{rq}$. This is easily converted to regular expressions by substituting $+$ for $\cup$.

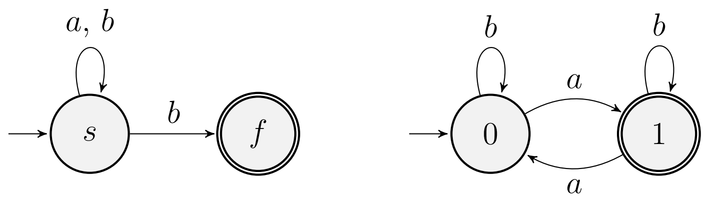
For the base case of $X = \varnothing$, observe that

$$L^{\varnothing}_{pq} = \{a \in A + \epsilon \mid \widehat{\delta}(p, a) = q\}$$

is finite, and hence can be converted to a regular expression.

By induction on $|X|$ with base case $\varnothing$, we go upto $X = Q$ to prove that $L = \bigcup_{f \in F} L^Q_{sf}$ can be converted to a regular expression. $\square$

Exercise 2.28. Convert the following automata to regular expressions.



Solution. For the first one, the regex of $L^{\varnothing}_{sf}$ is b. The regex of $L^{\{s\}}_{sf}$ is $b + (a + b + \epsilon)(a + b + \epsilon)^* b$. This can be simplified as

$$\begin{aligned} b + (a + b)(a + b)^* b + (a + b)^* b &= b + (a + b)^+ b + (a + b)^* b \\ &= (a + b)^* b + (a + b)^* b \\ &= (a + b)^* b \end{aligned}$$

The regex of $L^{\{s,f\}}_{sf}$ is also $(a + b)^* b$ since $L^{\{s\}}_{ff} = \varnothing$.

For the second one, $L^{\varnothing}_{01}$ has regex a. Then $L^{\{0\}}_{01}$ has regex $a + b(b + \epsilon)^* a = b^* a$.

We also need $L^{\{0\}}_{11}$. So we compute $L^{\varnothing}_{11} = b + \epsilon$ and

$$L^{\{0\}}_{11} = b + \epsilon + a(b + \epsilon)^* a = \epsilon + b + ab^* a.$$

So

$$L^{\{0,1\}}_{01} = b^* a + (b^* a)(b + \epsilon)^*$$

We can also view this as a system of equations. For example, for the second automaton of the exercise above, we set up equations to capture

$L_q = \left\{w \in A^* \mid \widehat{\delta}(q, w) \in F\right\}$ as

$$x_0 = b \cdot x_0 + a \cdot x_1$$
$$x_1 = \epsilon + a \cdot x_0 + b \cdot x_1$$

In general, such equations can have many solutions. But for equations arising from DFAs, we can show that there is a unique solution. We can write the equations above as

$$\begin{pmatrix} x_0 \\ x_1 \end{pmatrix} = \begin{pmatrix} b & a \\ a & b \end{pmatrix} \begin{pmatrix} x_0 \\ x_1 \end{pmatrix} + \begin{pmatrix} \varnothing \\ \epsilon \end{pmatrix}$$

For any such system $X = AX + B$, A^*B is the *least* solution. When A^* is a 2×2 matrix,

$$\begin{pmatrix} a & b \\ c & d \end{pmatrix}^* = \begin{pmatrix} (a + bd^*c)^* & (a + bd^*c)^*bd^* \\ (d + ca^*b)^*ca^* & (d + ca^*b)^* \end{pmatrix}$$

Reading: Kozen, Supplementary Lecture A.

Lecture 09: Tue 30 Jan '24

2.7 Myhill-Nerode Theorem

We will see several

- A language L is regular iff a certain equivalence relation induced by L (called $\equiv_L$) has a finite number of equivalence classes.
- Every language L has a "canonical" deterministic automaton accepting it. Every other DA for L is a "refinement" of this canonical DA. For regular languages, there is a unique DA for L with the minimal number of states.

  Remark. Every language L has a DA accepting it, the "free" DA for L, which has one state for each string over the alphabet.

Definition 2.29 (Refinement)**.** Let $\mathcal{A} = (Q, s, \delta, F)$ be a deterministic automaton over an alphabet A. We say that $\mathcal{A}$' is a *refinement* of $\mathcal{A}$ if there exists a Q-indexed partition $\left\{Q'_p\right\}_{p \in Q}$ of Q' such that for all $p, q \in Q$ and $a \in A$ such that $\delta(p, a) = q$, there exists a $p' \in Q'_p$ and a $q' \in Q'_q$ such that $\delta'(p', a) = q'$.

Definition 2.30. For any language $L \subseteq A^*$, we define the canonical equivalence relation $\equiv_L$ on A^* as

$$x \equiv_L y \iff \forall\, z \in A^*(xz \in L \iff yz \in L).$$

Exercise 2.31. Describe the equivalence classes for $L =$ "odd number of a's".

Solution. L and L^c.

Exercise 2.32. Describe precisely the equivalence classes of $\equiv_L$ for the language $L \subseteq \{a, b\}^*$ comprising strings in which the 2nd last letter is a b.

Solution. $\epsilon + a + (.^*)aa$, $b + (.^*)ab$, $(.^*)ba$, $(.^*)bb$.

Exercise 2.33. Descrive the equivalence classes of $\equiv_L$ for the language $L = \{a^n b^n \mid n \geq 0\}$.

Solution. L is the disjoint union of $\bigsqcup_{0 \leq m \leq n} \{a^n b^m\}$ and its complement.

Lecture 10: Thu 01 Feb '24

Definition 2.34 (Myhill-Nerode relation)**.** An MN relation for a language L over an alphabet A is an equivalence relation $\sim$ on A^* satisfying

- $\sim$ is right invariant, *i.e.*, if $x \sim y$, then for all $a \in A$, $xa \sim ya$.
- $\sim$ refines L, *i.e.*, if $x \sim y$, then $x \in L$ iff $y \in L$.

Note that the first condition is equivalent to saying that for all $x, y \in A^*$ and $w \in A^*$, $x \sim y$ implies $xw \sim yw$ This can be proven by induction on the length of w (the base case $w = \varepsilon$ is by the refinement condition).

Proposition 2.35. Let L be a language over an alphabet A. Then $\equiv_L$ is an MN relation for L.

Proof. $\equiv_L$ is right invariant. $x \equiv_L y$ iff for all $w \in A^*$, $xw \in L$ iff $yw \in L$. So for all $aw \in A^*$, $xaw \in L$ iff $yaw \in L$ and so $x \equiv_L y$.

$\equiv_L$ refines L. $x \equiv_L y$ implies that $x \in L$ iff $y \in L$ (take $w = \varepsilon$). $\square$

Proposition 2.36. Let L be a language over an alphabet A. Then DFAs for L are in one-to-one correspondence with finite-index MN relations for L in the following manner:

- Given a DFA $\mathcal{A} = (Q, q_0, \delta, F)$, the relation $\sim$ defined by $x \sim y$ iff $\widehat{\delta}(q_0, x) = \widehat{\delta}(q_0, y)$ is an MN relation for L. Denote by Φ the map sending $\mathcal{A}$ to $\sim$.
- Given an MN relation $\sim$ for L, the DFA $\mathcal{A} = (Q, q_0, \delta, F)$ defined by

$$\begin{aligned} Q &= \{[x]_\sim \mid x \in A^*\} \\ q_0 &= [\varepsilon]_\sim \\ \delta([x]_\sim, a) &= [xa]_\sim \\ F &= \{[x]_\sim \mid x \in L\} \end{aligned}$$

  is a DFA accepting L. Denote by Ψ the map sending $\sim$ to $\mathcal{A}$.

Moreover, these correspondences are inverses of each other, in the sense that for all DFAs $\mathcal{A}$ without unreachable states, $\Psi(\Phi(\mathcal{A})) \cong \mathcal{A}$; and for all MN relations $\sim$, $\Phi(\Psi(\sim)) = \sim$.

Here, $\cong$ denotes isomorphism of DFAs, *i.e.*, the DFAs are equivalent in the sence that their exists a bijection between their states that preserves the initial state, the transition function, and the set of accepting states.

Proof. Let $\mathcal{A} = (Q, q_0, \delta, F)$ be a DFA for L. We need to show that $\sim = \Phi(\mathcal{A})$ is an MN relation for L.

- $\sim$ is right invariant: Let $x \sim y$. Then $\widehat{\delta}(q_0, x) = \widehat{\delta}(q_0, y)$. It immediately follows that $\widehat{\delta}(q_0, xa) = \widehat{\delta}(q_0, ya)$.
- $\sim$ refines L: Let $x \sim y$. Then $\widehat{\delta}(q_0, x) = \widehat{\delta}(q_0, y) =: q$. Then $x \in L$ iff $q \in F$ iff $y \in L$.

Now let $\sim$ be an MN relation for L. We need to show that $\mathcal{A} = \Psi(\sim)$ is a DFA accepting L.

- δ is well-defined: Let $q = [x] = [y]$. Then $x \sim y$ and so for any $a \in A$, $xa \sim ya$. Thus $[xa] = [ya]$ and so δ is well-defined.
- $\mathcal{A}$ accepts L: We claim that,

$$\widehat{\delta}(q_0, w) = [w] \text{ for all } w \in A^* \tag{2.1}$$

The case $w = \varepsilon$ is immediate. Let $w = xa$. Then $\widehat{\delta}(q_0, w) = \delta(\widehat{\delta}(q_0, x), a) = \delta([x], a) = [xa] = [w]$. Thus $\mathcal{A}$ accepts a string w iff $[w] \in F$ which is true iff $w \in L$.

Why are Φ and Ψ inverses of each other? Let $\mathcal{A} = (Q, q_0, \delta, F)$ be a DFA for L with no unreachable states. Let $\sim = \Phi(\mathcal{A})$, and let $\mathcal{A}' = \Psi(\sim)$. We need to show that $\mathcal{A} \cong \mathcal{A}'$. Furthermore, we need to show that $\Phi(\Psi(\sim)) = \sim$.

- Define $\phi : Q' \to Q$ by $\phi([x]) = \widehat{\delta}(q_0, x)$. Equation (2.1) gives that $\phi(\widehat{\delta}'(q_0', x)) = \widehat{\delta}(q_0, x)$ for all $x \in A^*$.

  Note that every state in $\mathcal{A}$ is reachable. The same is true for $\mathcal{A}'$ since $Q = \left\{\widehat{\delta}'(q_0', x) \mid x \in A^*\right\}$.

  Thus ϕ preserves the initial state (let $x = \varepsilon$), the transition function, and the set of accepting states.

- Let $\sim' = \Phi(\Psi(\sim))$.

$$\begin{aligned} x \sim y &\iff [x]_\sim = [y]_\sim \\ &\iff \widehat{\delta}'(q_0', x) = \widehat{\delta}'(q_0', y) \\ &\iff x \sim' y \end{aligned}$$

□

Definition 2.37 (Refinement)**.** An equivalence relation S on a set X *refines* an equivalence relation R on X if $S \subseteq R$, *i.e.*, for all $x, y \in X$, if xSy then xRy.

Lemma 2.38. Let L be a language over an alphabet A. Let $\sim$ be an MN-relation for L. Then R refines $\equiv_L$.

Proof. $\sim$ is right invariant. This means that for all $x, y \in A^*$, if $x \sim y$ then $\forall\, z \in A^*(xz \sim yz)$, so $x \equiv_L y$. □

Theorem 2.39 (Myhill-Nerode)**.** $L \subseteq A^*$ is regular iff $\equiv_L$ is of finite index.

Proof. Follows from propositions 2.35 and 2.36 and lemma 2.38. If L is regular, then there exists a DFA $\mathcal{A}$ for L. Then $\Phi(\mathcal{A})$ is an MN relation for L, but it has finitely many equivalence classes and refines $\equiv_L$. So $\equiv_L$ has finitely many equivalence classes.

If $\equiv_L$ is of finite index, then $\Psi(\equiv_L)$ is a DFA for L. $\square$

Lecture 11: Tue 06 Feb '24

Definition 2.40 (Stable partitioning)**.** Let $\mathcal{A} = (Q, q_0, \delta, F)$ be a DFA. An equivalence relation $\sim$ on Q is said to be a *stable partitioning* of $\mathcal{A}$ if for all $p, q \in Q$, if $p \sim q$, then

- $p \in F$ iff $q \in F$, and
- $\delta(p, a) \sim \delta(q, a)$ for all $a \in A$.

Definition 2.41 (DFA refinement)**.** We say that a DFA $\mathcal{B}$ refines a DFA $\mathcal{A}$ iff there exists a stable partitioning $\sim$ of $\mathcal{B}$ such that $\mathcal{B}/\sim$ is isomorphic to $\mathcal{A}$.

Here, $\mathcal{B}/\sim$ is the DFA obtained by merging all states in the same equivalence class of $\sim$.

Proposition 2.42. Let $\sim$ be a stable partitioning of a DFA $\mathcal{A}$. Then $\mathcal{A}/\sim$ accepts the same language as $\mathcal{A}$.

Proof. Let $\mathcal{A} = (Q, q_0, \delta, F)$. Let $\mathcal{A}' = (Q/\sim, [q_0], \delta', F')$, where

$$\begin{aligned} \delta'([p], a) &= [\delta(p, a)] \\ F' &= \{[p] \mid p \in F\}. \end{aligned}$$

Suppose $\widehat{\delta}(q_0, w) = p$. Then we claim that $\widehat{\delta}'([q_0], w) = [p]$. This is immediate from the definition of δ'. Thus the two automata accept the same language. $\square$

Definition 2.43. Let $\mathcal{A} = (Q, s, \delta, F)$ be a DA for L with no unreachable states. We define $\approx_L$ as follows.

$$p \approx_L q \iff \exists\, x \equiv_L y \text{ such that } \widehat{\delta}(s, x) = p \text{ and } \widehat{\delta}(s, y) = q.$$

Proposition 2.44. $\approx_L$ is a stable partitioning of $\mathcal{A}$.

Proof. Suppose $p \approx_L q$. Then $p \in F$ iff $x \in L$ iff $y \in L$ iff $q \in F$. Also, since $xa \equiv_L ya$ for all $a \in A$, we have that $\widehat{\delta}(s, xa) \approx_L \widehat{\delta}(s, ya)$ and so $\delta(p, a) \approx_L \delta(q, a)$ for all $a \in A$. $\square$

Theorem 2.45. Let $\mathcal{A} = (Q, s, \delta, F)$ be a DFA for L with no unreachable states. Define $\approx$ as follows.

$$p \approx q \iff \forall\, z \in A^* \left(\widehat{\delta}(p, z) \in F \iff \widehat{\delta}(q, z) \in F\right).$$

Then $\approx = \approx_L$.

Proof. Let $p \approx q$. Then for all $z \in A^*$, $\widehat{\delta}(p, z) \in F$ iff $\widehat{\delta}(q, z) \in F$. Since no state in $\mathcal{A}$ is unreachable, let $p = \widehat{\delta}(s, x)$ and $q = \widehat{\delta}(s, y)$. Then $\widehat{\delta}(s, xz) \in F \iff \widehat{\delta}(s, yz) \in F$ so that $x \equiv_L y$. Thus $p \approx_L q$.

Conversely, let $p \approx_L q$. Let $x \equiv_L y$ such that $\widehat{\delta}(s, x) = p$ and $\widehat{\delta}(s, y) = q$. Then since $xz \in L$ iff $yz \in L$, we have that $\widehat{\delta}(p, z) \in F$ iff $\widehat{\delta}(q, z) \in F$. $\square$

Exercise 2.46. Run algorithm to compute $\approx$ for the DFA below

Solution.

	s	p	q	r
s				
p	✓			
q		✓		
r	✓		✓	

2.7.1 Analysis of the minimization algorithm

We claim that the algorithm always terminates. Let $n = |Q|$. Since the algorithm terminates when the table is unchanged, it must terminate after at most $\binom{n}{2}$ iterations.

In fact, the algorithm terminates after at most n iterations. In each iteration, we mark a pair if they lead to some states upon reading the same input, which were marked in the previous iteration.

We argue that at the end of the i^{th} iteration, the algorithm has marked all pairs of states that are distinguishable by a string of at most i symbols.

$i = 0$ is trivial. Suppose this holds for some $i \geq 0$. Let $p, q \in Q$ be such that p and q are distinguishable by a string w of at most $i+1$ symbols. If $|w| \leq i$, then p and q are marked at the end of the i^{th} iteration. Otherwise, $w = av$ for some $a \in A$ and $v \in A^i$. Then since $\widehat{\delta}(p, w) = \widehat{\delta}(\delta(p, a), v)$ and

$\widehat{\delta}(q, w) = \widehat{\delta}(\delta(q, a), v)$, we have that $\delta(p, a)$ and $\delta(q, a)$ are distinguishable by a string of i symbols, and so they will be marked at the end of the i^{th} iteration. This means that p and q will be marked at the end of the $(i+1)^{\text{th}}$ iteration.

Lecture 12: Thu 08 Feb '24

Chapter 3

Context-Free Grammars

3.1 Introduction

The syntax of regular expressions over an alphabet $\{a, b\}$ is defined by

$$r ::= \varnothing \mid a \mid b \mid r + r \mid r \cdot r \mid r^*.$$

This is an example of a context-free grammar (CFG). Context-free grammars arise enaturally in syntax of programming language, parsing, compiling.

Examples.

- G_1 is the grammar given by the rules

$$\begin{aligned} S &\to aX \\ X &\to aX \\ X &\to bX \\ X &\to b \end{aligned}$$

  A string is derived from these rules as follows: Begin with S and keep rewriting the current string by replacing a non-terminal with the right-hand side in a production rule. For example,

$$S \to aX \to abX \to abb.$$

  The language defined by a grammer G, written $L(G)$, is the set of all terminal strings that can be generated by G.
  The language generated by G_1 is $a(a + b)^*b$.

- G_2 is the grammar given by the rules

$$\begin{aligned} S &\to aSb \\ S &\to \epsilon \end{aligned}$$

An example string in this grammer is

$$S \to aSb \to aaSbb \to aaaSbbb \to aaabbb$$

and $L(G) = \{a^n b^n \mid n \geq 0\}$. Suppose we add the rule $S \to bSa$. We write this in short as

$$S \to aSb \mid bSa \mid \epsilon.$$

The language generated by this grammar is all even length "inverse" palindromes, *i.e.*, the mirror image of any alphabet about the midpoint inverts a's to b's and vice versa.

- Let G_3 be given be

$$S \to aSa \mid bSb \mid a \mid b \mid \epsilon.$$

Then $L(G_3)$ is all palindromes.

- Let G_4 be given by

$$S \to (S) \mid SS \mid \epsilon.$$

This gives the language of all balanced parentheses. Formally, a $w \in \{(,)\}$ is balanced if

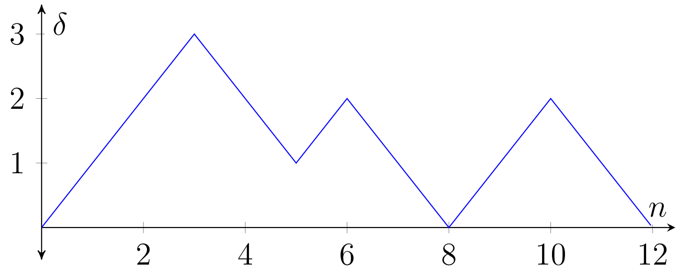
  - $\#_((w) = \#_)(w)$, and
  - for each prefic u of w, $\#_((u) \geq \#_)(u)$.

Exercise 3.1. Derive the string $((()())()())$ from G_4.

Solution.

$$\begin{aligned} S &\to (S) \\ &\to (SS) \\ &\to ((S)S) \\ &\to ((SS)SS) \\ &\to (((S)(S))(S)(S)) \\ &\to ((()())()()) \end{aligned}$$

One can visualise any balanced string w as a graph of points (n, δ), where $0 \leq n \leq |w|$ and δ is the difference between the number of left and right parentheses in the prefix of w of length n. The graph starts at $(0,0)$ and ends at $(|w|,0)$, and never goes below the x-axis.



Definition 3.2 (Context-free grammar)**.** A Context-Free Grammar (CFG) is a 4-tuple $G = (N, A, S, P)$, where

- N is a finite set of *non-terminal symbols*,
- A is a finite set of *terminal symbols* disjoint from N,
- $S \in N$ is the non-terminal *start symbol*, and
- P is a finite subset of $N \times (N \cup A)^*$, called the set of *productions* or *rules*. A production (X, α) is written as $X \to \alpha$.

We will denote letters with lower-case letters, non-terminals with upper-case letters, and strings of both letters and non-terminals with Greek letters.

Definition 3.3. Given a CFG $G = (N, A, S, P)$, we define the relation $\stackrel{n}{\Rightarrow}$ on $(N \cup A)^*$ inductively as follows:

- $\alpha \stackrel{0}{\Rightarrow} \alpha$;
- $\alpha \stackrel{1}{\Rightarrow} \beta$ if α is of the form $\alpha_1 X \alpha_2$ and $X \to \gamma$ is a production rule such that $\beta = \alpha_1 \gamma \alpha_2$; and
- $\alpha \xRightarrow{n+1} \beta$ if there exists a string γ such that $\alpha \stackrel{n}{\Rightarrow} \gamma$ and $\gamma \stackrel{1}{\Rightarrow} \beta$.

We further define $\Rightarrow_G^*$ as the union of all $\stackrel{n}{\Rightarrow}$ for $n \in \mathbb{N}$.

A *sentential form* of G is any $\alpha \in (N \cup A)^*$ such that $S \Rightarrow_G^* \alpha$.

The language defined by G is $L(G) = \{w \in A^* \mid S \Rightarrow_G^* w\}$.

3.2 Parse Trees

Each derivation of a string w from a CFG G can be represented by a parse tree, where each internal node is labelled by a non-terminal and each leaf is labelled by a terminal or ϵ. Each internal node has children corresponding to the right-hand side of a production rule for the non-terminal label of the node.

The string represented by a parse tree is the concatenation of the labels of the leaves read from left to right.

Exercise 3.4. Consider G_1 given in the examples above.

$$\begin{aligned} S &\to aX \\ X &\to aX \mid bX \mid b \end{aligned}$$

Prove that $L(G_1) = a(a+b)^*b$.

Proof. Let $P(n)$ be that for any $w \in (N \cup A)^*$, $S \overset{n}{\Rightarrow} w$ if and only if

$$w \in a(a+b)^{n-1}X + a(a+b)^{n-2}b,$$

where we consider the second term to be $\varnothing$ if $n < 2$. The case $n = 1$ is direct.

Suppose $P(k)$ holds. Let $w \in (N \cup A)^*$. Then $S \overset{k+1}{\Longrightarrow} w$ iff there is some α such that $S \overset{k}{\Rightarrow} \alpha$ and $\alpha \overset{1}{\Rightarrow} w$. By the induction hypothesis, this is iff $\alpha \in a(a+b)^{k-1}X + a(a+b)^{k-2}b$ and $\alpha \overset{1}{\Rightarrow} w$. Since the second term has no non-terminals, this is equivalent to $\alpha \in a(a+b)^{k-1}X$ and $\alpha \overset{1}{\Rightarrow} w$ so

$$\begin{aligned} w &\in a(a+b)^{k-1}aX + a(a+b)^{k-1}bX + a(a+b)^{k-1}b \\ &= a(a+b)^k X + a(a+b)^{k-1}b \end{aligned}$$

Thus $P(k+1)$ holds.

By induction, $P(n)$ holds for all $n \in \mathbb{N}$. Then $L(G_1) = \{w \in A^* \mid S \Rightarrow_G^* w\}$ is the union of $a(a+b)^{n-2}b$ which is $a(a+b)^*b$. $\square$

3.3 Chomsky Normal Form

Lecture 13: Tue 13 Feb '24

Definition 3.5 (Chomsky normal form)**.** A context-free grammar G is said to be in *Chomsky normal form* if all of its production rules are of the form

$$X \to YZ$$
$$X \to a$$

where Y and Z are non-terminals, and a is a terminal.

Example. Consider G_4 in the examples above, which generates balanced parentheses.

$$S \to (S) \mid SS \mid \epsilon$$

This can be converted into a CNF as follows

$$\begin{aligned} L &\to (\\ R &\to) \\ S &\to LR \mid SS \mid LX \\ X &\to SR \end{aligned}$$

Why do we care about Chomsky normal form? Suppose we have a context-free grammar G and a string w. We want to know if $w \in L(G)$. This is hard to do in general, but it is trivial if G is in CNF. Any production rule applied to an intermediate string w cannot decrease the length of w, so we will know to terminate in finitely many steps.

Theorem 3.6. Every context-free grammar G can be converted into a Chomsky normal form grammar G' such that $L(G') = L(G) \setminus \{\epsilon\}$.

Choose any problematic production rule in G. If the RHS has more than two (say n) non-terminals, we can introduce a new non-terminal in place of $n-1$ of them, from which we generate those $n-1$ non-terminals in sequence.
If the RHS has more than one terminal, we can introduce a new non-terminal for each of those, and we have just shown how to deal with that case.
In fact, if the RHS is of length at least 2, we can replace its terminals with non-terminals.
Thus the only problematic case is when the RHS is either a single terminal, a single non-terminal, or the empty string.

Theorem 3.7. Let G be a context-free grammar. Then there is a context-free grammar G' such that $L(G') = L(G)$ and G' has no unit- or ϵ-productions.

We give an algorithm to achieve this, and will prove its correctness later.

Let $G = (N, A, S, P)$ be a context-free grammar. Create a new set of productions P' as follows: First add all the productions from P to P'. Then,

- if P has productions $X \to \alpha Y \beta$ and $Y \to \epsilon$, add the rule $X \to \alpha\beta$ to P'.
- if $X \to Y$ and $Y \to \gamma$, add $X \to \gamma$ to P'.

This gives us a new grammar $G' = (N, A, S, P')$. Finally, drop all unit- and ϵ-productions from P' to get P''. Then $G'' = (N, A, S, P'')$ is an "equivalent" grammar without unit- or ϵ-productions. Equivalence is in the sense that $L(G') = L(G) \setminus \{\epsilon\}$.

Example. We apply this to G_4 from above. The inital grammar is

$$S \to \texttt{(}S\texttt{)} \mid SS \mid \epsilon.$$

We can apply the first rule to add the production

$$S \to \texttt{()}.$$

There are no more productions to add, so we remove the ϵ-production to get the grammar

$$S \to \texttt{()} \mid \texttt{(}S\texttt{)} \mid SS.$$

Exercise 3.8. Put the following grammar into CNF.

$$\begin{aligned} S &\to aSbb \mid T \\ T &\to bTaa \mid S \mid \epsilon \end{aligned}$$

Solution. By rule 2, we can add all productions of S to T, and vice versa. This gives the grammar

$$S, T \to aSbb \mid T \mid bTaa \mid S \mid \epsilon$$

By rule 1, we can add the productions

$$\begin{aligned} S &\to abb \quad \text{since } S \to aSbb \text{ and } S \to \epsilon \\ S &\to baa \quad \text{since } S \to bTaa \text{ and } T \to \epsilon \end{aligned}$$

And add both of these to T as well. This gives the grammar

$$S, T \to aSbb \mid T \mid bTaa \mid S \mid \epsilon \mid abb \mid baa.$$

We cannot add any more productions, so we have our grammar G'. Dropping unit- and ϵ-productions gives us G'' as

$$\begin{aligned} S &\to aSbb \mid bTaa \mid abb \mid baa \\ T &\to aSbb \mid bTaa \mid abb \mid baa \end{aligned}$$

We can obviously omit T and replace it with S to get

$$S \to aSbb \mid bSaa \mid abb \mid baa$$

We now prove that the algorithm terminates with a correct grammar.

Termination. The algorithm terminates because any new production added has an RHS that is a subsequence of the RHS of an original production. Only finitely many such subsequences exist. $\square$

Correctness. We first show that $L(G') = L(G)$. Let G'_i be the grammar after the ith iteration of the algorithm. Clearly $L(G'_0) = L(G)$. It is easy to see that $L(G'_{i+1}) = L(G'_i)$. Thus $L(G') = L(G)$.

We need to show that $L(G'') = L(G') \setminus \{\epsilon\}$. We do this by first showing that for any $w \in L(G') \setminus \{\epsilon\}$, any minimal length derivation of $w \in G'$ does not use unit- or ϵ-productions.

Suppose the derivation uses an ϵ-production $Y \to \epsilon$. Since $w \neq \epsilon$, this cannot be the first step. So the derivation looks like

$$S \overset{l}{\Rightarrow} \alpha X \beta \overset{1}{\Rightarrow} \alpha\alpha' Y \beta'\beta \overset{m}{\Longrightarrow} \gamma Y \delta \overset{1}{\Rightarrow} \gamma\delta \overset{n}{\Rightarrow} w$$

where $\alpha\alpha' \rightsquigarrow \gamma$ and $\beta'\beta \rightsquigarrow \delta$. But then we can give a shorter derivation

$$S \overset{l}{\Rightarrow} \alpha X \beta \overset{1}{\Rightarrow} \alpha\alpha'\beta'\beta \overset{m}{\Longrightarrow} \gamma\delta \overset{n}{\Rightarrow} w.$$

Similarly, a derivation which contains a unit-production

$$S \overset{l}{\Rightarrow} \alpha X \beta \overset{1}{\Rightarrow} \alpha Y \beta \overset{m}{\Longrightarrow} \gamma Y \delta \overset{1}{\Rightarrow} \gamma\phi\delta \overset{n}{\Rightarrow} w$$

can be shortened to

$$S \overset{l}{\Rightarrow} \alpha X \beta \overset{1}{\Rightarrow} \alpha\phi\beta \overset{m}{\Longrightarrow} \gamma\phi\delta \overset{n}{\Rightarrow} w.$$

This proves that for any $w \in L(G')\setminus\{\epsilon\}$, $w \in L(G'')$. Thus $L(G')\setminus\epsilon \subseteq L(G'')$ and since $P'' \subseteq P'$, $L(G'') \subseteq L(G)$. All that remains to be shown is that

$L(G'')$ does not contain ϵ.

Since each production in P'' (weakly) increases the length of any intermediate string, no derivation in G'' can produce ϵ. $\square$

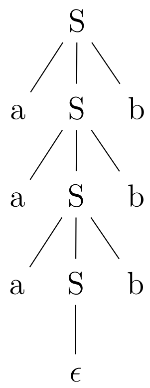
Lecture 14: Thu 15 Feb '24

3.4 Pumping Lemma

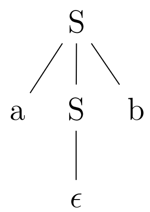
Theorem 3.9 (Pumping lemma for CFLs)**.** For every CFL L there is a constant $k \geq 0$ such that such that for any word z in L of length at least k, there are strings u, v, w, x, y such that

- $z = uvwxy$,
- $vx \neq \epsilon$,
- $|vwx| \leq k$, and
- for each $i \geq 0$, the string uv^iwx^iy belongs to L.

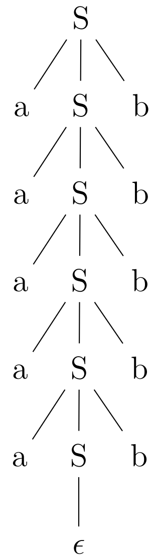
Consider a parse tree for any string. Note that subtrees hanging at the same non-terminal can be replaced by each other. For example, if



is a derivation, then so are



and



Proof. Let $G = (N, A, S, P)$ be a CNF grammar for L. A full binary tree of height h has 2^h leaves. A parse tree in G with height h has a terminal string of length at most 2^{h-1}, since a terminal node has no sibling. Thus a string of length 2^n or more, must have a parse tree of at least height $n+1$. Let $k = 2^{|N|}$. Consider a parse tree in G of a string z of length at least k. The longest path from the root to a leaf has length at least $|N| + 1$, and thus has at least $|N| + 2$ nodes, or $|N| + 1$ non-terminal nodes. By the pigeonhole principle, two of these nodes must have the same label. Let X be the lowest repeated non-terminal, and let $X_\perp$ and $X^\top$ be the two lowest occurrences of X, with $X_\perp$ closer to the leaves.

Let w be the string of terminals derived from $X_\perp$, and let vwx be the string of terminals derived from $X^\top$. Let $z = uvwxy$. Since the longest path from $X^\top$ down to a leaf has length at most $|N| + 1$, we have $|vwx| \leq 2^{|N|} = k$.

One of the strings v and x must be non-empty, since G is CNF ($X^\perp$ must have a sibling, which must lead to a terminal). We can then replace the subtree at $X^\top$ by the subtree at $X_\perp$, and obtain a parse tree for uwy. We can also replace the subtree at $X^\perp$ by the subtree at $X_\top$, and obtain a parse tree for uv^2wx^2y.

Continuing in this way, we can obtain a parse tree for uv^iwx^iy for any $i \geq 0$. $\square$

Exercise 3.10. Show that the following languages are not context-free.

- $\{a^n b^n c^n \mid n \geq 0\}$.
- $\{ww \mid w \in \{a, b\}^*\}$.

Solution.

- Let k be the pumping lemma constant. Then $a^k b^k c^k$ is in the language, and so there are strings u, v, w, x, y such that $uvwxy = a^k b^k c^k$, $vx \neq \epsilon$, $|vwx| \leq k$, and uv^2wx^2y is in the language. Since $|vwx| \leq k$, it cannot contain all three letters. Since $vx \neq \epsilon$, uwy will have more of the letters that vwx does not contain, and less of the letters it does contain.

- Suppose it is. Let k be the pumping lemma constant. Then
$$a^{k+1} b^{k+1} a^{k+1} b^{k+1}$$
is in the language, and so there are strings u, v, w, x, y as above. Since $|vwx| \leq k$, it cannot overlap with 3 or 4 of the homogenous blocks.

  Thus uwx will still be of the form $a^p b^q a^r b^s$, with none of p, q, r, or s being zero. Since uwy is in the language, $(p, q) = (r, s)$. But removing v and x will change the number of as and bs in only one of the blocks, so $uwy \neq a^p b^q a^p b^q$.

3.5 Closure properties

	Closed?
Union	✓
Intersection	✗
Complement	✗
Concatenation	✓

Table 3.1: Closure properties of CFLs.

We have the following closure properties for CFLs.

Theorem 3.11. The class of context-free languages is closed under union and concatenation.

Proof. Let $G_1 = (N_1, A, S_1, P_1)$ and $G_2 = (N_2, A, S_2, P_2)$ be CFGs for L_1 and L_2. Let $N = N_1 \cup N_2 \cup \{S\}$, where S is a new start symbol. Let $G = (N, A, S, P)$, where

$$P = P_1 \cup P_2 \cup \{S \to S_1 \mid S_2\}.$$

Then G is a CFG for $L_1 \cup L_2$. $\square$