UMC 205: Automata and Computability

Naman Mishra

January 2024

Contents

Lectures

The Course

Lecture 1.
Tuesday
January 02

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/

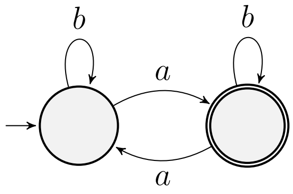
Grading

- 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 I

Finite-State Automata

I.1 Languages

Lecture 2.
Thursday
January 04

Definition I.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, denoted $|s|$, 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 I.2. *Let A be an alphabet. Then A^* is countably infinite.*

Proof. Let $n = \#A$. Let $f\colon A \to \{1, \dots, n\}$ 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. □

Definition I.3 (Language)**.** A *language* over an alphabet A is a subset of A^*.

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

Definition I.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 I.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 I.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. □

I.2 Deterministic Finite-State Automata

Figure I.1 shows 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

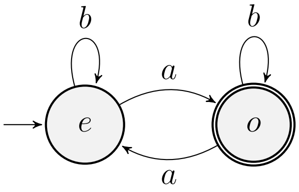
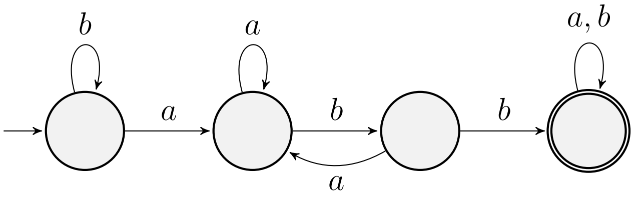


Figure I.1: A DFA that accepts strings with an odd number of a's.

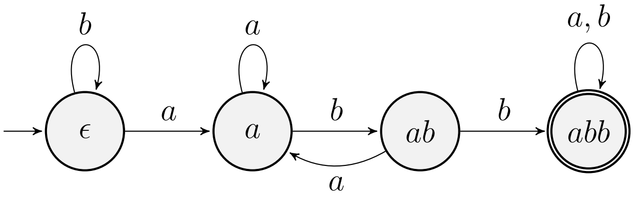
to the state specified by the arrows from the current state. State o is an ‘accepting state’. Each string whose evaluation ends at state o is said to be accepted by the automaton.

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



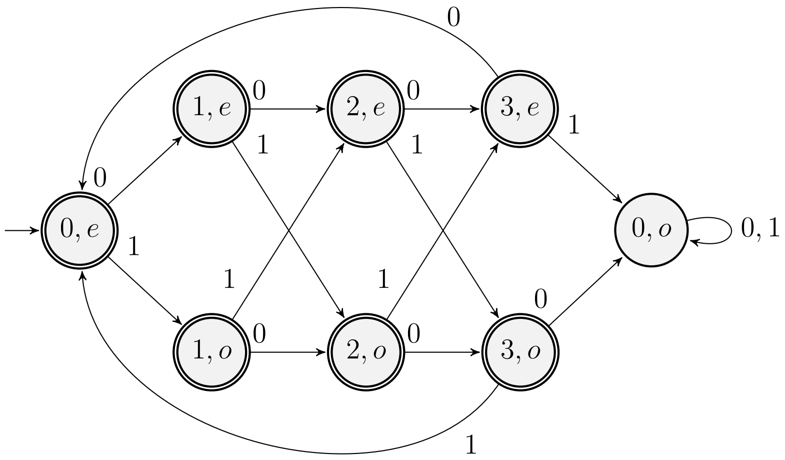
and consider the property corresponding to each state.

- 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.

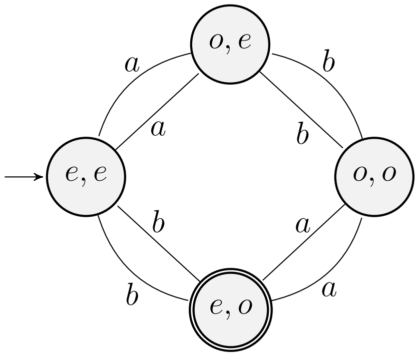
- State a: the string seen so far ends in a.
- State ab: the string seen so far ends in ab.
- State ϵ: every other string seen so far.

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 I.7. *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.*

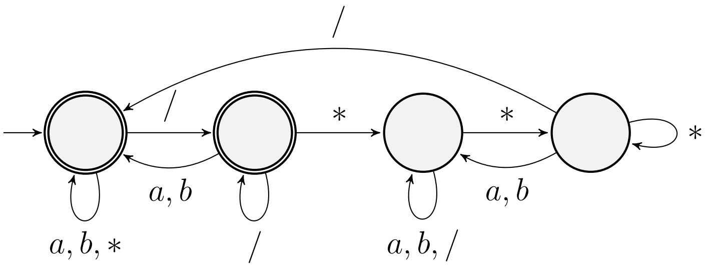
Solution.



■

Exercise I.8. *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.



■

I.3 A good way to construct DFAs

Lecture 3. Tuesday January 09

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.

I.4 DFAs Formally

Definition I.9 (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 DFA in figure I.1 can be written as

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

and

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

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

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

Definition I.10 (Language of a DFA)**.** The *language of a DFA* $\mathcal{A}$ is

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

I.5 Regular Languages

Definition I.11 (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 I.12. *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. □

Lecture 3

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

Corollary I.13. *There are uncountably many languages that are not regular.*

Theorem I.14 (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

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

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

Definition I.15 (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 $\widehat{\Delta}$ is given by

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

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

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

and if

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

then

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

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

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

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 $L(\mathcal{A}) \cup L(\mathcal{B})$. □

I.6 Two Necessary Conditions for Regular Languages

Lecture 4. Thursday January 11

In a given DFA $\mathcal{A}$ with n states, any path of length greater than $n-1$ must contain a cycle. Let u be the string of symbols on the path from the start state to the beginning of the cycle, let v be the (non-empty) string of symbols on the cycle, and let w be the string of symbols on the path from the end of the cycle to the final state.

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

Theorem I.16 (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 $y = a_1 a_2 \dots a_m$. 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 $u = a_1 \dots a_i$, $v = a_{i+1} \dots a_j$, and $w = a_{j+1} \dots a_m$, 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

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

□

Proposition I.17. *The language $\{a^n b^n \mid n \geq 0\}$ is not regular.*

Proof. Let $k \in \mathbb{N}$. Choose $t = a^k b^k = xyz$ 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. □

We will later see that it is a context-free language.

Exercise I.18. *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 I.19. *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 5.
Tuesday
January 16

Definition I.20. 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 I.21. *If* $L \subseteq \{a\}^*$, *show that* L^* *is regular.*

Exercise I.22. *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 I.23 (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 I.24. *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 I.25. For a language $L \subseteq A^*$, define lengths(L) to be $\{\#w \mid w \in L\}$.

Theorem I.26. *If* L *is a regular language, then* lengths(L) *is ultimately periodic.*

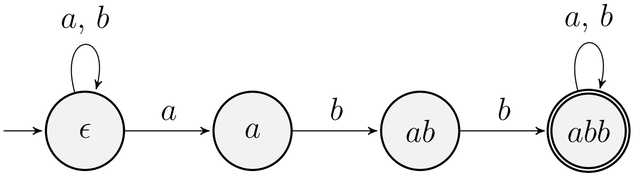
Lecture 6

I.7 Non-deterministic Finite Automata

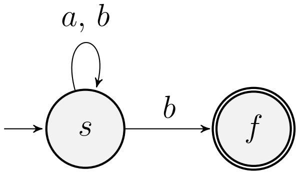
Lecture 6.
Thursday
January 18

Examples.

- An NFA for “contains *abb* as a subword”



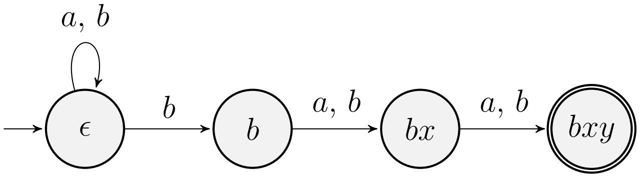
- An NFA for “last symbol is *b*”



Exercise I.27. *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 $x, y \in \{a,b\}$. ■

I.8 NFAs Formally

Definition I.28. 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

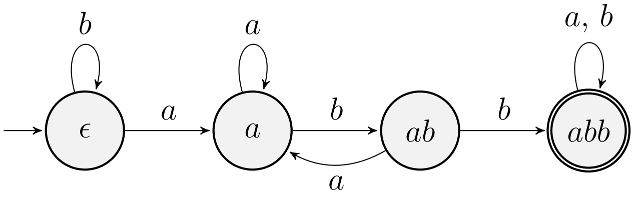
- $p \xrightarrow{\epsilon} p$ for all $p \in Q$, and
- $p \xrightarrow{ua} q$ if there is a state $r \in Q$ such that $p \xrightarrow{u} r$ and $q \in \Delta(r, a)$.

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

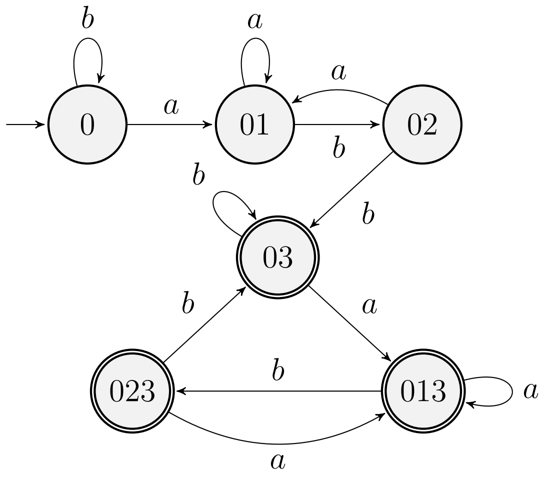
$$L(\mathcal{A}) \coloneqq \left\{ w \in A^* \;\middle|\; s \xrightarrow{w} f \text{ for some } s \in S \text{ and } f \in F \right\}.$$

Exercise I.29. *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 I.30. *Every NFA has an equivalent DFA.*

Proof. We give the *subset construction* which converts an NFA to an equivalent DFA. Let $\mathcal{A} = (Q, S, \Delta, F)$ be an NFA. We construct a DFA $\mathcal{B} = (2^Q, S, \delta, F')$ where

$$\delta(X, a) = \bigcup_{p \in X} \Delta(p, a),$$
$$F' = \{X \subseteq Q \mid X \cap F \neq \emptyset\}.$$

We claim that

$$\widehat{\delta}(X, w) = \bigcup_{p \in X} \left\{ q \in Q \mid p \xrightarrow{w} q \right\}.$$

The base case $w = \epsilon$ is direct. For the inductive case, we have

$$\begin{aligned}\widehat{\delta}(X, aw) &= \widehat{\delta}(\delta(X, a), w)\\ &= \bigcup_{p \in \delta(X,a)} \left\{ q \in Q \mid p \xrightarrow{w} q \right\}\end{aligned}$$

by the inductive hypothesis

$$\begin{aligned}&= \bigcup_{p \in \{q \in Q \mid \exists q_0 \in X (q_0 \xrightarrow{a} p)\}} \left\{ q \in Q \mid p \xrightarrow{w} q \right\}\\ &= \left\{ q \in Q \mid \exists q_0 \in X (q_0 \xrightarrow{aw} q) \right\}.\end{aligned}$$

□

Theorem I.31 (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 $\epsilon \in A$, 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 7.
Tuesday
January 23

Theorem I.32 (Glushkov construction)**.** *For every regular language L, there exists an NFA with exactly one initial and final state that accepts $L \setminus \{\epsilon\}$.*

Proof. Let $\mathcal{A} = (Q, S, \Delta, F)$ be an NFA accepting L. Let s and f be new states. We construct an NFA

$$\mathcal{A}' = (Q \cup \{s,f\}, \{s\}, \Delta', \{f\})$$

where

$$\Delta'(s,a) = \bigcup_{q \in S} \Delta(q,a),$$
$$\Delta'(q,a) = \begin{cases} \Delta(q,a) & \text{if } \Delta(q,a) \cap F = \varnothing \\ \Delta(q,a) \cup \{f\} & \text{otherwise.} \end{cases}$$

Let $w \in L \setminus \{\epsilon\}$. Then there exists a path from some initial state s_i to some final state f_j in $\mathcal{A}$. This path is preserved in $\mathcal{A}'$, only with s in place of s_i and f in place of f_j.

Conversely, every path from s to f in $\mathcal{A}'$ corresponds to a path from some initial state to some final state in $\mathcal{A}$. □

Theorem I.33 (Closure under Kleene star)**.** *The class of regular languages is closed under Kleene star.*

Proof. Let L be a regular language with NFA $\mathcal{A} = (Q, S, \Delta, F)$. Let $\mathcal{A}'$ be the NFA constructed using the Glushkov construction. Now fuse the initial and final states of $\mathcal{A}'$ to obtain an NFA accepting L^*. □

I.9 Regular Expressions

Consider the alphabet $\{a, b\}$. A regular expression over this language is built from a, b, ϵ and $\varnothing$ 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 I.34. 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 *, $\cdot$ and $+$ in that order.

Parentheses can be used to override precedence. Note that we do not give ϵ as an atom, but it can be represented using $\varnothing^*$.

Theorem I.35 (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 8. Thursday January 25

DFA to RE. 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 $\{w \in A^* \mid \widehat{\delta}(p, w) = q\}$. Then $L(\mathcal{A}) = \bigcup_{f \in F} L_{sf}$. For $X \subseteq Q$, define

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

It is easy to see that $L^{X+r}_{pq} = L^X_{pq} \cup L^X_{pr}(L^X_{rr})^* L^X_{rq}$ by conditioning on whether the path from p to q goes through r.

This is easily converted to regular expressions by substituting $+$ for $\cup$.

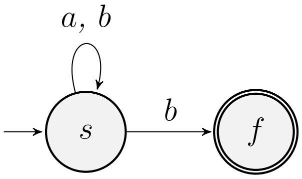
For the base case of $X = \varnothing$, observe that

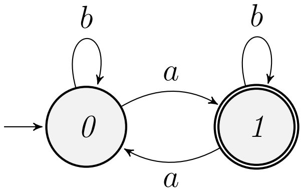
$$L^{\varnothing}_{pq} = \{a \in A + \epsilon \mid \widehat{\delta}(p,a) = q\}$$

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. □

Exercise I.36. *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

$$\begin{aligned} x_0 &= b \cdot x_0 + a \cdot x_1 \\ x_1 &= \epsilon + a \cdot x_0 + b \cdot x_1 \end{aligned}$$

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.

I.10 Myhill-Nerode Theorem

Lecture 9.
Tuesday
January 30

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 I.37. For any language $L \subseteq A^*$, we define the canonical equivalence relation $\equiv_L$ on A^* as

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

Exercise I.38. *Describe the equivalence classes for $L =$ "odd number of a's".*

Solution. L and L^c. ■

Exercise I.39. *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 I.40. *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 $X = \bigsqcup_{0 \leq m \leq n} \{a^n b^m\}$ and X^c. ■

Lecture 10.
Thursday
February 01

Definition I.41 (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: if $x \sim y$, then for all $a \in A$, $xa \sim ya$.
- $\sim$ refines L: 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 I.42. *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 $w = \varepsilon$). □

Proposition I.43. *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,
  $$\widehat{\delta}(q_0, w) = [w] \text{ for all } w \in A^* \tag{I.1}$$
  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 (I.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 I.44 (Equivalence 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 I.45. *Let L be a language over an alphabet A. Let $\sim$ be an MN-relation for L. Then $\sim$ refines $\equiv_L$.*

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

Theorem I.46 (Myhill-Nerode)**.** *$L \subseteq A^*$ is regular iff $\equiv_L$ is of finite index.*

Proof. Follows from propositions I.42 and I.43 and lemma I.45. 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. □

Lecture 11. Tuesday February 06

Definition I.47 (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 I.48 (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 $\sim$. How is this merging done? Let $p \sim q$. Then for all $a \in A$, we have that $[\delta(p, a)] = [\delta(q, a)]$. Thus we define $\delta'([p], a) = [\delta(p, a)]$.

Definition I.49. Let $\mathcal{A} = (Q, s, \delta, F)$ be a DFA and let $\sim$ be a stable partitioning of $\mathcal{A}$. We define $\mathcal{A}/\sim$ as follows.

$$\mathcal{A}/\sim \, = \Big(Q/\sim, [s], [p] \xrightarrow{a} [\delta(p, a)], \{[p] \mid p \in F\}\Big).$$

Proposition I.50. *Let $\sim$ be a stable partitioning of a DFA $\mathcal{A}$. Then $\mathcal{A}/\sim$ accepts the same language as $\mathcal{A}$.*

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

Definition I.51. 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 I.52. *$\approx_L$ 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 $a \in A$. □

Theorem I.53. *Let $\mathcal{A} = (Q, s, \delta, F)$ be a DFA for L with no unreachable states. Define $\approx$ as follows.*

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

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 $\widehat{\delta}(q, z) \in F$. □

Exercise I.54. *Run algorithm to compute $\approx$ for the DFA below*

Solution. ■

	s	p	q	r
s				
p	✓			
q		✓		
r	✓		✓	

I.10.1 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.

Lecture 11

$i = 0$ 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.

Chapter II

Context-Free Grammars

Lecture 12. Thursday February 08

II.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 naturally in syntax of programming language, parsing, compiling.

Examples.

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

- Let G_2 be the grammar given by the rules

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

  An example string in this grammar 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 by

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

  Then $L(G_3)$ is all palindromes.

- Let G_4 be given by

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

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

  - $\#_{\texttt{(}} w = \#_{\texttt{)}} w$, and
  - for each prefix u of w, $\#_{\texttt{(}} u \geq \#_{\texttt{)}} u$.

  Exercise II.1. *Derive the string* `((()())()())` *from* G_4.

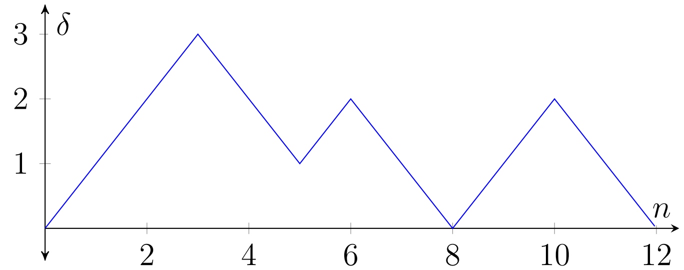
  Solution.

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

  ■

Lecture 12

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 II.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 $X \to \alpha$.

Notation. 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 II.3. Given a CFG $G = (N, A, S, P)$, we define the relation $\overset{n}{\Rightarrow}$ on $(N \cup A)^*$ inductively as follows:

- $\alpha \overset{0}{\Rightarrow} \alpha$;
- $\alpha \overset{1}{\Rightarrow} \beta$ 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 $\alpha \overset{n}{\Rightarrow} \gamma$ and $\gamma \overset{1}{\Rightarrow} \beta$.

We further define $\Rightarrow_G^*$ as the union of all $\overset{n}{\Rightarrow}$ 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 $L(G) = \{w \in A^* \mid S \Rightarrow_G^* w\}$.

II.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 II.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 $a(a+b)^*b$. □

II.3 Chomsky Normal Form

Lecture 13. Tuesday February 13

Definition II.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

$$\begin{aligned} X &\to YZ \\ X &\to a \end{aligned}$$

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

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

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

This can be converted into a CNF as follows

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

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 II.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 II.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 II.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. □

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. □

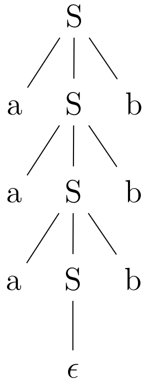
Lecture 14.
Thursday
February 15

II.4 Pumping Lemma

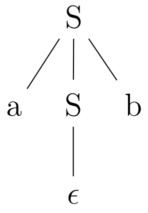
Theorem II.9 (Pumping lemma for CFLs)**.** *For every CFL L there is a constant $k \geq 0$ such that such that for any word z in L of length at least k, there are strings u, v, w, x, y such that*

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

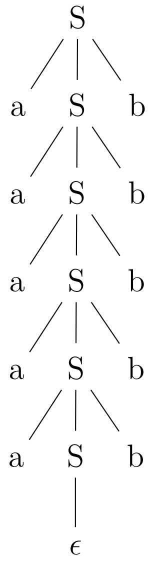
Consider a parse tree for any string. Note that subtrees hanging at the same non-terminal can be replaced by each other. For example, if



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 $i \geq 0$. □

Exercise II.10. *Show that the following languages are not context-free.*

Lecture 14

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

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 $uwy \neq a^p b^q a^p b^q$.

■

II.5 Closure properties

	Closed?
Union	✓
Intersection	✗
Complement	✗
Concatenation	✓

Table II.1: Closure properties of CFLs.

We have the following closure properties for CFLs.

Theorem II.11. *The class of context-free languages is closed under union and concatenation.*

Lecture 14

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 $L_1 \cup L_2$. □

Lecture 15.
Thursday
February 29

Lecture 16.
Thursday
February 29

Exercise II.12. *Consider the CFG G_1 again.*

$$\begin{aligned} S &\to XC \mid AY \\ X &\to aXb \mid ab \\ Y &\to bYc \mid bc \\ A &\to aA \mid a \\ C &\to cC \mid c \end{aligned}$$

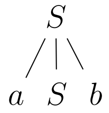
(i) What is the language of G_1?

(ii) What is the Parikh image of this language?

(iii) Write it as a semi-linear set.

Definition II.13 (Pump)**.** Let $G = (N, A, S, P)$ be a context free grammar in CNF. A *pump* of G is a derivation tree s with at least 2 nodes, such that $\text{yield}(s) = u \cdot \text{root}(s) \cdot v$ for some $u, v \in A^*$.

Example. For the grammar $S \to \epsilon \mid SS \mid aSb$, example pumps are



Definition II.14 (Basic pumps)**.** Let s and t be pumps of a context free grammar G. Then we say $s \lhd t$ iff t can be grown from s by replacing a non-terminal X in s by a pump of X.

A pump which is $\lhd$-minimal is called *basic.*

Lemma II.15. *The height of a basic pump is at most $2|N|$.*

Lecture 16

Proof. Let s be a basic pump. Let L be the longest path in s. If L has more than $2|N|+1$ nodes, then L has more than $2|N|$ nodes excluding the root. So by the pigeonhole principle, there are two nodes x and y in L with the same label. □

Definition II.16 (Order on parse trees)**.** Let s and t be derivation trees of terminal strings starting from the start symbol S. Then we say $s \leq t$ iff t can be grown from s by basic pumps whose non-terminals are comtained in those of s.

II.6 Pushdown Automata

CFG's were introduced by Noam Chomsky in 1956. Oettinger introduced pushdown automata in 1961. Chomsky, Schützenberger, and Evey showed equivalence of CFG's and PDA's in 1962.

A pushdown automaton reads a string from left to right and uses a stack to store information about the string it has read so far. Each step of the PDA looks like:

- read and pop the top-of-stack symbol;
- read current symbol and advance the read head;
- push in a string of symbols onto the stack;
- change state.

Thus each transition of a PDA is of the form

$$(p, a, X) \to (q, Y_1 Y_2 \ldots Y_k)$$

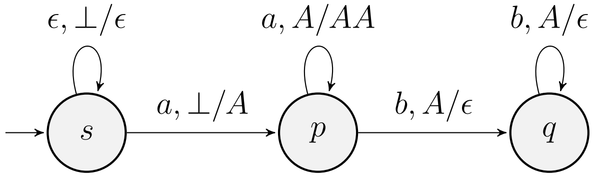
There are two commonly-used mechanisms of acceptance for PDA's:

- *Empty stack acceptance*: accept if the input is exhausted and the stack is empty.
- *Final state acceptance*: accept if the input is exhausted and the PDA is in a final state.

Example. An acceptance-by-empty-stack PDA for the language $\{a^n b^n \mid n \geq 0\}$ is

$$\begin{aligned}(s, \epsilon, \bot) &\to (s, \epsilon)\\ (s, a, \bot) &\to (p, A)\\ (p, a, A) &\to (p, AA)\\ (p, b, A) &\to (q, \epsilon)\\ (q, b, A) &\to (q, \epsilon)\end{aligned}$$

This is represented by the following diagram:



The automaton is in state p while reading the run of a's, and moves to state q when it starts reading the run of b's.

Definition II.17 (Pushdown automaton)**.** A *pushdown automaton* is a tuple $\mathcal{M} = (Q, A, \Gamma, s, \delta, \bot, F)$ where

- Q is a finite set of states,
- A is the input alphabet,
- Γ is the stack alphabet,
- $s \in Q$ is the start state,
- $\delta \subseteq Q \times (A \cup \{\epsilon\}) \times \Gamma \times Q \times \Gamma^*$ is a finite set of (non-deterministic) transitions,
- $\bot \in \Gamma$ is the bottom-of-stack symbol,
- $F \subseteq Q$ is the set of final states.

A *configuration* of $\mathcal{M}$ is of the form $(q, u, \gamma) \in Q \times A^* \times \Gamma^*$, which encodes the current state, the input read so far, and the stack contents.

The initial configuration of $\mathcal{M}$ is $(s, w, \perp)$ for input $w \in A^*$. We can define transition relations on configurations:

Definition II.18. For a 1-step transition of $\mathcal{M}$, we define

$$(p, au, X\beta) \overset{1}{\Rightarrow} (q, u, \alpha\beta)$$

iff $(p, a, X) \to (q, \alpha)$ is a transition in δ.

We recursively define $\overset{*}{\Rightarrow}$ as for NFAs. Then $\mathcal{M}$is said to accept a word w

- by empty stack iff $(s, w, \perp) \overset{*}{\Rightarrow} (q, \epsilon, \epsilon)$ for some $q \in F$, and
- by final state iff $(s, w, \perp) \overset{*}{\Rightarrow} (f, \epsilon, \gamma)$ for some $f \in F$ and $\gamma \in \Gamma^*$.

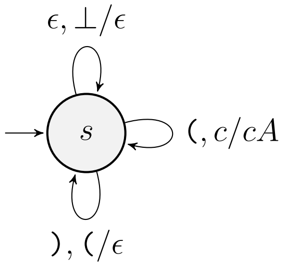
Exercise II.19. *Design PDA's for the following languages:*

- *Balanced parentheses.*
- $\{a, b\}^* \setminus \{ww \mid w \in \{a, b\}^*\}$.

Solution. For balanced parentheses, we can use the following PDA:

$$\begin{gathered}(s, \epsilon, \perp) \to (s, \epsilon) \\ (s, \texttt{(}, c) \to (s, cA) \\ (s, \texttt{)}, \texttt{(}) \to (s, \epsilon)\end{gathered}$$

or diagrammatically:



■

Lecture 17.
Tuesday
March 05

II.6.1 Equivalence of acceptance criteria

Theorem II.20. *Any language is accepted by a PDA via empty stack if and only if it is accepted by another PDA via final state.*

Proof. Suppose L is accepted by a PDA $\mathcal{M}$ via final state. Then we can add a new state Hole and a new ϵ-transition from every final state to Hole. For Hole, we add an ϵ-transition loop to itself that pops the stack and goes to the final state.

But what if the stack is emptied without reaching a final state? We introduce a new bottom-of-stack symbol $\perp\!\!\!\perp$. Let $\mathcal{M} = (Q, A, \Gamma, s, \delta, \perp, F)$. Define

$$\mathcal{M}' = (Q \cup \{s', \textsc{Hole}\}, A, \Gamma \cup \{\perp\!\!\!\perp\}, s', \delta', \perp\!\!\!\perp, \{\textsc{Hole}\})$$

where δ' has transitions from δ and the new transitions

$$\begin{aligned}(s', \epsilon, \perp\!\!\!\perp) &\to (s, \perp\perp\!\!\!\perp)\\ (f, \epsilon, X) &\to (\textsc{Hole}, \epsilon) \quad \text{for all } f \in F, X \in \Gamma \cup \{\perp\!\!\!\perp\}\\ (\textsc{Hole}, \epsilon, X) &\to (\textsc{Hole}, \epsilon) \quad \text{for all } X \in \Gamma \cup \{\perp\!\!\!\perp\}\end{aligned}$$

Suppose a word w is accepted by $\mathcal{M}$ via final state. Then once $\mathcal{M}$' reads w, it will land on a final state in $\mathcal{M}$ and will have a non-empty stack (since $\mathcal{M}$ transitions cannot pop the new $\perp\!\!\!\perp$ symbol). Then it will transition to Hole and empty the stack.

Suppose a word w is accepted by $\mathcal{M}$' via empty stack. The stack can only be emptied by the transitions from Hole at the end of the input tape. But to get to Hole, $\mathcal{M}$ must have been in a final state after reading the entire input. □

II.7 CFGs and PDAs

Theorem II.21 (Chomsky)**.** *The class of languages accepted by a PDA is precisely the class of context-free languages.*

CFG to PDA. Let $G = (N, A, S, P)$ be a CFG. Assume WLOG that all rules of G are of the form

$$X \to cB_1B_2 \ldots B_k$$

where $c \in A \cup \{\epsilon\}$ and $B_i \in N$. This can always be done, since for any $a \in A$ we can introduce a new non-terminal A' and a rule $A' \to a$.

We construct a PDA $\mathcal{M}$ that simulates a leftmost derivation of G on the given input. Define $\mathcal{M} = (\{s\}, A, N, s, \delta, S)$ where δ is given by

$$(s, c, X) \to (s, B_1 B_2 \dots B_k)$$

for any rule $X \to cB_1B_2 \dots B_k$ in P. Then $\mathcal{M}$ accepts $L(G)$ by empty stack. □

Exercise II.22. *Construct a PDA for the CFG*

$$\begin{aligned} &S \to \texttt{(}SR \mid SS \mid \epsilon \\ &R \to \texttt{)} \end{aligned}$$

Proof. We have the following transitions in δ:

$$\begin{aligned} (s, \texttt{(}, S) &\to (s, SR) \\ (s, \epsilon, S) &\to (s, SS) \\ (s, \epsilon, S) &\to (s, \epsilon) \\ (s, \texttt{)}, R) &\to (s, \epsilon) \end{aligned}$$

□

Lemma II.23. *For any PDA $\mathcal{M}$ that accepts via empty stack, there is a PDA $\mathcal{M}$' with a single state that accepts the same language via empty stack.*

Proof. Let $\mathcal{M} = (Q, A, \Gamma, s, \delta, \bot, F)$. Define

$$\mathcal{M}' = (\{Q\}, A, Q \times \Gamma \times Q, Q, \delta', (s, \bot), \varnothing)$$

where we will define δ' as follows. For every original transition

$$(p, a, X) \to (q, B_1 B_2 \dots, B_k),$$

we add the transitions

$$(Q, a, \langle p, X, r \rangle) \to (Q, \langle q B_1 q_1 \rangle \langle q_1 B_2 q_2 \rangle \dots \langle q_{k-1} B_k r \rangle)$$

for every $q_1, q_2, \dots, q_{k-1}, r \in Q$. In particular, for every original transition

$$(p, \epsilon, X) \to (q, \epsilon),$$

we add the transition

$$(Q, \epsilon, (p, X, r)) \to (Q, \epsilon)$$

for every $r \in Q$. □

Chapter III

Turing Machines

Lecture 18.
Thursday
March 07

III.1 A Brief History of Logic & Computability

In 1928, David Hilbert posed the Entscheidungsproblem – deciding validity of first-order logic formulas.

Kurt Gödel showed

- in 1929, the completeness of first-order logic;
- in 1931, the incompleteness of first-order arithmetic;
- in 1931, primitive recursive functions.

In 1936, Alan Turing proved the unsolvability of the Entscheidungsproblem using Turing machines. In the same year, Alonzo Church proved the same result using λ-calculus.

III.2 Introduction

A Turing machine reads a tape that is infinite to the right. The tape is divided into cells, each of which contains a symbol from a finite alphabet A.

In each step:

- In current state p, read current symbol under the tape head, say a.
- Change state to q.
- Replace a with b.

- Move the tape head left or right.

$$\delta(p, a) = (q, b, L/R)$$

The machine has special designated *accept* state t and *reject* state r. These states are assumed to be "sink" states. The machine accepts if it enters the accept state, and rejects if it enters the reject state. The machine never "falls off" the left end of the tape. That is, on reading '$\vdash$', it always moves right.

III.3 Turing Machines Formally

Lecture 18

Definition III.1 (Turing machine)**.** A *Turing machine* is a tuple

$$\mathcal{M} = (Q, A, \Gamma, s, \delta, \vdash, \flat, t, r)$$

where

- Q is a finite set of states;
- A is the finite input alphabet;
- $\Gamma \supseteq A$ is the finite tape alphabet;
- $s \in Q$ is the start state;
- $\delta : Q \times \Gamma \to Q \times \Gamma \times \{L, R\}$ is the (deterministic) transition function;
- $\vdash \in \Gamma \setminus A$ is the left end marker;
- $\flat \in \Gamma \setminus A$ is the blank symbol;
- $t \in Q$ is the accept state; and
- $r \in Q \setminus \{t\}$ is the reject state.

δ must follow the following constraints:

- For any state p, $\delta(p, \vdash) = (q, \vdash, R)$ for some $q \in Q$;
- For any $a \in \Gamma$, $\delta(t, a) = (t, b, \sigma)$ for some b and σ;
- For any $a \in \Gamma$, $\delta(r, a) = (r, b, \sigma)$ for some b and σ.

Definition III.2 (Configuration)**.** A *configuration* of a Turing machine $\mathcal{M}$ as above is a triple

$$(p, y\flat^\omega, n) \in Q \times \Gamma^\omega \times \mathbb{N}$$

which specifies that the machine is in state p, with "non-blank" tape contents y (which cannot end with $\flat$), and the read head positioned at the nth cell.

Remark. The initial configuration of $\mathcal{M}$ on input $w \in A^*$ is $(s, \vdash w\flat^\omega, 0)$.

Definition III.3 (Acceptance)**.** We define the 1-step transition of $\mathcal{M}$ as follows:

If $\delta(p,a) = (q,b,\sigma)$, and $z_n = a$, then

$$(p,z,n) \overset{1}{\Rightarrow} (q, s_b^n(z), n+\sigma).$$

where we identify L with -1 and R with $+1$.

We define $\overset{n}{\Rightarrow}$ and $\overset{*}{\Rightarrow}$ recursively as before.

We say that $\mathcal{M}$ *accepts* $w \in A^*$ if

$$(s, \vdash w\flat^\omega, 0) \overset{*}{\Rightarrow} (t,z,i)$$

for some $z \in \Gamma^*$, $i \in \mathbb{N}$.

We say that $\mathcal{M}$ *rejects* $w \in A^*$ if

$$(s, \vdash w\flat^\omega, 0) \overset{*}{\Rightarrow} (r,z,i)$$

for some z and i.

Definition III.4 (Halting)**.** A Turing machine $\mathcal{M}$ is said to *halt* on an input w if $\mathcal{M}$ either accepts or rejects w. Otherwise, we say that $\mathcal{M}$ *loops* on w.

The *language accepted by* $\mathcal{M}$ is

$$L(\mathcal{M}) = \{w \in A^* \mid \mathcal{M} \text{ accepts } w\}.$$

A language $L \subseteq A^*$ is said to be *recursively enumerable* if it is the language accepted by some Turing machine.

It is said to be *recursive* if it is the language accepted by some Turing machine that halts on all inputs.

Why do we use the terms “recursively enumerable” and “recursive”?

Proposition III.5 (Recursively enumerable)**.** *Let $\mathcal{M}$ be a Turing machine. Then there exists a procedure to list all strings in $L(\mathcal{M})$ (possibly with repetitions).*

Proof. □

III.4 Computability

Lecture 19.
Tuesday
March 12

Definition III.6 (Computability)**.** A function $f\colon \mathbb{N} \to \mathbb{N}$ is said to be *computable* if there exists a Turing machine M such that for all $n \in \mathbb{N}$, M given input $\vdash 0^n$ halts with output $\vdash 0^{f(n)}$.

We can view a function on the naturals as a language in the following manner.

Definition III.7 (Language of a function)**.** Let $f\colon \mathbb{N} \to \mathbb{N}$ be a function. The *language of f* is

$$L_f = \left\{0^n\#0^{f(n)} \mid n \in \mathbb{N}\right\}.$$

Theorem III.8. *A function $f\colon \mathbb{N} \to \mathbb{N}$ is computable if and only if its language L_f is recursive.*

Proof. Suppose f is computable. Let M_f be the Turing machine that computes f.

We define an M that decides L_f as follows. First, reject all strings not of the form $0^i\#0^j$. Then, simulate M_f on input 0^i (push the hash and 0^j forward as needed). Finally, check that the tape is in the form $0^j\#0^j$ and accept if so.

For the converse, suppose L_f is recursive, with Turing machine M. We define a Turing machine M_f that computes f as follows. We append a # to the input, then simulate M on the input. If M rejects, we append a 0 and repeat. When M finally accepts (which it must, since M halts on all inputs), we output the part of the tape after the #. □

III.5 Equivalent Models

III.5.1 Multiple Tapes

Consider a Turing machine with k tapes. That is, the transition function is of the form

$$\delta\colon Q \times \Gamma^k \to Q \times \Gamma^k \times \{L, R\}^k.$$

Theorem III.9. *A k-tape Turing machine is equivalent to a single-tape Turing machine.*

Proof. We give a recipe to simulate a k-tape Turing machine with a single-tape Turing machine. Let the k-tape machine be

$$M = (Q, A, \Gamma, s, \delta, \vdash, \flat, t, r).$$

Define a new alphabet $\Gamma' = (\Gamma)^k\{\vDash\}$. Define new states □

Consider a Turing machine with a two-way infinite tape. That is, the tape extends infinitely to the left and right. There is no left-end marker, and the input is given as $\flat^\omega x \flat^\omega$, with the read head initially on the first letter of x.

Theorem III.10. *A two-way infinite tape Turing machine is equivalent to a one-way infinite tape Turing machine.*

Proof. We can simulate a two-way tape using two one-way tapes. In each configuration we also store whether the read head is on the upper or lower tape. □

Consider a Turing machine with a two-dimensional tape. That is, the tape is a grid of cells, and the read head can move in four directions.

Theorem III.11. *A two-dimensional tape Turing machine is equivalent to a one-dimensional tape Turing machine.*

Proof. We can simulate a two-dimensional tape using two one-dimensional tapes. The first tape stores the cells in a diagonally ennumerated pattern, and the second stores the index of the current cell. The mapping from the current index to any of its neighbours is computable, so we can gather the number of cells to move the read head using another Turing machine. □

III.6 Non-deterministic Turing Machine

A non-deterministic Turing machine is a Turing machine with several possible transitions from each state. A word is accepted if there exists a sequence of transitions that leads to an accepting state.

Theorem III.12. *A non-deterministic Turing machine is equivalent to a deterministic Turing machine.*

Lecture 20

III.7 Universal Turing Machine

Lecture 20.
Thursday
March 14

We can construct a Turing Machine U that takes the encoding of a turing machine M, its input x, and simulates M on x. U accepts, rejects, or loops as M does on x.

III.7.1 Encoding

Represent a Turing Machine with

- states $\{1, 2, \dots, n\}$,
- tape alphabet $\{1, 2, \dots, m\}$,
- input alphabet $\{1, 2, \dots, k\}$ (where $k \geq m$),
- start, accept and reject states $s, t, r \in [n]$ (distinct),
- left end marker u and blank symbol $v \in \{k + 1, \dots, m\}$,
- transitions $(i_z, a_z) \to (j_z, b_z, s_z), z \in [l]$ L and R are taken to be 0 and 1 respectively

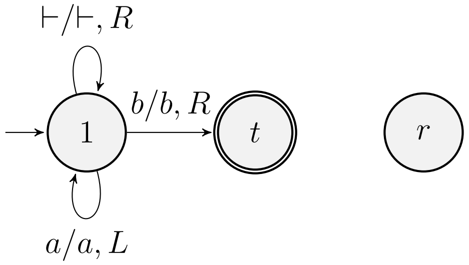
as

$$0^n10^m10^k10^s10^t10^r10^u10^v10^{i_1}10^{a_1}10^{j_1}10^{b_1}10^{s_1}\dots0^{i_l}10^{a_l}10^{j_l}10^{b_l}10^{s_l}.$$

Exercise III.13. *Give the Turing machine encoded by*

000100001001010010001000010000 1
01000101000100 1 0100100100100 1 010101010.

Solution.



■

III.7.2 How does the Universal Turing Machine work?

Use 3 tapes: for the input $M\#x$, for current tape contents of M, and for current state and position of head.

We assume that the input x is written on the tape in the form $0^{x_1}10^{x_2}1\dots0^{x_k}1$. The third tape is in the form 0^q10^n1, where q is the current state and n is the position of the head.

To simulate one step of M on x:

(i) Read the current position n from the third tape. Move to that position on the second tape (skip over n many 1s).

(ii) Move to the beginning of the third tape.

(iii) Find the transition which matches the current state and the symbol under the head.

(iv) Write the new state and the new symbol under the head on the third and second tapes.

(v) Increment or decrement the position on the third tape.

(vi) Compare q against t and r, and change state accordingly.

III.8 The Halting Problem

Fix an encoding enc of Turing Machines. Define the language

$$\mathsf{HP} = \{\mathrm{enc}(M)\#\,\mathrm{enc}(x) \mid M \text{ halts on } x\}.$$

As an example, the string $\mathrm{enc}(M)\#010010 \notin \mathsf{HP}$, where M is the Turing machine in the previous exercise.

What can we say about HP?

Theorem III.14. HP *is recursively enumerable.*

Proof. We have already constructed a universal Turing machine U. Modify it so that it accepts the input whether it is accepted or rejected by M. Then U accepts $\mathrm{enc}(M)\#\,\mathrm{enc}(x)$ if and only if M halts on x. □

Lecture 20

Theorem III.15. HP *is undecidable.*

Proof. Suppose there is a Turing machine $\mathcal{H}$ that decides HP. For each binary string γ, define the Turing machine M_γ as

- M, if $\gamma = \mathrm{enc}(M)$, and
- M_{loop}, if γ is not a valid encoding, where M_{loop} is a Turing machine that loops on every input.

For all binary strings γ and x, we can then compute

$$H(\gamma, x) = [M_\gamma \text{ halts on } x]$$

by running $\mathcal{H}$ on $\mathrm{enc}(M_\gamma)\#x$. What if x is not a valid encoding? We deem $H(\gamma, x) = 0$.

We can then define a new Turing machine N as follows:

(i) For a binary input x, transform it to $\mathrm{enc}(M_x)\#x$.

(ii) Run $\mathcal{H}$ on this transformed input.

(iii) If $\mathcal{H}$ accepts, then reject w. If $\mathcal{H}$ rejects, then accept w.

Then N halts on x if and only if M_x does not halt on x. Thus for all x, $N \neq M_x$. But N must occur in the list of all Turing machines, so we have a contradiction.

In particular, N halts on $\mathrm{enc}(N)$ iff $M_{\mathrm{enc}(N)}$ does not halt on $\mathrm{enc}(N)$. This is a contradiction. □

Corollary III.16. *The language* $\neg\mathsf{HP}$ *is not even recursively enumerable.*

We first prove the following lemma.

Lemma III.17. *If A and $\bar{A}$ are both recursively enumerable, then A is recursive.*

Proof. Let M and $\overline{M}$ be Turing machines that enumerate A and $\bar{A}$ respectively. Construct the Turing machine N with two tapes, that simulates M on one and $\overline{M}$ on the other, alternating between the two. Eventually, one of these must halt, and N will make the appropriate decision. □

Proof of Corollary. Suppose $\neg\mathsf{HP}$ is recursively enumerable. Then HP and $\neg\mathsf{HP}$ are both recursively enumerable. By the lemma, HP is recursive, which is a contradiction. □

Lecture 21.
Tuesday
March 19

Lecture 21

III.9 More Decidability Problems

Exercise III.18. *Is it decidable whether a given Turing machine has at least* 481 *states? Assume that the TM is given using our standard encoding.*

Solution. Yes. Simply check the appropriate section of zeros in the encoding. ■

Exercise III.19. *Is it decidable whether a given Turing machine takes more than* 481 *steps on input ϵ without halting?*

Solution. Yes. Simply simulate the TM for 481 steps using the universal TM, while storing the number of steps on a separate tape. ■

Exercise III.20. *Is it decidable whether a given Turing machine takes more than* 481 *steps on* some *input without halting?*

Solution. Yes. If the TM halts on some input w of length more than 481 in at most 481 steps, then it must halt on $w_1 w_2 \ldots w_{481}$ in at most 481 steps, since the read head cannot proceed to any letter beyond this.

Thus, we can simulate the TM on all inputs of length at most 481 for 481 steps, and reject if it halts on all of them, accepting otherwise. ■

Exercise III.21. *Is it decidable whether a given Turing machine takes more than* 481 *steps on* all *inputs without halting?*

Solution. Yes. Again check words of length up to 481. Reject if it halts on any of them, accept otherwise. ■

Exercise III.22. *Is it decidable whether a given Turing machine moves its head more than* 481 *cells away from the left-end marker, on input ϵ?*

Solution. Yes. The number of possible configurations without ever reading the 482nd cell is finite (precisely $n \cdot m^{481} \cdot 482$, where n and m are the number of states and tape symbols, respectively). Simulate the TM for one more step than this number of configurations.

If it ever reads the 482nd cell, accept. If not, then by the pigeonhole principle it must have entered a loop, and we reject. ■

Exercise III.23. *Is it decidable whether a given Turing machine accepts the null-string ϵ?*

Solution. No. This is equivalent to the halting problem.

Suppose N is a TM that decides whether a given TM M accepts ϵ. We can construct a new TM $\mathcal{H}$ as follows:

(i) On input $\mathrm{enc}(M)\#x$, construct a new TM $P_{M,x}$ that erases the input, writes x on the tape, and then simulates M on the input x. Accept if M ever halts on x.

(ii) Run N on $P_{M,x}$.

(iii) If N accepts, then M halts on x.
If N rejects, then M does not halt on x.

Then this $\mathcal{H}$ is the fabled decider for the halting problem. ■

Note that the language of $P_{M,x}$ is

$$L(P_{M,x}) = \begin{cases} A^* & \text{if } M \text{ halts on } x, \\ \varnothing & \text{otherwise.} \end{cases}$$

This will be useful for the coming exercises.

Exercise III.24. *Is it decidable whether a given Turing machine accepts any string at all? That is, is $L(M) \neq \varnothing$?*

Similarly, is it decidable whether a given Turing machine accepts all strings? That is, is $L(M) = A^*$?

Solution. Neither. If either of these were decidable, we could decide the halting problem by deciding whether $L(P_{M,x}) \neq \varnothing$ or equivalently $L(P_{M,x}) = A^*$. ■

Exercise III.25. *Is it decidable whether a given Turing machine accepts a finite set?*

Solution. No. Again, decide whether $L(P_{M,x})$ is finite. ■

Exercise III.26. *Is it decidable whether a given Turing machine accepts a regular set?*

Solution. No. Construct the Turing machine $Q_{M,x}$ which does the following:

- First, check if $x = a^n b^n c^n$ for some n. If not, reject.
- Then, simulate M on x. If it halts, accept.

Then

$$L(Q_{M,x}) = \begin{cases} \{a^n b^n c^n \mid n \in \mathbb{N}\} & \text{if } M \text{ halts on } x, \\ \varnothing & \text{otherwise.} \end{cases}$$

Then deciding whether $L(Q_{M,x})$ is regular ammounts to deciding whether M halts on x. ■

Exercise III.27. *Is it decidable whether a given Turing machine accepts a CFL?*

Solution. No. Same construction as before. ■

Exercise III.28. *Is it decidable whether a given Turing machine accepts a recursive set?*

Lecture 22. Tuesday March 26

III.10 Reductions

Lecture 23. Tuesday March 26

Definition III.29 (Computable map)**.** A map $\sigma\colon A^* \to B^*$ is *computable* if there exists a Turing machine $\mathcal{M}$ that, for every input $w \in A^*$, halts with output $\sigma(w)$.

Definition III.30 (Reduction)**.** Let $L \subseteq A^*$ and $M \subseteq B^*$ be languages. We say that L *reduces* to M, denoted $L \leq M$, if there exists a computable map $\sigma\colon A^* \to B^*$ such that

$$w \in L \iff \sigma(w) \in M.$$

Examples.

- Let $L = 2\mathbb{N}$ and $L' = \{x\#y\#m \mid x \equiv y \pmod{m}\}$. Then $L \leq L'$ via the computable map $n \mapsto n\#2\#0$.

  Is $L' \leq L$? Yes! L' is computable, so we can simply take the computable map

  $$w \mapsto \begin{cases} 0 & \text{if } w \in L', \\ 1 & \text{otherwise.} \end{cases}$$

- Let $L = \{M \mid M \text{ accepts } \epsilon\}$. Then $\mathsf{HP} \leq L$ by exercise III.23, where the map is $\mathrm{enc}(M)\#x \mapsto \mathrm{enc}(P_{M,x})$.
  $L \leq \mathsf{HP}$ by the map $\mathrm{enc}(M) \mapsto \mathrm{enc}(M)\#$.

Theorem III.31. *Let $L \leq M$.*

(i) If M is recursively enumerable, then so is L.

(ii) If M is recursive, then so is L.

Proof. For each case, let $\mathcal{M}$ be a Turing machine that accepts/deicdes M.

Let $\sigma \colon A^* \to B^*$ be the computable map such that $w \in L \iff \sigma(w) \in M$, computed by a Turing machine Σ.

Define the Turing machine $\mathcal{L}$ that first simulates Σ on input w, and then simulates $\mathcal{M}$ on $\sigma(w)$. □

Exercise III.32. *Show that the language*

$$L = \{M \mid M \textit{ accepts a regular language}\}$$

is not recursively enumerable.

Solution. We will show that $\neg\mathsf{HP} \leq L$. Use the computable map $M\#x \mapsto Q_{M,x}$, where $Q_{M,x}$ does the following:

- First simulate M on x.
- If M halts, check if the input is of the form $a^n b^n$.

Then

$$L(Q_{M,x}) = \begin{cases} \varnothing & \text{if } M \text{ does not halt on } x, \\ \{a^n b^n\} & \text{if } M \text{ halts on } x. \end{cases}$$

Then $M\#x \in \neg\mathsf{HP} \iff Q_{M,x} \in L$ so $\neg\mathsf{HP} \leq L$. Since $\neg\mathsf{HP}$ is not recursively enumerable, neither is L.

This is heavily inspired by exercise III.26. ■

Definition III.33 (Properties)**.** A *property* P of languages over an alphabet A is a subset of languages over A.

A property P is *trivial* if $P = \varnothing$ or $P = \{A^*\}$.

Examples.

- "is empty" is non-trivial.
- "contains ϵ" is non-trivial.

- “is accepted by a TM” is non-trivial.

A non-trivial property of r.e. languages is any property P such that P contains at least one r.e. language and at least one non-r.e. language.

From now on, “trivial” will mean “trivial with respect to recursively enumerable languages”.

Examples.

- “is empty” is non-trivial.
- “contains ϵ” is non-trivial.
- “is accepted by a TM” is trivial, since each recursively enumerable language is by definition accepted by a TM. So $P = \mathsf{RE}$.

Definition III.34 (Monotonicity)**.** A property P is *monotone* (with respect to RE) if for all languages L, L',

$$L \subseteq L' \implies L \in P \implies L' \in P.$$

Examples.

- “is infinite” is monotone.
- “is finite” is not monotone.

Definition III.35. For a property P, we define

$$L_P = \{\text{enc}(M) \mid L(M) \in P\}.$$

Theorem III.36 (Rice 1953)**.** *Any non-trivial property of recursively enumerable languages is undecidable. That is, if P is a non-trivial property of r.e. languages, then L_P is undecidable.*

Lecture 24.
Thursday
March 28

Theorem III.37 (Rice 1956)**.** *Any* non-monotone *property of recuresive enumerable languages is not even recursively enumerable. That is, if P is a non-monotone propertty of r.e. languages, then L_P is not even r.e.*

Remark. Languages that are not recursively enumerable are sometimes called *highly undecidable.*

Proof. We wish to reduce $\neg\mathsf{HP}$ to L_P. That is, $\neg\mathsf{HP} \le L_P$.

That is, we want a computable function $M\#x \overset{\sigma}{\mapsto} Q_{M,x}$ such that $Q_{M,x}$ is in L_P iff M does not halt on x.

We will define $Q_{M,x}$ as follows:

- Since P is non-monotone, there exist r.e. languages $A \subseteq B$ such that $A \in P$ but $B \notin P$.
- Take three tapes. Write the input on the first two tapes. Write x on the third tape.
- Run a Turing machine for A on the first tape, a TM for B on the second tape, and M on the third tape, all in parallel.
- Accept if the first tape accepts, *i.e.*, if $x \in A$.
- Also accept if M halts on x, and the second tape accepts. That is, if $M\#x \notin \neg\mathsf{HP}$ and $x \in B$.

Then the language accepted by $Q_{M,x}$ is

$$L(Q_{M,x}) = \begin{cases} A & \text{if } M \text{ does not halt on } x, \\ B & \text{if } M \text{ halts on } x. \end{cases}$$

□

III.11 Decidability Problems Regarding CFLs

Exercise III.38. *Is it decidable whether a given CFG accepts a non-empty language?*

Solution. Yes. One way is to use the bound for the height of minimal parse trees (**??**). Check all trees up to that height.

Another way is to check whether there is a way to reach a terminal string from each non-terminal. First, mark all terminals. Then, iteratively mark each non-terminal X such that $X \to \gamma$ is a rule with each symbol in γ marked. If the start symbol is marked at the end, the language is non-empty. ■

Lecture 24

Exercise III.39. *Is it decidable whether a given CFG accepts a finite language?*

Solution. Yes. Check all trees up to height $n + 2n$. ■

Exercise III.40. *Is it decidable whether a given CFG G is universal? That is, is $L(G) = A^*$?*

Solution. No!

We will reduce $\neg\mathsf{HP}$ to this problem. Highly undecidable!

Let $M = (Q, A, \Gamma, s, \delta, \vdash, \flat, t, r)$. Let $x = a_1 a_2 \dots a_n$ be an input to M.

We can represent a configuration of M as follows:

$$\begin{matrix} \vdash & b_1 & b_2 & b_3 & \dots & b_n \\ - & - & - & q & \dots & - \end{matrix}$$

Thus we define the double-decker alphabet

$$\Delta = \Gamma \times \{Q \cup \{-\}\} \cup \{\#\}$$

Define a *valid computation* of M on x as a string

$$c_0\#c_1\#\dots\#c_N\#$$

where each c_i is a valid configuration, c_0 is the initial configuration, c_N is a *halting* configuration, and $c_i \overset{1}{\Rightarrow} c_{i+1}$ for all i. Further, all c_i are of the same length, and minimal. That is, the length is either the length of the input, or the last cell that is ever read (basically every cell that matters). ■

Chapter IV

Gödel's Incompleteness Theorem

Lecture 25.
Tuesday
April 09

Theorem IV.1 (Godel (1931))**.** *There is no sound and complete proof system for arithmetc, i.e., first-order logic of natural numbers with addition and multiplication* $(\mathbb{N}, +, \cdot)$.

Definition IV.2 (Validity)**.** A formula φ is *valid* if it is true in every model.

Example. $\forall x(x = x)$ is valid. $(p \to q) \to (\neg q \to \neg p)$ is valid.

IV.1 Arithmetic

The first order logic of $(\mathbb{N}, +, \cdot)$ is defined by

- Domain $\mathbb{N} = \{0, 1, 2, \dots\}$
- Terms 0, 1, $0+1$, $1 \cdot x$, $x + y$, $x \cdot y$, etc
- Atomic formulas $t_1 = t_2$
- Formulas derived as
  - Atomic formulas,
  - Quantification $\forall x \varphi$, $\exists x \varphi$, where φ is a formula,
  - Connectives $\neg$, $\wedge$, $\vee$, etc.

We can build up more complex formulas using these building blocks.

- "x gives a quotient q and remainder r when divided by y" can be expressed as
$$\mathit{intdiv}(x, y, q, r) \coloneqq x = (q \cdot y) + r \wedge r < y.$$
- "y divides x" can be expressed as
$$\mathit{div}(x, y) \coloneqq \exists q \ \mathit{intdiv}(x, y, q, 0)$$
or more directly as
$$\mathit{div}(x, y) \coloneqq \exists q(x = q \cdot y).$$
- x is prime.
$$\mathit{prime}(x) \coloneqq \neg(x = 1) \wedge \forall y(\mathit{div}(y, x) \to y = 1 \vee y = x).$$
- x is a power of 2.
$$\mathit{power}_2(x) \coloneqq \forall y(\mathit{prime}(y) \wedge \mathit{div}(y, x) \to y = 2).$$
- Every number has a successor.
$$\forall x \exists y(y = x + 1).$$
- Every number has a predecessor.
$$\forall x \exists y(y + 1 = x).$$
- There are only finitely many primes.
$$\exists x \forall y(\mathit{prime}(y) \to y \leq x).$$
- There are infinitely many primes.
$$\forall x \exists y(y > x \wedge \mathit{prime}(y)).$$

Let $\mathsf{Th}(\mathbb{N}, +, \cdot)$ be the set of sentences of $\mathsf{FO}(\mathbb{N}, +, \cdot)$ that are true.

IV.2 Peano's Proof System for Arithmetic

Peano introduced the axioms

- $\forall x \neg(x + 1 = 0)$

Lecture 25

- $\forall x \forall y (x + 1 = y + 1 \to x = y)$
- $\forall x (x + 0 = x)$
- $\forall x \forall y \forall z (x + (y + 1) = (x + y) + 1)$
- $\forall x (x \cdot 0 = 0)$
- $\forall x \forall y \forall z (x \cdot (y + 1) = (x \cdot y) + x)$.
- Axiom schema of induction:

$$\varphi(0) \wedge \forall x \Big(\varphi(x) \to \varphi(x+1) \Big) \to \forall x \varphi(x).$$

Along with the inference rules from first-order logic, such as modus ponens, universal instantiation, etc., this gives a proof system for arithmetic.

Definition IV.3 (Proofs)**.** A *proof* of a sentence φ is a finite sequence of sentences $\varphi_1, \varphi_2, \dots, \varphi_n$ such that each φ_i is either an axiom or follows from previous sentences by an inference rule, and $\varphi_n = \varphi$.

Definition IV.4 (Soundness and Completeness)**.** A proof system is *sound* if every provable sentence is true, and *complete* if every true sentence is provable.

Lemma IV.5. *If* $\mathsf{Th}(\mathbb{N}, +, \cdot)$ *is not recursively enumerable, then there is no sound and complete proof system for arithmetic.*

Proof. Suppose there is a sound and complete proof system for arithmetic. Then we can decide whether a sentence is true by enumerating all proofs. □

Thus we need to show that $\mathsf{Th}(\mathbb{N}, +, \cdot)$ is not recursively enumerable, which of course we will do by reduction to the negative halting problem.

Let Δ be the alphabet defined for valcomp. Let $p > |\Delta|$ be a prime number. We can view each string $w \in \Delta^*$ as a number in base p. We will not assign the digit 0 to any letter.

Define valid computations as

$$\begin{aligned} \mathit{valcomp}_{M,x}(v) \coloneqq \exists c \exists d \Big(& \mathit{power}_p(c) \wedge \mathit{power}_p(d) \\ & \wedge \mathit{length}(v, d) \\ & \wedge \mathit{start}(v, c) \\ & \wedge \mathit{move}(v, c, d) \wedge \mathit{halt}(v, d) \Big). \end{aligned}$$

Define $\varphi_{M,x}$ as

$$\varphi_{M,x} \coloneqq \exists v \; valcomp_{M,x}(v).$$