KTH CSC VT 2008

DD2371 Automata Theory

Dilian Gurov

Lecture Outline

1. The lecturer
2. Introduction to automata theory
3. Course syllabus
4. Course objectives
5. Course organization
6. First definitions

1. Lecturer

Name: DilianGurov

E-mail: dilian@csc.kth.se

Phone: 08-790 81 98 (office)

Visiting address: Osquarsbacke 2, room4417

Research interests:
- Analysis of program behaviour
- Correctness: logics, compositionality

2. Introduction to Automata Theory

Automata are *abstract computing devices*.

Purpose: to capture the abstract notions of *computation* and *effective computability*.

We shall study and compare the computational power of three different classes of automata: Finite Automata, Pushdown Automata, and Turing Machines.

The comparison is made through the concept of *formal languages*.

Basic notions: state, nondeterminism, equivalence and minimization.

Algorithmic decidability.

Aim

The overall aim of the course is to provide students with a profound understanding of computation and effective computability through the abstract notion of automata and the language classes they recognize.

Along with this, the students will get acquainted with the important notions of state, nondeterminism and minimization.

3. Course Syllabus

Part I. *Finite Automata and Regular Languages*: determinisation, closure properties, regular expressions, state minimization, proving non-regularity with the Pumping lemma, Myhill-Nerode relations.

Part II. *Pushdown Automata and Context-Free Languages*: context-free grammars and languages, normal forms, closure properties, proving non-context-freeness with the Pumping lemma, pushdown automata.

Part III. *Turing Machines and Effective Computability*: Turing machines, recursive sets, Universal Turning machines, diagonalization, decidable and undecidable problems, reduction, other models of computability.

4. Course Objectives

After the course, the successful student will be able to perform the following *constructions*:

- Determinize and minimize automata;
- Construct an automaton for a given regular expression;
- Construct a pushdown automaton for a given context-free language;
- Construct a Turing machine deciding a given problem,

...be able to *prove* results such as:

- Closure properties of language classes;
- Prove that a language is not regular or context-free by using the Pumping Lemmata;
- Prove that a given context-free grammar generates a given context-free language;
- Prove undecidability of a problem by reducing from a known undecidable one,

as well as be able to *apply* the fundamental theorems of the course:

- Myhill-Nerode, Chomsky-Schützenberger, and Rice's theorems.

5. Course Organization

Credits: 4 points. Optional for graduate students.

Webpage: www.csc.kth.se/DD2371

Structure:
- 15 lectures/tutorials,
- 3 assignments,
- 1 written exam (open book).

Graduate students work in addition on a project.

Course book: Dexter Kozen, *Automata and Computability*, Springer, 1997. (Kårbokhandeln)

Course board: Group of student representatives. Any volunteers?

Strings and Sets

Definition 1 (Strings) *Basic notions:*

- *An alphabet is a finite set Σ of symbols.*
- *A string x over Σ is a finite-length sequence of elements of Σ. Concatenation $x \cdot y$ or simply xy. The set of all strings over Σ is denoted Σ^*.*
- *A language over Σ is a subset of Σ^*.*
- *The length of a string x is denoted $|x|$.*
- *The empty string is denoted ϵ.*
- *x is a prefix of y if $xz = y$ for some z.*

Definition 2 (Sets) *Basic notions:*

- *Set membership* $x \in A$
- *Set union*

$$A \cup B \stackrel{def}{=} \{x \mid x \in A \text{ or } x \in B\}$$

- *Set intersection*

$$A \cap B \stackrel{def}{=} \{x \mid x \in A \text{ and } x \in B\}$$

- *String set concatenation*

$$A \cdot B \stackrel{def}{=} \{xy \mid x \in A \text{ and } y \in B\}$$

- *Powers* A^n
- *Asterates* A^*, A^+

Finite Automata and Regular Languages

Definition 3 (DFA) *A deterministic finite automaton is a structure*

$$M \stackrel{def}{=} (Q, \Sigma, \delta, s, F)$$

where:

- Q - *finite set of states.*
- Σ - *input alphabet.*
- $\delta : Q \times \Sigma \to Q$ - *transition function. Induces* $\hat{\delta} : Q \times \Sigma^* \to Q$.
- $s \in Q$ - *initial state.*
- $F \subseteq Q$ - *final states.*

Graphical representation.

M accepts x if $\widehat{\delta}(s, x) \in F$.

The language accepted by M is

$$L(M) \stackrel{\text{def}}{=} \{x \in \Sigma^* \mid M \text{ accepts } x\}.$$

A language is called regular if it is accepted by some DFA.

Example 1 *Build* M_1 *accepting* $L_1 \stackrel{def}{=} \{ax \mid x \in \Sigma^*\}$ *and* M_2 *accepting* $L_2 \stackrel{def}{=} \{xb \mid x \in \Sigma^*\}$.

Exrecise: HW 1.1(a)

Closure properties of regular languages.

The Complement Construction

Let $M = (Q, \Sigma, \delta, s, F)$ be a DFA.

The complement of M is defined as the automaton:

$$\overline{M} \stackrel{\text{def}}{=} (Q, \Sigma, \delta, s, \overline{F})$$

Theorem 1 $L(\overline{M}) = \overline{L(M)}$

Hence regular languages are closed under complement.

The Product Construction

Let
$M_1 = (Q_1, \Sigma, \delta_1, s_1, F_1)$ and
$M_2 = (Q_2, \Sigma, \delta_2, s_2, F_2)$ be two DFAs.

The product of M_1 and M_2 is defined as the automaton:

$$M_1 \times M_2 \stackrel{\text{def}}{=} (Q_1 \times Q_2, \Sigma, \delta, \langle s_1, s_2 \rangle, F_1 \times F_2)$$

where

$$\delta((q_1, q_2), a) \stackrel{\text{def}}{=} (\delta_1(q_1, a), \delta_2(q_2, a))$$

Theorem 2 $L(M_1 \times M_2) = L(M_1) \cap L(M_2)$

Hence regular languages are closed under intersection.

Exercise: HW 1.2(a)

Home exercises: HW 1.1, HW 1.2, ME 2.

Nondeterministic Finite Automata

Example 2 $L \stackrel{def}{=} \{xab \mid x \in \Sigma^*\}$

Definition 4 (NFA) *A nondeterministic finite automaton is a structure*

$$N \stackrel{def}{=} (Q, \Sigma, \Delta, S, F)$$

where:

- $\Delta : Q \times \Sigma \to 2^Q$ - *transition function*, *inducing* $\hat{\Delta} : 2^Q \times \Sigma^* \to 2^Q$.

- $S \subseteq Q$ - *initial states*.

N accepts x if $\hat{\Delta}(S, x) \cap F \neq \emptyset$.

Theorem 3 *The languages accepted by NFAs are the regular languages.*

From DFA to NFA

For $M = (Q, \Sigma, \delta, s, F)$ we construct

$$N \stackrel{\text{def}}{=} (Q, \Sigma, \Delta, \{s\}, F)$$

where

$$\Delta(q, a) \stackrel{\text{def}}{=} \{\delta(q, a)\}$$

Theorem 4 $L(N) = L(M)$

Hence every regular language is accepted by some NFA.

From NFA to DFA: The Subset Construction

For $N = (Q_N, \Sigma, \Delta_N, S_N, F_N)$ we construct

$$M \stackrel{\text{def}}{=} (Q_M, \Sigma, \delta_M, s_M, F_M)$$

so that:

- $Q_M \stackrel{\text{def}}{=} 2^{Q_N}$
- $\delta_M(A, a) \stackrel{\text{def}}{=} \hat{\Delta}_N(A, a)$
- $s_M \stackrel{\text{def}}{=} S_N$
- $F_M \stackrel{\text{def}}{=} \{A \in Q_M \mid A \cap F_N \neq \emptyset\}$.

Theorem 5 $L(M) = L(N)$

Hence every language accepted by an NFA is regular.

Exercises: ME 4(a), HW 2.2

NFA extension: ϵ-transitions.

More closure properties.

Home exercises: HW 2.1, ME 3, ME 5, ME 6 (!), ME 10 (a, b).

Pattern Matching

For any pattern α:

$$L(\alpha) = \{x \in \Sigma^* \mid x \text{ matches } \alpha\}$$

Atomic patterns:

- a - exactly by $a \in \Sigma$

- ε - exactly by ϵ

- $\oslash$ - by no string

- # - by any symbol in Σ

- @ - by any string in Σ^*

Compound patterns, matched by x:

- $\alpha + \beta$ - if x matches α or β

- $\alpha \cap \beta$ - if x matches α and β

- $\alpha \cdot \beta$ - if for some y, z such that $x = y \cdot z$,
  y matches α and z matches β

- $\sim \alpha$ - if x does not match α

- α^* - if x matches α^n for some $n \geq 0$

- α^+ - if x matches α^n for some $n > 0$

Example 3 *All strings:*
- *of the shape* $xaybz$
- *with no occurrence of* a

Pattern Matching and Regular Expressions

Regular expressions:
- atomic patterns: a, ε, $\oslash$
- operators: $+$, $\cdot$, *

Theorem 6 *Let $A \subseteq \Sigma^*$. The statements:*
(i) $A = L(M)$ for some finite automaton M
(ii) $A = L(\alpha)$ for some pattern α
(iii) $A = L(\alpha)$ for some regular expression α
are equivalent.

Proof.
(i) $\Rightarrow$ (iii) - next lecture
(iii) $\Rightarrow$ (ii) - trivial
(ii) $\Rightarrow$ (i) - by structural induction:
- holds for the basic patterns, and
- is preserved by the operators. []

Example 4 *Automaton for $(ab)^* + (bc)^*$*

Regular Expressions and Finite Automata

Let $N = (Q, \Sigma, \Delta, S, F)$ be an NFA.

For $X \subseteq Q$ and $u, v \in Q$, let α_{uv}^{X} denote the regular expression representing all paths in N from u to v with intermediate nodes in X.

Then for

$$e_N \stackrel{\text{def}}{=} \sum_{\substack{s \in S \\ f \in F}} \alpha_{sf}^{Q}$$

we have $L(e_N) = L(N)$.

We can build α_{uv}^{X} inductively:

$$\left|\begin{array}{l} \alpha_{uv}^{\emptyset} \stackrel{\text{def}}{=} \begin{cases} \oslash + \sum\limits_{a \in \lambda(u,v)} a & \text{if } u \neq v \\ \varepsilon + \sum\limits_{a \in \lambda(u,v)} a & \text{otherwise} \end{cases} \\ \\ \alpha_{uv}^{X} \stackrel{\text{def}}{=} \alpha_{uv}^{X-\{q\}} + \alpha_{uq}^{X-\{q\}} (\alpha_{qq}^{X-\{q\}})^{*} \alpha_{qv}^{X-\{q\}} \end{array}\right.$$

where $\lambda(u,v) \stackrel{\text{def}}{=} \{a \in \Sigma \mid v \in \Delta(u,a)\}$.

Example 5 *Automaton:*

- *initial state* q_0*, final state* q_1*,*
- a*-edge from* q_0 *to* q_0*,* b*-edge from* q_0 *to* q_1*.*

Kleene Algebra and Regular Expressions

Equivalence $\alpha \equiv \beta$ when $L(\alpha) = L(\beta)$.

Axioms:

$$
\begin{array}{lrcl}
(A_1) & \alpha + (\beta + \gamma) & \equiv & (\alpha + \beta) + \gamma \\
(A_2) & \alpha + \beta & \equiv & \beta + \alpha \\
(A_3) & \alpha + \varnothing & \equiv & \alpha \\
(A_4) & \alpha + \alpha & \equiv & \alpha \\
 & & & \\
(A_5) & \alpha \cdot (\beta \cdot \gamma) & \equiv & (\alpha \cdot \beta) \cdot \gamma \\
(A_6) & \alpha \cdot \varepsilon & \equiv & \alpha \\
(A_7) & \alpha \cdot \varnothing & \equiv & \varnothing \\
 & & & \\
(A_8) & \alpha \cdot (\beta + \gamma) & \equiv & \alpha \cdot \beta + \alpha \cdot \gamma \\
(A_9) & (\beta + \gamma) \cdot \alpha & \equiv & \beta \cdot \alpha + \gamma \cdot \alpha \\
 & & & \\
(A_{10}) & \varepsilon + \alpha \cdot \alpha^* & \equiv & \alpha^*
\end{array}
$$

Rules of Equational Logic:
- equivalence rules:
  reflexivity, symmetry, transitivity
- substitution rule

Example 6 $\varepsilon + \alpha^* \equiv \alpha^*$

Other derived laws:

$$\begin{array}{lrcl}
(L_1) & (\alpha \cdot \beta)^* \cdot \alpha & \equiv & \alpha \cdot (\beta \cdot \alpha)^* \\
(L_2) & \alpha^* \cdot \beta^* \cdot \alpha^* & \equiv & (\alpha + \beta)^* \\
(L_3) & \varepsilon + \alpha^* & \equiv & \alpha^* \\
(L_4) & \alpha \cdot \alpha^* & \equiv & \alpha^* \cdot \alpha
\end{array}$$

Exercises: HW 3, ME 11–20.

DFA State Minimization

Observable behaviour of a system. Distinguishing experiment. Indistinguishability by experiment is an equivalence! Forms the basis for minimization.

For two DFAs: what is a distinguishing experiment? The equivalence is:

$$M_1 \approx M_2 \overset{\text{def}}{\iff} L(M_1) = L(M_2)$$

One can apply the same reasoning to states. Equivalence of states:

$$q_1 \approx q_2 \overset{\text{def}}{\iff} \forall x \in \Sigma^*.\, (\widehat{\delta}_1(q_1, x) \in F_1 \Leftrightarrow \widehat{\delta}_2(q_2, x) \in F_2)$$

Minimization of a DFA by collapsing equivalent states: the quotient construction. Example.

Equivalence class of q:

$$[q] \overset{\text{def}}{=} \left\{ q' \in Q \mid q' \approx q \right\}$$

The Quotient Construction

For $M = (Q, \Sigma, \delta, s, F)$ we construct

$$M/\approx \stackrel{\text{def}}{=} (Q', \Sigma, \delta', s', F')$$

so that:

- $Q' \stackrel{\text{def}}{=} \{[q] \mid q \in Q\} = Q/\approx$
- $\delta'([q], a) \stackrel{\text{def}}{=} [\delta(q, a)]$
  sound because $p \approx q \Rightarrow \delta(p, a) \approx \delta(q, a)$
- $s' \stackrel{\text{def}}{=} [s]$
- $F' \stackrel{\text{def}}{=} \{[q] \mid q \in F\} = F/\approx$
  sound because $p \approx q \wedge q \in F \Rightarrow p \in F$

Theorem 7 $L(M/\approx) = L(M)$

Minimality of $M/\approx$:

- w.r.t. M: $[p] \approx [q] \Rightarrow [p] = [q]$
  because $q \approx [q]$

- w.r.t. $L(M)$: yes, in later lecture.

Minimization Algorithms

Equivalence/undistinguishability of states:

$$q_1 \approx q_2 \overset{\text{def}}{\iff} \forall x.\, (\hat{\delta}_1(q_1, x) \in F_1 \Leftrightarrow \hat{\delta}_2(q_2, x) \in F_2)$$

Stratified equivalence/undistinguishability: "within k steps":

$$q_1 \approx_k q_2 \overset{\text{def}}{\iff} \forall x : |x| \leq k.\, (\hat{\delta}_1(q_1, x) \in F_1 \Leftrightarrow \hat{\delta}_2(q_2, x) \in F_2)$$

Then we have:

$$q_1 \approx q_2 \Leftrightarrow \forall k.\, q_1 \approx_k q_2$$

But actually, if q_1 and q_2 are distinguishable, then there is a distinguishing sequence of length less than $|Q|$:

$$q_1 \approx q_2 \Leftrightarrow \forall k \leq |Q| - 1.\, q_1 \approx_k q_2$$

We can compute these "approximants" iteratively:

$$\begin{aligned} p \approx_0 q &\overset{\text{def}}{\iff} p \in F \Leftrightarrow q \in F \\ p \approx_{i+1} q &\overset{\text{def}}{\iff} p \approx_i q \text{ and } \forall a \in \Sigma.\, \delta(p, a) \approx_i \delta(q, a) \end{aligned}$$

Define, for any relation $R \subseteq Q \times Q$, the mapping $f : 2^{Q \times Q} \to 2^{Q \times Q}$ by:

$$p \, f(R) \, q \stackrel{\text{def}}{\iff} \forall a \in \Sigma. \, \delta(p, a) \, R \, \delta(q, a)$$

Using this notation, we can redefine:

$$\begin{array}{rcl} \approx_0 & \stackrel{\text{def}}{=} & (F \times F) \cup ((Q - F) \times (Q - F)) \\ \approx_{i+1} & \stackrel{\text{def}}{=} & \approx_i \cap f(\approx_i) \end{array}$$

Algorithm:

$E :=\approx_0$;
while $E \neq E \cap f(E)$ **do** $E := E \cap f(E)$

Example.

Exercises: HW 4.3, ME 47.

Myhill–Nerode Theorem

Let $M = (Q, \Sigma, \delta, s, F)$ be a DFA. Recall:

$$p \approx q \overset{\text{def}}{\iff} \forall x \in \Sigma^*.\, \hat{\delta}(p, x) \in F \Leftrightarrow \hat{\delta}(q, x) \in F$$

$$x \in L(M) \overset{\text{def}}{\iff} \hat{\delta}(s, x) \in F$$

Question: Is $M/\!\approx$ the least DFA for $L(M)$?

Answer: Yes.

We need an abstract notion of state in terms of strings.

state - a maximal set of histories undistinguishable by experiment!

..., i.e., an equivalence class of Σ^* w.r.t. undistinguishablity.

- what is a history?
- what is a distinguishing experiment?
- is undistinguishability an equivalence?

In our case:

- histories are strings,
- a distinguishing experiment is appending some string to both histories and checking membership to L,
- undistinguishability is an equivalence.

Let $L \subseteq \Sigma^*$. Define $\equiv_L \subseteq \Sigma^* \times \Sigma^*$ by:

$$x_1 \equiv_L x_2 \stackrel{\text{def}}{\iff} \forall y \in \Sigma^*.(x_1 \cdot y \in L \Leftrightarrow x_2 \cdot y \in L)$$

Reformulated question: Do $Q/\approx$ correspond to $\Sigma^*/_{\equiv_{L(M)}}$? Yes! Indeed:

$\hat{\delta}$ establishes the correspondence $\Sigma^* \leftrightarrow Q$. We have:

$$\begin{array}{ll} & \hat{\delta}_{M/\approx}(s, x_1) = \hat{\delta}_{M/\approx}(s, x_2) \\ \Leftrightarrow & \hat{\delta}_M(s, x_1) \approx \hat{\delta}_M(s, x_2) \\ \Leftrightarrow & \forall y \in \Sigma^*.(\hat{\delta}_M(\hat{\delta}_M(s, x_1), y) \in F \Leftrightarrow \hat{\delta}_M(\hat{\delta}_M(s, x_2), y) \in \\ \Leftrightarrow & \forall y \in \Sigma^*.(\hat{\delta}_M(s, x_1 \cdot y) \in F \Leftrightarrow \hat{\delta}_M(s, x_1 \cdot y) \in F) \\ \Leftrightarrow & \forall y \in \Sigma^*.(x_1 \cdot y \in L(M) \Leftrightarrow x_2 \cdot y \in L(M)) \\ \Leftrightarrow & x_1 \equiv_{L(M)} x_2 \end{array}$$

One can even directly construct the minimal automaton for L as:

$$M_L \stackrel{\text{def}}{=} (\Sigma^*/_{\equiv_L}, \Sigma, \delta_L, [s]_L, L/_{\equiv_L})$$

where $\delta_L([x]_L, a) \stackrel{\text{def}}{=} [xa]_L$.

Theorem 8 (Myhill-Nerode Theorem)

L is regular $\Leftrightarrow$ $\Sigma^*/_{\equiv_L}$ is finite.

Example 7 *Construct M_L for $L(a^*b^*)$.*

Exercise: ME 55.

Limitations of Finite Automata

Theorem 9 (Cantor) *Let S be a set. There is no bijection:*

$$f : S \to 2^S$$

Proof. Let $f : S \to 2^S$ be a mapping. Define the set:

$$A \stackrel{\text{def}}{=} \{s \in S \mid s \notin f(s)\}$$

Assume f is a bijection. Then $A = f(s)$ for some $s \in S$. But then:

$$\begin{array}{rcl} s \in f(s) & \Leftrightarrow & s \in A \\ & \Leftrightarrow & s \notin f(s) \end{array}$$

which is a contradiction. Hence f is not a bijection. []

For example, there is no bijection

$$f : \Sigma^* \to 2^{\Sigma^*}$$

from strings over Σ to languages over Σ.

Theorem 10 (DG) *Let $\mathcal{M}$ be a class of accepting automata. Let $\widehat{\ }: \mathcal{M} \to \Sigma^*$ be an (injective) encoding. Then there is a language $L \subseteq \Sigma^*$ which is not accepted by any $M \in \mathcal{M}$.*

Proof. Define the set:

$$L \stackrel{\text{def}}{=} \left\{ \widehat{M} \in \Sigma^* \mid M \text{ does not accept } \widehat{M} \right\}$$

Assume there is $M \in \mathcal{M}$ such that $L(M) = L$. But then:

$$\begin{aligned} M \text{ accepts } \widehat{M} &\Leftrightarrow \widehat{M} \in L(M) \\ &\Leftrightarrow \widehat{M} \in L \\ &\Leftrightarrow M \text{ does not accept } \widehat{M} \end{aligned}$$

which is a contradiction. Hence there is no such M. []

Pumping Lemma

Consider the language:

$$B \stackrel{\text{def}}{=} \{a^n b^n \mid n \geq 0\}$$

It is not regular, and to recognize it we need unbounded memory!

For, if we assume otherwise, then there must be a DFA M so that:

$$L(M) = B$$

Then, take the string $a^k b^k$ for some $k > |Q|$. It must be that:

$$\overbrace{aaaaa}^{u} \cdot \overbrace{aaaa}^{v} \cdot \overbrace{aaa \cdot bbbbbbbbbbb}^{w}$$

$$\uparrow \quad \uparrow \quad \uparrow \quad \uparrow$$

$$s \quad q \quad q \quad f$$

Then $u \cdot w$ must also be accepted, and all other strings of the form $u \cdot v^i \cdot w$ as well. But none of these strings, with the exception of uvw itself, is in B!

Theorem 11 (Pumping Lemma) *Let A be regular. Then:*

$$\exists k \geq 0.$$
$$\quad \forall x, y, z \in \Sigma^* : xyz \in A \wedge |y| \geq k.$$
$$\quad\quad \exists u, v, w \in \Sigma^* : y = uvw \wedge v \neq \epsilon.$$
$$\quad\quad\quad \forall i \geq 0.$$
$$\quad\quad\quad\quad xuv^iwz \in A$$

Or, in contrapositive form:

If for $A \subseteq \Sigma^*$:

$$\forall k \geq 0.$$
$$\quad \exists x, y, z \in \Sigma^* : xyz \in A \wedge |y| \geq k.$$
$$\quad\quad \forall u, v, w \in \Sigma^* : y = uvw \wedge v \neq \epsilon.$$
$$\quad\quad\quad \exists i \geq 0.$$
$$\quad\quad\quad\quad xuv^iwz \notin A$$

then A is <u>not</u> regular.

Exercises: L 12, HW 4.1, ME 35-45.

Context-Free Grammars and Lanuages

Finite-state vs. finite-control.

Grammars vs. automata.

Definition 5 (CFG) *A context-free grammar is a structure*

$$G \stackrel{def}{=} (N, \Sigma, P, S)$$

where:

- N - *finite set of non-terminals.*
- Σ - *finite set of terminals.*
- $P \subseteq N \times (N \cup \Sigma)^*$ - *finite set of productions of the shape* $A \rightarrow \alpha$.
- $S \in N$ - *start symbol.*

Example 8 $S \to \epsilon \mid aSb$

One-step derivability:

$$\alpha \to_G \beta$$

if $\alpha = \alpha_1 A \alpha_2$ and $\beta = \alpha_1 \gamma \alpha_2$ for some α_1, α_2 and production $(A \to \gamma) \in P$.

A sentential form of G is a string over $(N \cup \Sigma)^*$ derivable from S.

A sentence of G is a string over Σ^* derivable from S.

The language of G is the set of all its sentences:

$$L(G) \stackrel{\text{def}}{=} \left\{ x \in \Sigma^* \mid S \stackrel{*}{\to}_G x \right\}$$

A language is context-free if it is the language of some CFG.

Balanced Parentheses

Let $N = \{S\}$ and $\Sigma = \{[,]\}$.

Define the functions:

$$L(x) \stackrel{\text{def}}{=} \#[(x)$$
$$R(x) \stackrel{\text{def}}{=} \#](x)$$

A string x of parentheses is balanced if:

- $L(x) = R(x)$
- $L(y) \geq R(y)$ for all prefixes y of x

Theorem 12 *The set of all balanced strings of parentheses is a context-free language.*

Proof. Consider the CFG

$$S \to \epsilon \mid [S] \mid SS$$

We show that a string of parentheses is balanced exactly when it is a sentence of this grammar.

Normal Forms

Definition 6 *Let G be a CFG.*

- *G is in Chomsky normal form (CNF) if all its productions are of the form:*

$$A \to BC \text{ or } A \to a$$

- *G is in Greibach normal form (GNF) if all its productions are of the form:*

$$A \to aB_1B2 \dots B_k$$

Theorem 13 *For every CFG G there is a CFG G' in CNF and a CFG G'' in GNF such that:*

$$L(G') = L(G'') = L(G) - \{\epsilon\}$$

Pumping Lemma for CFL

Consider the language:

$$L \stackrel{\text{def}}{=} \{a^n b^n \mid n \geq 0\}$$

and the grammar G given by

$$S \to \epsilon \mid aSb$$

generating L. Consider the string $aabb \in L(G)$. It is generated by the parse tree:

We have a path where the non-terminal S occurs more than once. Take the last two occurrences. These generate two subtrees, T and t. But replacing T for t yields another parse tree for a word in L! So, $aaabbb$, $aaaabbbb$, . . . , are also in L.

If $x \in L$ is sufficiently long, there is a parse tree where some non-terminal repeats along a path! For G in CNF this is guaranteed for $|x| \geq 2^{|N|+1}$.

Theorem 14 (Pumping Lemma for CFL)
Let A be context-free. Then:

$$\begin{array}{l}\exists k \geq 0. \\ \quad \forall z \in A : |z| \geq k. \\ \quad\quad \exists u, v, w, x, y \in \Sigma^* : \\ \quad\quad z = uvwxy \wedge vx \neq \epsilon \wedge |vwx| \leq k. \\ \quad\quad\quad \forall i \geq 0. \\ \quad\quad\quad\quad uv^i wx^i y \in A\end{array}$$

Or, in contrapositive form:

If for $A \subseteq \Sigma^*$:

$$\begin{array}{l}\forall k \geq 0. \\ \quad \exists z \in A : |z| \geq k. \\ \quad\quad \forall u, v, w, x, y \in \Sigma^* : \\ \quad\quad z = uvwxy \wedge vx \neq \epsilon \wedge |vwx| \leq k. \\ \quad\quad\quad \exists i \geq 0. \\ \quad\quad\quad\quad uv^i wx^i y \notin A\end{array}$$

then A is <u>not</u> context-free.

Exercises: HW 5.1-3, ME 72, 84.

Pushdown Automata

Definition 7 (NPDA) *A nondeterministic pushdown automaton is a structure*

$$M \stackrel{def}{=} (Q, \Sigma, \Gamma, \delta, s, \bot)$$

where:

- Q - *finite set of control states.*
  Σ - *a finite input alphabet.*
  Γ - *a finite stack alphabet.*

- $\delta \subseteq (Q \times \Gamma) \times \Sigma \times (Q \times \Gamma^*)$ - *a finite set of labelled productions of the shape:*

$$\langle q_1, A \rangle \stackrel{a}{\hookrightarrow} \langle q_2, \gamma \rangle$$

- $s \in Q$ - *start state.*
  $\bot \in \Gamma$ - *initial stack symbol.*

A state of a NPDA consists of a control state and the state of the stack: configurations $Q \times \Gamma^*$.

Initial configuration: $\langle s, \bot \rangle$.

Let $\Delta \subseteq (Q \times \Gamma^*) \times \Sigma \times (Q \times \Gamma^*)$ be the least labelled transition relation closed under the *prefix* rule:

$$\langle q_1, A \cdot \gamma' \rangle \xrightarrow{a} \langle q_2, \gamma \cdot \gamma' \rangle \text{ if } \langle q_1, A \rangle \overset{a}{\hookrightarrow} \langle q_2, \gamma \rangle \in \delta$$

inducing $\hat{\Delta} \subseteq (Q \times \Gamma^*) \times \Sigma^* \times (Q \times \Gamma^*)$.

M accepts x if $\langle s, \bot \rangle \xrightarrow{x} \langle q, \epsilon \rangle$.

The language accepted by M is

$$L(M) \stackrel{\text{def}}{=} \{x \in \Sigma^* \mid M \text{ accepts } x\}.$$

Example 9 $L(M) = \{a^n b^n \mid n \geq 1\}$

$Q \stackrel{def}{=} \{q_+, q_-\}$

$\Sigma \stackrel{def}{=} \{a, b\}$

$\Gamma \stackrel{def}{=} \{S, A\}$

$$\delta \stackrel{def}{=} \left\{ \begin{array}{l} \left\langle q_+, S \right\rangle \stackrel{a}{\hookrightarrow} \left\langle q_+, A \right\rangle, \\ \left\langle q_+, A \right\rangle \stackrel{a}{\hookrightarrow} \left\langle q_+, AA \right\rangle, \\ \left\langle q_+, A \right\rangle \stackrel{b}{\hookrightarrow} \left\langle q_-, \epsilon \right\rangle, \\ \left\langle q_-, A \right\rangle \stackrel{b}{\hookrightarrow} \left\langle q_-, \epsilon \right\rangle \end{array} \right\}$$

$s \stackrel{def}{=} q_+$

$\perp \stackrel{def}{=} S$

This PDA is actually deterministic. How about a nondeterministic PDA with one control state?

From CFG to NPDA

From a CFG $G = (N, \Sigma, P, S)$ in GNF, we construct canonically a NPDA

$$M \stackrel{\text{def}}{=} (\{q\}, \Sigma, N, \delta, q, S)$$

where:

- q - single control state,
  the input alphabet of M are the terminals of G,
  the stack alphabet of M are the non-terminals of G,
  the initial stack symbol of M is the start symbol of G,

- $\langle q, A\rangle \stackrel{a}{\hookrightarrow} \langle q, \gamma\rangle$ in δ iff $A \to a \cdot \gamma$ in P.

Theorem 15 *L(M) = L(G).*

Exercises: HW 5.4, 6.2, 6.4, ME 76.

Homework: HW 7.2.

Parsing

Parsing is the process of producing a parse tree for a sentence w.r.t. a grammar.

Example 10 *Arithmetic expressions:*

$$
\begin{aligned}
E &\to (EBE) \mid (UE) \mid C \mid V \\
B &\to + \mid - \mid \times \mid \div \\
U &\to - \\
C &\to 0 \mid 1 \mid 2 \mid \cdots \\
V &\to X \mid Y \mid Z \mid \cdots
\end{aligned}
$$

Parse the expression:

$$(((X + 1) \times Y) + (2 \times (-X)))$$

Parsing procedure for arithmetic expressions.

Ambiguous and unambigous grammars.

Operator precedence.

Example: $X + 2 \times Y$ and $X + 2 - Y$.

Example 11 *Arithmetic expressions:*

$$
\begin{aligned}
&E \to EB_L F \mid F \\
&F \to FB_H G \mid G \\
&G \to UG \mid H \\
&H \to C \mid V \mid (E) \\
&B_L \to + \mid - \\
&B_H \to \times \mid \div \\
&U \to - \\
&C \to 0 \mid 1 \mid 2 \mid \cdots \\
&V \to X \mid Y \mid Z \mid \cdots
\end{aligned}
$$

Parse the expression:

$$X + 2 \times 4 + -Y$$

Modified parsing procedure.

Exercises: HW 7.1.

Turing Machines and Effective Computability

What is effective computability?

Formalisms:

- Turing machines, by Alan Turing
- Post systems, by Emil Post
- μ-recursive functions, by Kurt Gödel
- λ-calculus, by Alonzo Church
- Combinatory logic, by Haskell B. Curry

Church's thesis.

Universality and self-reference.

Definition 8 (TM) *A deterministic one-tape Turing machine is a structure*

$$M \stackrel{def}{=} (Q, \Sigma, \Gamma, \vdash, \sqcup, \delta, s, t, r)$$

where:

- Q - *finite set of control states*,
  Σ - *a finite input alphabet*,
  $\Gamma \supset \Sigma$ - *a finite tape alphabet*,
  $\vdash \in \Gamma - \Sigma$ - *the left endmarker*,
  $\sqcup \in \Gamma - \Sigma$ - *the blank symbol*,

- $\delta : Q \times \Gamma \rightarrow Q \times \Gamma \times \{L, R\}$ - *the transition function*,

- $s \in Q$ - *the start state*,
  $t \in Q$ - *the accept state*,
  $r \in Q$ - *the reject state*.

Configurations: $Q \times \Gamma^* \times N$.

Start configuration on input $x \in \Sigma^*$: $\langle s, \vdash x, 0 \rangle$.

Let

$$\rightarrow \subseteq (Q \times \Gamma^* \times N) \times (Q \times \Gamma^* \times N)$$

be the least transition relation closed under the rules:

$$\frac{\delta(p, z_n) = (q, b, L)}{\langle p, z, n \rangle \rightarrow \left\langle q, \mathbf{s}_b^n(z), n-1 \right\rangle}$$

$$\frac{\delta(p, z_n) = (q, b, R)}{\langle p, z, n \rangle \rightarrow \left\langle q, \mathbf{s}_b^n(z), n+1 \right\rangle}$$

Machine M accepts input $x \in \Sigma^*$ if:

$$\langle s, \vdash x, 0 \rangle \rightarrow^* \langle t, y, n \rangle$$

and rejects x if:

$$\langle s, \vdash x, 0 \rangle \rightarrow^* \langle r, y, n \rangle$$

Machine M halts on x if it either accepts or rejects x.

M is total if it halts on all inputs.

As usual, the language $L(M)$ of M is the set of all input strings accepted by M.

A set of strings $A \subseteq \Sigma^*$ is called recursively enumerable if $A = L(M)$ for some Turing machine M, and recursive if M is total.

Example 12 *Turing machine for:*

$$L \stackrel{def}{=} \{w \cdot w \mid w \in \{a, b\}^*\}$$

Equivalent models: multiple tapes, two-way infinite tapes, two stacks, counter automata, enumeration machines.

Exercises: HW 8.1, ME 96.

Universal Machines and Undecidability

Encoding Turing machines over $\{0, 1\}$:

$$0^n 1 0^m 1 0^k 1 0^s 1 0^t 1 0^r 1 0^u 1 0^v 1 \cdots$$

for a Turing machine with:
- n states represented by 0 to $n-1$,
- m tape symbols represented by 0 to $m-1$,
- of which the first k are the input symbols,
- s, t, r are the start, accept and reject states,
- u, v are the endmarker and blank symbols,

followed by a sequence of substrings:

$$0^p 1 0^a 1 0^q 1 0^b 1 0^d 1 \cdots$$

for each $\delta(p, a) = (q, b, d)$.

We can construct a universal Turing machine U such that:

$$L(U) = \left\{ \widehat{M} \sharp \widehat{x} \mid M \text{ accepts } x \right\}$$

One can view U as a programmable device, and M as its program! U can also be easily modified to U' for semideciding rejection.

There are countably many Turing machines, but uncountably many decision problems on Turing machines, and, due to Cantor's theorem, there is no bijection between the two sets.

Recall from Theorem 10 that:

$$L \stackrel{\text{def}}{=} \left\{ \widehat{M} \mid M \text{ does not accept } \widehat{M} \right\}$$

is not r.e. But then, neither is

$$L_{NSA} \stackrel{\text{def}}{=} \left\{ \widehat{M} \sharp \widehat{M} \mid M \text{ does not accept } \widehat{M} \right\}$$

r.e. since we can reduce from the previous problem: If there was a machine M_{NSA} accepting L_{NSA}, we could modify it to M_L so that it first scans the input x and replaces it by $x\sharp x$, and then continues as M_{NSA}. But then M_L would accept L which is impossible!

By a similar argument,

$$L_{NA} \stackrel{\text{def}}{=} \left\{ \widehat{M} \sharp \widehat{x} \mid M \text{ does not accept } x \right\}$$

is not r.e., since we can reduce from the previous problem by inserting an initial check whether the input string is of the shape $x \sharp x$. (*cf.* $L(U)$)

Now,

$$L_H \stackrel{\text{def}}{=} \left\{ \widehat{M} \sharp \widehat{x} \mid M \text{ halts on } x \right\}$$

is r.e. (*cf.* $L(U)$), but not recursive since we can reduce from the previous problem: If there was a total M_H accepting L_H, we could modify it to a machine M_{NA} by running L_H first, and either accepting if L_H rejects, or continuing as U' otherwise. M_{NA} will thus accept L_{NA}.