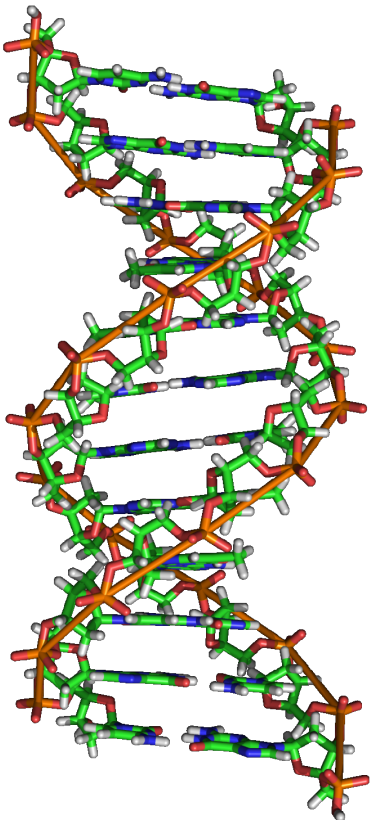
Genome Grammars

Genome Sequences and Formal Languages



Andreas de Vries

Version: June 17, 2011

Wir sind aus Staub und Fantasie

Andreas Bourani, *Nur in meinem Kopf* (2011)

Contents

Foreword

At the beginning of this millennium there has been made great progress in understanding the genetical foundations of life. During its first decade the human genome has been completely deciphered and stored in huge databases, accessible from the public. Additionally, the emphasis of molecular biology has shifted, today the genome is considered more likely as a whole rather than as a collection of genes [23, §2.6].

From the point of view of computer science, a genome sequence is nothing more than a string over an alphabet of four letters. Since there are biochemical laws evolutionary influences, the strings obey certain rules which in turn can be considered to form a grammar. Thus genome sequences are words of a formal language.

The idea to study this subject more extensively emerged in the beginning of 2011. Due to the brilliant, but sometimes hastily written book of Rainer Merkl and Stephan Waack [19] I could gain the current state of the art in bioinformatics. In the summer term 2011, the seminar *Genetischer Code und Genomgrammatik* has been offered at Southwestfalia University of Applied Sciences in Hagen, and much material of the document on hand has been compiled and worked out in this course. For this reason I would like to thank the participants of the seminar, especially Christina Funke, Cem Kiyak, and Stefan Böcker. Without them this document would not exist.

Hagen, June 2011

Andreas de Vries

Chapter 1

Genetics

1.1 Cell physiology

The physical aspects of function and structure of living cells is often called cell physiology. Cells are the fundamental functional units of life. They can occur as isolated individuals or integrated into organisms. Like entire organisms, cells take in chemical or solar energy, a fraction of which is turned into useful mechanical activity or the synthesis of enrgy-storing molecules, via a set of processes collectively called *metabolism*. In particular, each cell manufactures external structures in order to *grow* or *replicate*. Much of this structure consists of a versatile class of macromolecules called *proteins*. Human bodies contain about 100 000 different protein types.

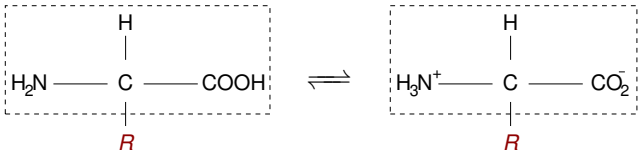
The simplest and most ancient types of cells are the *prokaryotes*, consisting of the family of *bacteria* and of the family of *archæa*. Prokaryotes are about 1 μm long, and their gross anatomy consists mainly of a thick, rigid *cell wall* surrounding a single interior compartment. Just inside the wall lies the *plasma membrane* as a thin layer.

Plants, fungi, and animals are collectively called *eukaryotes*. Baker's yeast, *Saccharomyces cervisiæ*, is an example of a simple eukaryotic cell. Eukaryotic cells are bigger than prokaryotes, typically 10 μm or more in diameter, and are surrounded by a plasma membrane. Cells of fungi and plants have a cell wall, cells of animals do not. Eukaryotes contain various internal compartiments, organelles, each bounded by one or more membranes similar to the plasma membrane In particular, eukaryotic cells by definition have a *nucleus*. It contains the genetic material which condenses to visible *chromosomes* during cell division, in which phase the nucleus dissolves and reform afterwards. The rest of the cell's content is called the *cytoplasm*. It contains several membrane-bounded organelles, in particular *mitochondria*, sausage-shaped organelles about 1 μm wide to carry out the final stages of the metabolism of food and the conversion of its chemical energy into molecules of ATP. The part of the cytoplasm not contained in any organelle is called the *cytosol*.

In this chapter we will consider basic biochemical macromolecules, namely, proteins and nucleic acids. Proteins are polymers made of amino acids, whereas the nucleic acids DNA and RNA are polymers of nucleotides. Both are essential components for the storage and the expression of genetic information [28, §A3].

1.2 Amino acids and proteins

There exist 22 natural *amino acids*, but only 21 are found in eukaryotes. Any amino acid consists of an amine group H_2N and a carboxyl group COOH, as well as a side chain that varies between different amino acids. The side chain is often called the *residue* of the amino acid. Thus, the chemical structure of an amino acid (more accurately, an α amino acid) is given by



(1.1)

where R denotes the residue and the framed part the amide and carboxyl groups and the central C atom, often called C_α [19, §1.5], [24, §5.1].[1] In an aqueous solution an amino acid has the form on the right hand side of Equation (1.1). As a solid in a liquid, the charge of an amino molecule depends on the pH of the liquid. At a certain pH, the *isoelectric point*, an amino acid has no overall charge since the number of protonated ammonia groups (NH_3^+, positive charges) and deprotonated carboxylate groups (COO^-, negative charges) are equal [22, p. 310]. Each amino acid has its own characteristic isoelectric point. The ions produced at the isoelectric point have both positive and negative charges and are known as a *zwitterion*, cf. right hand side of (1.1).

An essential property of the amino acids is their strong tendency to *polymerization*, i.e., to the formation of linear chains, especially under the influence of heat. During this process, two amino acid molecules form a *peptide bond* –NH–CO– when the carboxyl end of the one molecule reacts with the amino end of the other molecule. The peptide bond releases a molecule of water (H_2O). Thus this pro-

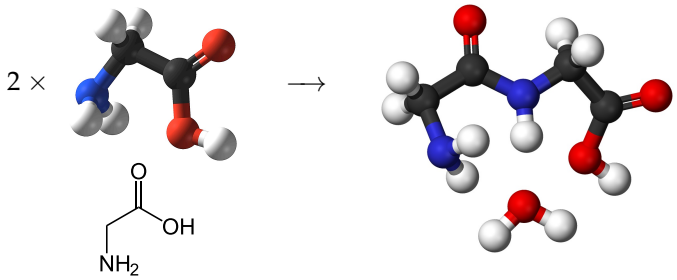


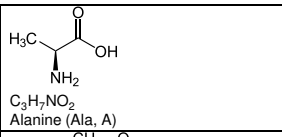
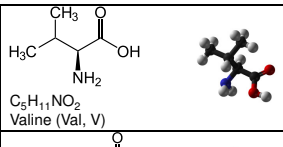
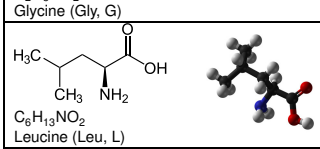
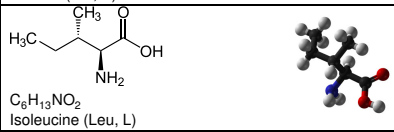
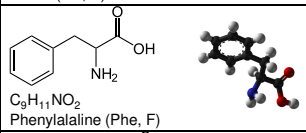
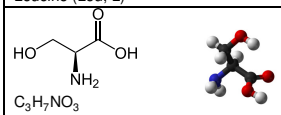
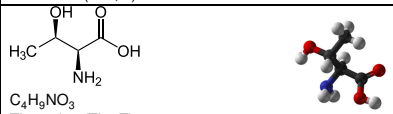
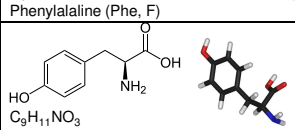
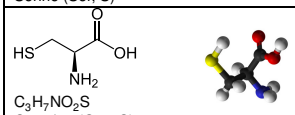
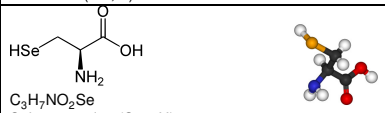
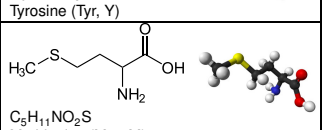
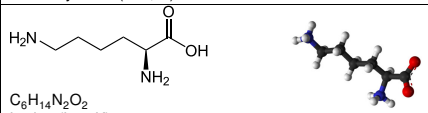
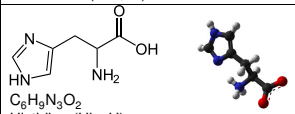
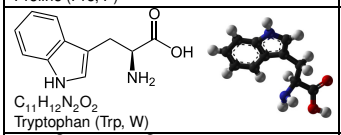
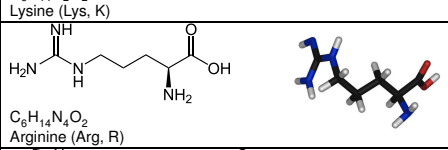
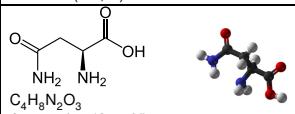
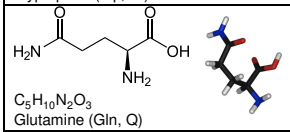
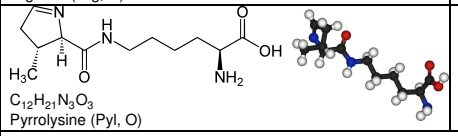
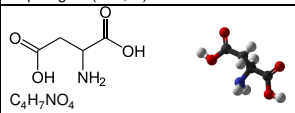
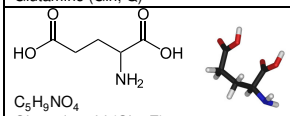
Figure 1.1: Condensation of two glycine molecules. From *en.wikipedia.org*

cess is a dehydration synthesis reaction, i.e., a condensation. In general,[2] after a synthesis of two amino acid molecules their residues remain unchanged, whereas

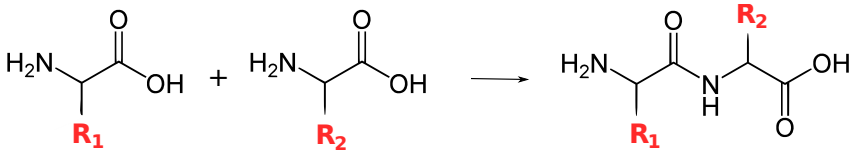
[1] The only exception is proline which lacks a hydrogen in its amide group. Strictly speaking, proline thus is *not* an amino acid but an *imino acid* [24, p 93].

[2] Again the exception from this rule is proline, where the hydrogen atom is replaced by a bond to the side chain. Cf footnote [1] above.

Table 1.1: The 22 amino acids Figures adapted from *wikipedia.org*

 $C_2H_5NO_2$ Glycine (Gly, G)	 $C_3H_7NO_2$ Alanine (Ala, A)	 $C_5H_{11}NO_2$ Valine (Val, V)
 $C_6H_{13}NO_2$ Leucine (Leu, L)	 $C_6H_{13}NO_2$ Isoleucine (Leu, L)	 $C_9H_{11}NO_2$ Phenylalaline (Phe, F)
 $C_3H_7NO_3$ Serine (Ser, S)	 $C_4H_9NO_3$ Threonine (Thr, T)	 $C_9H_{11}NO_3$ Tyrosine (Tyr, Y)
 $C_3H_7NO_2S$ Cysteine (Cys, C)	 $C_3H_7NO_2Se$ Selenocysteine (Sec, U)	 $C_5H_{11}NO_2S$ Methionine (Met, M)
 $C_5H_9NO_2$ Proline (Pro, P)	 $C_6H_{14}N_2O_2$ Lysine (Lys, K)	 $C_6H_9N_3O_2$ Histidine (His, H)
 $C_{11}H_{12}N_2O_2$ Tryptophan (Trp, W)	 $C_6H_{14}N_4O_2$ Arginine (Arg, R)	 $C_4H_8N_2O_3$ Asparagine (Asn, N)
 $C_5H_{10}N_2O_3$ Glutamine (Gln, Q)	 $C_{12}H_{21}N_3O_3$ Pyrrolysine (Pyl, O)	 $C_4H_7NO_4$ Aspartic acid (Asp, D)
 $C_5H_9NO_4$ Glutamic acid (Glu, E)		

the rest forms the peptide bond:[3]



(1.2)

See Figure 1.1, for instance. If n amino acid molecules are combined, the molecule is called *peptide* for small $n \lesssim 100$, and *protein* for great $n \gtrsim 100$. The sequence

$$\mathrm{H_2N-}\underbrace{\boxed{\mathrm{C_\alpha-CO-NH}}-\boxed{\mathrm{C_\alpha-CO-NH}}-\cdots-\boxed{\mathrm{C_\alpha-CO-NH}}}_{n\text{ times}}\mathrm{-COOH} \tag{1.3}$$

is called the *backbone* of the molecule. Its free amine end is called *N-terminus*, and its free carboxyl end *C-terminus*. In the backbone, the $3n$ atoms

$$\underbrace{\boxed{\mathrm{C_\alpha–C–N}}-\cdots-\boxed{\mathrm{C_\alpha–C–N}}}_{n\text{ times}} \tag{1.4}$$

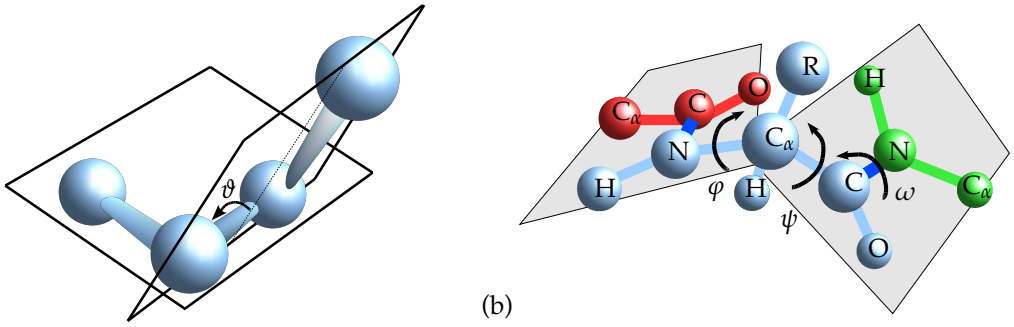
[3] Note the Haworth projection, see footnote 11 on p 12

directly form the peptide bond. In living systems, the the polymerization occurs under the consumption of ATP in the ribosomes.

A peptide bond can also be broken by *amide hydrolysis*. The peptide bonds in proteins are metastable, meaning that in the presence of water they will break spontaneously, releasing 2–4 kcal/mol of free energy. This process is extremely slow, in living organisms it is facilitated by enzymes.

1.2.1 Geometry of peptide bonds

For four atoms of a molecule bonded together in a linear chain, the *dihedral angle* or *torsional angle* is the angle between the plane formed by the first three atoms and the plane formed by the last three atoms (Figure 1.2). The enthalpy ΔH of



(a) (b)

Figure 1.2: (a) Dihedral angle ϑ of a linear chain of four bonde atoms. (b) Conformation of a peptide bond.

the molecule due to the rotation, i.e., the torsional energy, is approximated by the following formula,

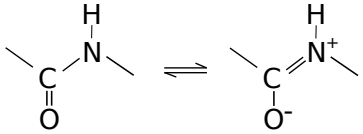
$$\Delta H = a_1(1 + \cos\vartheta) + a_2(1 - \cos 2\vartheta) + a_3(1 + \cos 3\vartheta) \tag{1.5}$$

where a_1, a_2, a_3 are scaling factors specific to the molecule.

In proteins, the six linearly chained atoms C_α – C – N – C_α – C – N of the backbone (1.4) of two bonded amino acid molecules determine the dihedral angles ω, φ and ψ as follows (Figure 1.2 b).

$$\omega = \measuredangle(C_\alpha\text{–C–N–}C_\alpha), \qquad \varphi = \measuredangle(\text{C–N–}C_\alpha\text{–C}), \qquad \psi = \measuredangle(\text{N–}C_\alpha\text{–C–N}). \tag{1.6}$$

The chain C_α – C – N – C_α is also called the *peptide group*. Its dihedral angle can only take the two values $\omega = 0°$ or $180°$. That means, that the four atoms of a peptide group always lie rigidly in a plane. The reason for this crucial property is as follows. Since the oxygen is strongly electronegative, one of the electrons of the double bond of the carbonyl group (C=O) is preferably at the oxygen atom. The electron orbitals of the peptide bond are a quantum-mechanical mixture of two structures,



(1.7)

a so-called *resonance structure*. Because of this special structure, the length of the C–N bond is 133 pm instead of the normal 145 pm, and it is not rotatable [24, §6.2.5].[4] The value $\omega = 0°$ refers to the *cis* or *(Z)* conformation, the value $\omega = 180°$ refers to the *trans* or *(E)* conformation. In proteins, the trans conformation occurs much more frequently, roughly to a ratio 1000:1, unless for peptide groups involving proline, which only has a 3:1 ration.[5]

Fixing the plane spanned by N – C_α – C as the molecules central plane, φ measures its dihedral angle to the "left" peptide bond plane and ψ to the "right" peptide bond plane. In other words, φ measures the torsion wandering along the backbone from the left peptide bond C – N, the backbone rotates around the axis N – C_α by the angle φ, and then by the angle ψ around the axis C_α – C to the next peptide bond. Hence for each pair of amino acid molecules in the protein, the dihedral angle pair (φ, ψ) completely determines the geometry of the backbone. The concrete values of the dihedral angles (φ, ψ) of a pair of amino acid molecules is determined by the residue R attached to the C_α atom.

In a peptide chain only those combinations of angles φ and ψ are allowed, in which the respective atoms do not collide, i.e., in which they stay outside their van der Waals radii. This was first done by Ramachandran and coworkers in 1963.[6] Moreover, those combinations are preferred which minimize the free Gibbs energy of the peptide bond [24, §6.2.5]. For each amino acid, the specific dihedral angles (φ, ψ) have been measured experimentally in many proteins and can be depicted[7] in the *Ramachandran plot*, see Figure 1.3. For all amino acids unless glycine

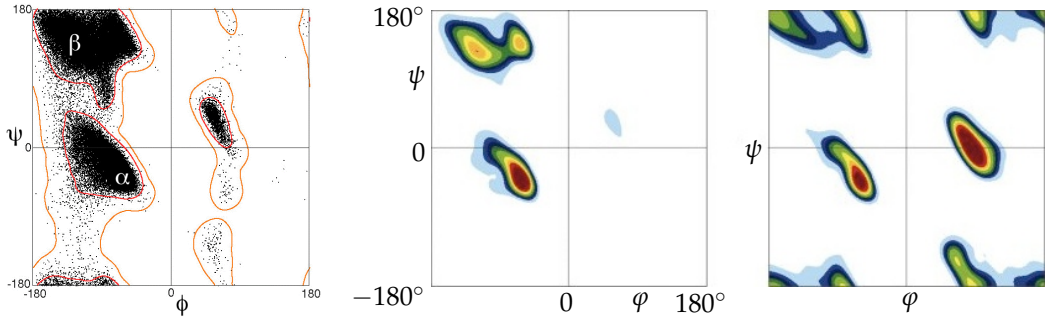


Figure 1.3: Experimental Ramachandran plot for all non-glycines (left) according to [18], as well as nonglycines (middle) and glycine (right) according to [14]. Adapted from *en.wikimedia.org*

the allowed regions in the (φ, ψ) region are rather small. The side group of glycine, however, consists solely of a hydrogen atom. For all other amino acids there are three main regions, corresponding to three particular secondary protein structures: right-handed α helices around the point $(-57°, -47°)$, parallel β sheets

[4] According to the Hartree–Fock method for molecules in molecular orbit theory, molecular orbitals are completely defined as eigenfunctions (wave functions) of the self-consistent field Hamiltonian. In calculations on molecules, the molecular orbitals are expanded in terms of an atomic orbital basis set determined by the Roothaan equations, which are a representation of the Hartree-Fock equation. [*http://www.ch.ic.ac.uk/vchemlib/course/mo_theory/main.html*] Complementarily, valence bond theory considers a covalent bond as being formed between two atoms by the overlap of half filled valence atomic orbitals of each atom containing one unpaired electron [20, §8.1].

[5] *http://en.wikipedia.org/wiki/Peptide_bond#Cis.2Ftrans_isomers_of_the_peptide_group*

[6] G N Ramachandran, C Ramakrishnan & V Sasisekharan (1963): 'Stereochemistry of polypeptide chain configurations', *J. Mol. Biol.* **7**: 1, 95–99. *doi:10.1016/S0022-2836(63)80023-6*

[7] *http://www.proteopedia.org/wiki/index.php/Ramachandran_Plot*

around $(-119°, 113°)$, antiparallel β sheets around $(-139°, 135°)$, [24, p 117], and left-handed α helices around $(57°, 47°)$, [19, p 15]. Glycine, however, has a frequent combination[8] around $(55, -116)$. Moreover, the distinctive cyclic structure of proline's side chain locks its φ dihedral angle at approximately[9] $\varphi \approx -75°$, i.e., proline has an exceptional conformational rigidity compared to other amino acids. Hence, proline loses less conformational entropy upon folding, which may account for its higher prevalence in the proteins of thermophilic organisms. Hence the insertion of glycine or prolin in a protein usually includes a sharp twist in the backbone conformation. In other words, glycine and proline act as structural disruptors in the middle of regular secondary structure elements such as α helices and β sheets.

1.2.2 Protein structure

In biochemistry and structural biology, the general three-dimensional form of local segments of biopolymers such as proteins and nucleic acids is called *secondary structure*. In proteins, it is defined by patterns of hydrogen bonds between back-

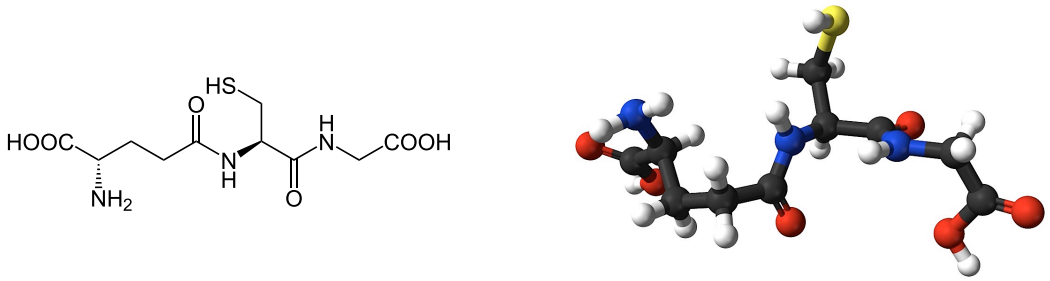


Figure 1.4: The tripeptid glutathione $C_{10}H_{17}N_3O_6S$. (C: gray, H: white, N: blue, O: red, S: yellow.) From *http://wikipedia.org/*

bone amide and carboxyl groups; sidechain-mainchain and sidechain-sidechain hydrogen bonds are irrelevant. The several secondary structures of a protein form the *tertiary structure*.

Alpha helix. The most prevalent secondary structure in proteins is the right-handed α helix, or *Pauling-Corey-Branson α helix*, in which every backbone N–H group donates a hydrogen bond to the backbone C=O group of the amino acid four residues earlier, i.e., $i + 4 \to i$ hydrogen bonding. Cf Figure 1.5 (a). In a right-

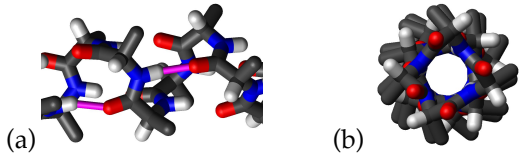


Figure 1.5: α helix (a) from sideview and (b) from topview From *http://wikipedia.org/*

handed α helix each amino acid residue corresponds to a 100° turn in the helix, i.e.,

[8] *http://www.proteopedia.org/wiki/index.php/Ramachandran_Plot#Glycine*
[9] *http://en.wikipedia.org/wiki/Proline#Structural_properties*

the helix has 3.6 residues per turn, and a translation of 0.15 nm along the helical axis. Thus the pitch of the α helix as the vertical distance between two consecutive helical turns is $3.6 \cdot 0.15$ nm $= 0.54$ nm [24, p 177]. Residues in α helices typically adopt backbone dihedral angles around $(\varphi, \psi) = (-57°, -47°)$, as shown in the Ramachandran plot Figure 1.3. In more general terms, they adopt dihedral angles such that the average of φ and ψ is approximately $-104°$,

$$\frac{\varphi + \psi}{2} \approx -104° = -\tfrac{26}{45}\,\pi. \tag{1.8}$$

For general values of (φ, ψ) the rotation angle θ per residue of any polypeptide helix with trans isomers is given by the equation[10]

$$3\cos\theta = 1 - 4\,\cos^2\frac{\varphi + \psi}{2}. \tag{1.9}$$

For a typical right-handed α helix with $\frac{\varphi+\psi}{2} \approx 52°$ we thus have $\theta \approx 100°$.

Beta sheet. A β sheet consists of several β strands connected laterally by at least two or three backbone hydrogen bonds, forming a twisted and pleated sheet. A β strand is a polypeptide chain typically 3 to 10 amino acids long. The pleating causes the average distance between the i-th and the $(i+2)$-th C_α atom to be approximately 0.6 nm, rather than the $2 \cdot 0.38 = 0.76$ nm expected from two fully extended trans peptide virtual bonds. The distance between two C_α atoms in adjacent β strands is roughly 0.5 nm. Usually, β strands are arranged adjacent to

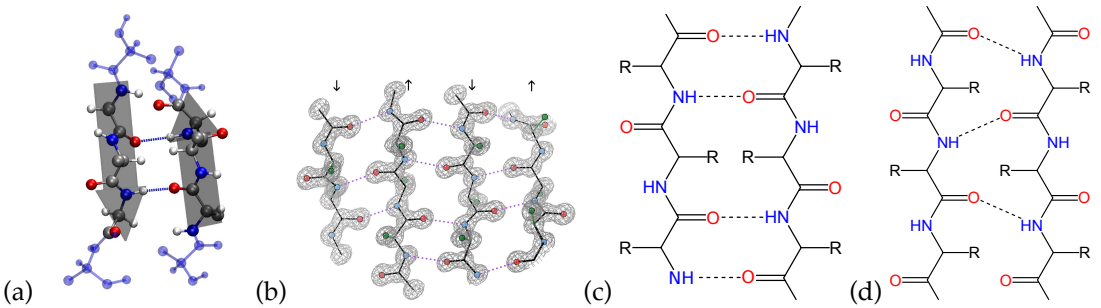


Figure 1.6: (a) Antiparallel β sheet with 2 strands. (b) Antiparallel β sheet with 4 strands. (c) Structure diagram of antiparallel β sheet. (d) Structure diagram of parallel β sheet. From *http://wikipedia.org/*

other strands and form an extensive hydrogen bond network with their neighbors in which the N–H groups in the backbone of one strand establish hydrogen bonds with the C=O groups in the backbone of the adjacent strands. Adjacent strands are aligned so that their C_α atoms are adjacent and their side chains point in the same direction. The pleats of β strands arise from the tetrahedral chemical bonding at the C_α atom: if, for instance, a side chain points straight up then the bond to the C atom must point slightly downwards, since its bond angle is approximately 109.5°.

[10] R.E. Dickerson & I. Geis: *Structure and Action of Proteins*, Harper, New York 1969

β strands exhibit a twist due to the chirality of their component amino acids. The energetically preferred peptide backbone dihedral angles are given as $(\varphi, \psi) = (-135°, 135°)$. In antiparallel sheets, we have $(\varphi, \psi) \approx (-140°, 135°)$, and in parallel sheets $(\varphi, \psi) \approx (-120°, 115°)$. This is in accordance to the Ramachandran plot Figure 1.3.

β sheets are implicated in the formation of protein aggregates and fibrils which are observed in some human diseases, notably the amyloidoses such as Alzheimer's disease.

1.3 Nucleic acids

The *nucleic acids* DNA and RNA are polymeres of nucleotides, which themselves consist of a nitrogenous base, a pentose sugar, and phosphoric acid.

Example 1.1. *(DNA base pairing) Deoxyribonucleic acid (DNA)* is a nucleic acid and serves as the main repository of genetic information of life, containing the instructions for creating and maintaining a living organism [22, §2.3.4]. DNA consists of multiple *nucleotides*, i.e., molecules consisting of three components: phosphate P, the sugar 2-deoxyribose, and one of the four *nucleobases* (or *nucleotide bases*) adenine (A), guanine (G), cytosine (C), and thymine (T), see Figure 1.7. They are

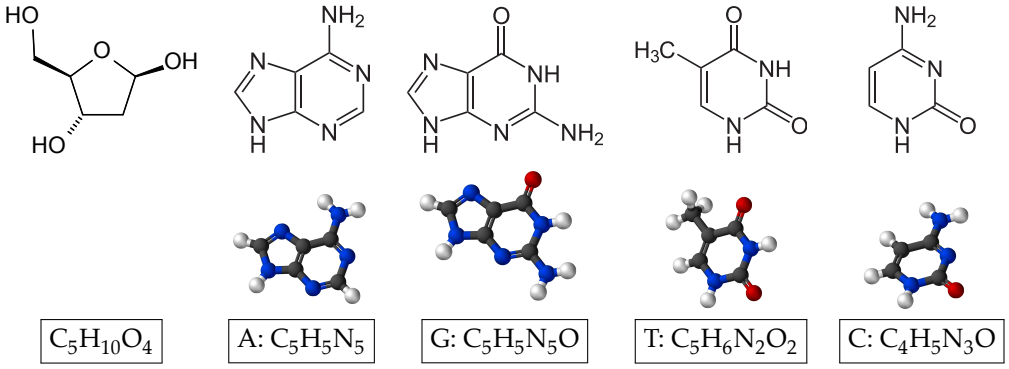


Figure 1.7: Structural formulas of deoxyribose and the nucleobases. [From de.wikipedia.org]

depicted with their structural formula in Haworth projection.[11] Adenine and guanine are *purines* containing two atomic rings, thymine and cytosine are *pyrimidines* having a single ring [22, §2.2.1].

In living organisms, DNA does not usually exist as a single molecule but instead as a pair of two molecule strands that are held tightly together. These two long strands entwine like vines, in the shape of a double helix. Each type of base on one strand forms a bond with just one type of base on the other strand. This is called *complementary base pairing*. Here, purines form hydrogen bonds to pyrimidines, with A bonding only to T, and C bonding only to G. In Figure 1.8, these pairs are depicted, with non-covalent hydrogen bonds between the pairs shown as dashed lines. This arrangement of two nucleotides binding together across the

[11] In Haworth projection, each vertex $\neq$ O or N represents an implicit C atom, each vertex in a pentagon not adjacent to a hexagon an implicit H atom, and each *unmarked* vertex an (extra) H atom [15, H02749], [24, p 106].

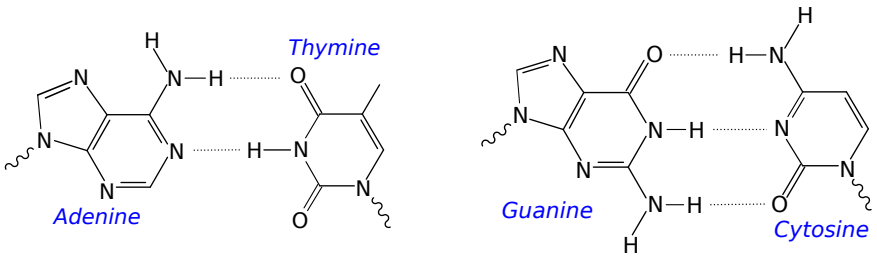


Figure 1.8: The DNA base pairs AT and GC. [From en.wikipedia.org]

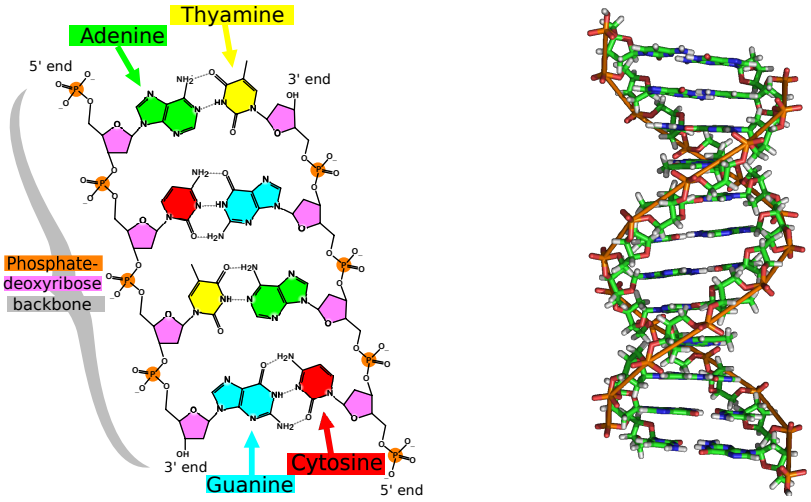


Figure 1.9: The chemical structure of the DNA and its double helix structure. [From *en.wikipedia.org*]

double helix is called a *base pair*. AT forms two hydrogen bonds, and GC forms three hydrogen bonds.

$$\underbrace{C_5H_5N_5}_{\text{adenine}} + \underbrace{C_5H_6N_2O_2}_{\text{thymine}} \longrightarrow \underbrace{C_{10}H_{11}N_7O_2}_{\text{AT base pair}}, \tag{1.10}$$

$$\underbrace{C_5H_5N_5O}_{\text{guanine}} + \underbrace{C_4H_5N_3O}_{\text{cytosine}} \longrightarrow \underbrace{C_9H_{10}N_8O_2}_{\text{GC base pair}}. \tag{1.11}$$

As hydrogen bonds are not covalent, they can be broken and rejoined relatively easily. The two strands of DNA in a double helix can therefore be pulled apart like a zipper, either by a mechanical force or by high temperature. As a result of the base pairing complementarity, all the information in the double-stranded sequence of a DNA helix is duplicated on each strand (Figure 1.9). DNA strands have a directionality, and the different ends of a single strand are called the "3′ (three-prime) end" and the "5′ (five-prime) end." These terms refer to the carbon atom in deoxyribose to which the next phosphate in the chain attaches. Since they are orientated in opposite directions, the two strands of DNA are antiparallel. ◊

Example 1.2. *Ribonucleic acid (RNA)* is a nucleic acid like DNA, but with the nucleobase thymine (T) replaced by uracil (U), and ribose instead of deoxyribose. RNA usually serves as a transmitter and processor of genetic information. see Figure 1.10. Like thymine, uracil is a pyrimidine having a single ring [22, §2.2.1].

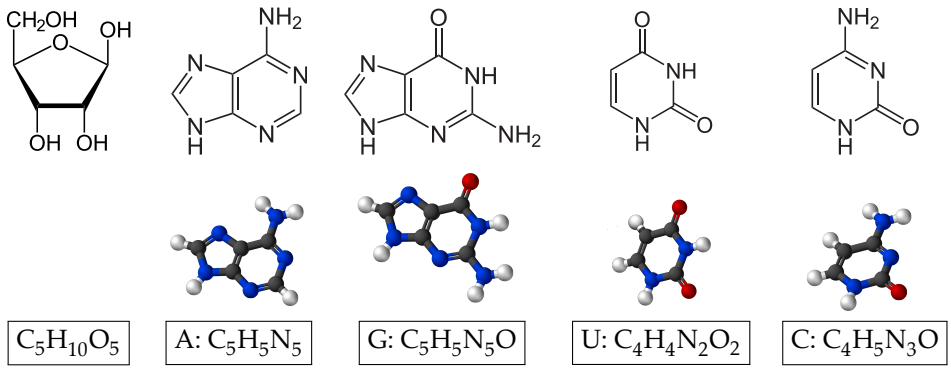
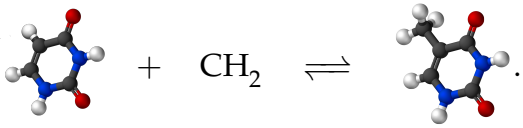


Figure 1.10: Structural formulas of ribose and the nucleobases. [From de.wikipedia.org]

Uracil has one less C atom and two less H atoms than thymine, i.e., uracil + methylene $\rightleftharpoons$ thymine,



(1.12)

Due to the lack of methylene, uracil is energetically less expensive to be produced than thymine. Moreover, ribose differs from deoxyribose only by an accessory oxygen atom (as the Greek prefix "deoxy-" suggests), which makes RNA less stable than DNA due to hydrolysis. Unlike DNA, most RNA molecules occur in single strands and may adopt very complex three-dimensional structures like hairpin loops, bulges, or internal loops, and thus RNA can undergo much more chemical reactions than DNA.

There are mainly three types of cellular RNA, messenger RNA (mRNA), ribosomal RNA (rRNA), and transfer RNA (tRNA).

In 1967, Carl Woese hypothesized an "RNA world" in which the earliest forms of life, i.e., self-replicating molecules, relied solely on RNA both to carry genetic information *and* to catalyze biochemical reactions. However, the chemical properties of RNA make large RNA molecules inherently fragile, especially due to hydrolysis. ◊

1.4 DNA replication

The replication of DNA is a crucial process for cell growth and, more fundamentally, for biological inheritance and the preservation of the genetic information beyond the individual death of a living organism. Three essential enzymes for DNA replication are *topoimerase*, *helicase*, and *DNA polymerase*, all three made of proteins. The first two enzymes split the DNA into a replication fork, the polymerase synthesizes a new strand from a given one cf. Figure 1.11.

Example 1.3. *(Replication fork)* The replication fork is created by enzymes called *helicases*, which break the hydrogen bonds holding the two DNA strands together. The resulting structure has two branches, each one made up of a single strand of

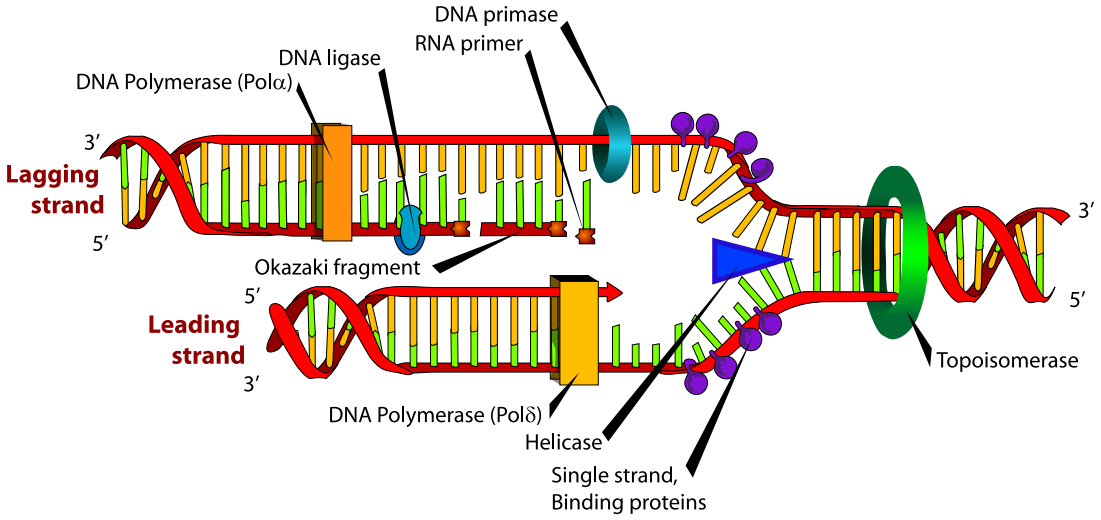


Figure 1.11: DNA replication mechanism. [From *en.wikipedia.org*]

DNA. These two strands serve as the template for the leading and lagging strands which will be created as DNA polymerase matches complementary nucleotides to the templates; the templates may be properly referred to as the leading strand template and the lagging strand template.

As helicase unwinds DNA at the replication fork, the DNA ahead is forced to rotate. This process results in a build-up of twists in the DNA ahead. Te enzymes *DNA topoisomerases* solve these physical problems in the coiling of DNA. Topoisomerase I cuts a single backbone on the DNA, enabling the strands to swivel around each other to remove the build-up of twists. Topoisomerase II cuts both backbones, enabling one double-stranded DNA to pass through another, thereby removing knots and entanglements that can form within and between DNA molecules. ◊

Example 1.4. *(DNA polymerase)* DNA polymerases are a family of enzymes which catalyze the synthesis of deoxyribonucleotides in DNA into a DNA strand. In DNA replication, a DNA polymerase "reads" an intact DNA strand as a template and uses it to synthesize the new strand. This process copies a piece of DNA. The newly-polymerized molecule is complementary to the template strand and identical to the template's original partner strand.

A DNA polymerase can only extend an existing DNA strand paired with a template strand; it cannot begin the synthesis of a new strand. To begin synthesis of a new strand, a short fragment of DNA or RNA, called a *primer*, must be created and paired with the template strand before DNA polymerase can synthesize new DNA.

Once a primer pairs with DNA to be replicated, DNA polymerase synthesizes a new strand of DNA by extending the 3′ end of an existing nucleotide chain, adding new nucleotides matched to the template strand one at a time via the creation of phosphodiester bonds. The energy for this process of DNA polymerization comes from two of the three total phosphates attached to each unincorporated base. When a nucleotide is being added to a growing DNA strand, two of the phosphates are removed and the energy produced creates a phosphodi-

ester (chemical) bond that attaches the remaining phosphate to the growing chain. The energetics of this process also help explain the directionality of synthesis — if DNA were synthesized in the 3′ to 5′ direction, the energy for the process would come from the 5′ end of the growing strand rather than from free nucleotides.

Some DNA polymerases also have proofreading ability; they can remove nucleotides from the end of a strand in order to correct mismatched bases. If the 5′ nucleotide needs to be removed during proofreading, the triphosphate end is lost. Hence, the energy source that usually provides energy to add a new nucleotide is also lost.

DNA polymerases use magnesium ions as cofactors.

There are known five prokaryotic DNA polymerases such as bacteria, and eukaryotes have at least 15 DNA polymerases. For E. coli, for instance, the replication is mainly due to the "DNA polymerase III", for eukaryotes it seems to be done mainly by Pol δ and Pol ε. $\Diamond$

1.5 Flow of information for cell growth

A *gene* is a basic unit of genetic information which, after the process of gene expression, affects the formation of a single protein. Exceptions are certain genes for RNA molecules, such as rRNA and tRNA. Genes are located on *chromosomes* [23, §2.4]. Nearly every cell of our body contains the same set of chromosomes, although each cell has its own specific task: pancrea cells secrete insulin, or hair cells grow hairs. Each cell type has a characteristic arrangements of genes which are active ("switched on") and inactive ("switched off").

Moreover, individual cells can modulate their gene activities depending on external circumstances: If we deny a bacterium its favorite food molecule but supply an alternative food, the cell will suddenly start synthesizing the chemicals needed to metabolize what is available. The secret to gene switching is a class of regulatory proteins which recognize and bind specifically to the beginning of the genes they control. One subclass, the repressors, can block the start of their gene, thereby preventing transcription. Other regulatory proteins help with the assembly of the transcriptional apparatus and have the opposite effect [22, §2.3.3].

In this way the *genome*, i.e., the cell's entire genetic information [19, p 23], specifies an algorithm for creating and maintaining the entire organism containing the cell. Gene regulatory proteins supply some of the switches turning specific instructions of the algorithm on and off. The DNA in the cell nucleus contains the master copy of the "software," a coding sequence of the letters A, T, G, C representing the four nucleobases at one strand and the redundant duplicate under the duality A $\leftrightarrow$ T, G $\leftrightarrow$ C, see Example 1.1. Specific sections of the sequence form the genes, which consist of regulatory regions and coding regions specifying the amino acid sequences of various needed proteins. A complex organism may have tens of thousands of distinct genes, whereas *E. coli* has fewer than 5000, and the simplest known organism, *Mycoplasma genitalium*, has fewer than 500. In addition to the genes, the DNA contains regulatory sequences for the binding of regulatory proteins, along with long stretches with no known function [22, pp 60]. The size of the complete genome of an organism differs considerably for each species [23,

§2.6.1]:

Organism	**Genome size (Mb)**
Escheria coli (bacterium)	4.6
Saccharomyces cerevisiae (yeast)	12.1
Drosophila melanogaster (fruit fly)	150
Homo sapiens	3 000
Mus musculus (mouse)	3 300
Nicotiana tabacum (Tobacco)	4 500
Triticum aestivum (wheat)	17 000

(1.13)

(Here 1 Mb means 1 megabase $= 1 \cdot 10^6$ base pairs.)

The following rough sketch describes the overall flow of information in living cells [22, §2.3.4].

1. *RNA polymerase* is a molecular machine being a combination of a walking motor and an enzyme. in a process called *transcription*, RNA polymerase attaches to the DNA near the start of a gene, pulls a polymer chain through a slot and reads the DNA master copy, simultaneously adding successive monomers to a growing transcript made of RNA, called *messenger RNA (mRNA)*. In eukaryotic cells mRNA leaves the nucleus through pores in the membrane and enters the cytosol. The energy needed to drive RNA polymerase comes from the added nucleotides themselves, which arrive in the high-energy ATP or GTP forms: the polymerase clips off two of the three phosphate groups from each ATP / GTP as it incorporates the nucleotide into the growing transcript.

2. In the cytosol, the information in the messenger RNA is translated into a sequence of amino acids making a new protein by the combined action of more than 50 molecular machines. In particular, a complex of devices called the *ribosome* binds the transcript and again walks along it, successively building up a polypeptide on the basis of instructions encoded in the transcript. The ribosome accomplishes this *translation* by orchestrating the sequential

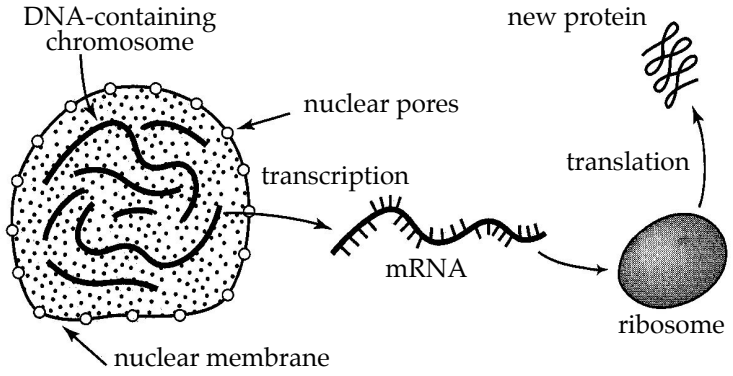


Figure 1.12: Flow of information in a cell. From [22, p. 60].

attachment of *transfer RNA* (tRNA) molecules, each binding to a particular

triplet (!) of monomers in the transcript, the bases, and each carrying the corresponding amino acid monomer, the residue, to be added to the growing polypeptide chain.

3. The linear polypeptide chain may spontaneously fold into a three-dimensional structured functioning protein [22, §2.2.3], or with the help of other auxiliary devices into *chaperones*. Additional chemical bonds can form to cross-link monomers distant from each other along the chain, or even another chain.

4. The folded protein may then form part of the cell's architecture, becoming a functioning device, or it may be a regulatory protein, helping close a feedback loop, creating a mechanism for orchestrating the development of the cell or a multicellular organism.

1.5.1 The genetic code

The sequence of the DNA letters T, C, A, G (the nucleobases or nucleotide bases, cf Example 1.1) forms several genes each of which codes a protein as a sequence of amino acids. The reading direction of the codon is from its 5′ end to its 3′ end [28, §P1]. Three subsequent letters code for a single amino acid. Such a triplett of nucleotides is called a *codon*. The mapping of the $4^3 = 64$ possible codons to the 22 amino acids is called the *genetic code* (Table 1.2). The genetic code is nearly

Table 1.2: The genetic code: DNA codon table (with 1-letter abbreviations for amino acids) and the amino acid abbreviations; TGA and TAG are usually stop cordons, but code in certain cases to Sec or Pyl [20, p 627].

Codon (5′ → 3′) ↦ amino acid							
TTT	F	TCT	S	TAT	Y	TGT	C
TTC	F	TCC	S	TAC	Y	TGC	C
TTA	L	TCA	S	TAA	*	TGA	**U**
TTG	L	TCG	S	TAG	**O**	TGG	W
CTT	L	CCT	P	CAT	H	CGT	R
CTC	L	CCC	P	CAC	H	CGC	R
CTA	L	CCA	P	CAA	Q	CGA	R
CTG	L	CCG	P	CAG	Q	CGG	R
ATT	I	ACT	T	AAT	N	AGT	S
ATC	I	ACC	T	AAC	N	AGC	S
ATA	I	ACA	T	AAA	K	AGA	R
ATG	**M**	ACG	T	AAG	K	AGG	R
GTT	V	GCT	A	GAT	D	GGT	G
GTC	V	GCC	A	GAC	D	GGC	G
GTA	V	GCA	A	GAA	E	GGA	G
GTG	V	GCG	A	GAG	E	GGG	G

Amino acid abbreviations			
A: Ala	alanine	N: Asn	asparagine
C: Cys	cysteine	**O**: ***Pyl***	***pyrrolysine***
D: Asp	aspartic acid	P: Pro	proline
E: Glu	glutamic acid	Q: Gln	glutamin
F: Phe	phenylalaline	R: Arg	arginine
G: Gly	glycine	S: Ser	serine
H: His	histidine	T: Thr	threonine
I: Ile	isoleucine	**U**: ***Sec***	***selenocysteine***
K: Lys	lysine	V: Val	valine
L: Leu	leucine	W: Trp	tryptophane
M: ***Met***	***methionine (start)***	Y: Tyr	tyrosine
		*: **(Stop)**	—

universal, all living organisms on earth use it to a great part, exceptions are mitochondria (phylogenetically old cells, the "cellular power plants" of eukaryotes producing ATP), mycoplasma and some protozoa [19, p 5], [28, §P1, Table 2].

The standard code enables the cells to produce proteins from the 20 α amino acids unless selenocysteine and pyrrolysine, but bacteria, archaea and eukaryotes can build in selenocysteine during the translation in a mechanism called *recoding*. In the usual cases, they both are read as stop codons.

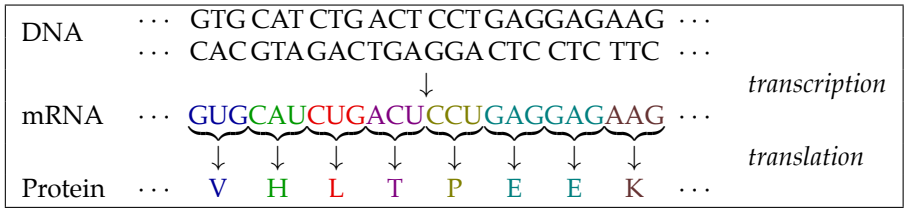


Figure 1.13: Diagram of the central dogma, DNA to RNA to protein, illustrating the genetic code. Adapted from *http://en.wikipedia.org/wiki/File:Genetic_code.svg*

The "central dogma" of molecular biology is the suggestion "DNA makes RNA makes protein," proposed by Francis Crick in the 1950's [28, §D5]. In fact, in both prokaryotic and eukaryotic cells, DNA is transcribed to an RNA molecule (mRNA) which contains the same information, then that messenger RNA is translated into a protein sequence according to the genetic code, cf. Figure 1.12. The DNA replication also is included in the central dogma, where two daughter DNA molecules are formed by duplication of the information in the DNA mother molecule.

However, a few important exceptions to the central dogma have been identified. A number of classes of virus contain a genome consisting of RNA. In *retroviruses*, such as HIV, the single-stranded RNA molecule is converted to a double-stranded DNA copy (with T's replacing U's) which is then inserted into the genome of the host cell. This process is called *reverse transcription*, and the re-

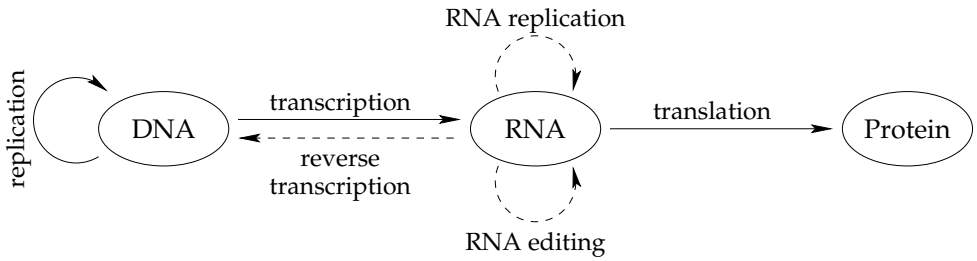


Figure 1.14: The flow of genetic information; the parts contradicting the central dogma are dashed.

sponsible enzyme, encoded in the viral genome, is called *reverse transcriptase*. For some viruses, their RNA genome is copied directly into RNA *(RNA replication)*, for instance coronaviruses or hepatitis C virus. In some organisms, seemingly more common in nonvertebrates, the sequence of the messenger RNA is altered after it is transcribed from the DNA, in a process known as *RNA editing* [28, §O4]. To date, there are not known any examples of a protein being able to specify a nucleic acid sequence, so the *translational step* of the central dogma seems to be unidirectional.

1.5.2 Open reading frames, coding regions, and genes

An *open reading frame (ORF)* is a continous group of adjacent codons following a start codon (TGA) and ending at a stop codon. ORFs are suspected coding regions usually identified by computer in sequenced DNA. When a particular ORF

is known to encode a certain protein, it is usually referred to a a *coding region* [28, §P1]. In this case the ORF can represent a gene, or a part of a gene. In principle, to a given DNA sequence there exist six ORFs, depending on the start nucleobase and the reading direction, i.e., the relevant DNA strand [19, §1.2]. Usually, only further analysis of mRNA transcripts and DNA sequences identifies the correct ORF, see [23, §12.1.2] for the special case of the CFTR gene.

The hunt for the cystic fibrosis gene

Cystic fibrosis (CF), also known as *mucoviscidosis*, is a serious genetically based desease presenting with various symptoms, the most serious of which is the clogging of the respiratory passageways with thick, sticky muscus. This is too thick to be moved by the cilia that line the air passages, and the patient is likely to suffer persistent and repeated infections. Lung function is therefore compromised in CF patients, and even with improved treatments the life expectancy is only around 30 years. The name cystic fibrosis refers to the characteristic scarring (fibrosis) and cyst formation within the pancreas, first recognized in the 1930s. Characterization of the desease at the molecular level was not obtained until the 1980's. CF is the most common genetically based desease found in Western Caucasians, appearing with a frequency of around 1 in 2000–2500 live births. The genetic defect responsible for CF affects a membrane proteininvolved in chloride ion transport [23, §12.1.3].

The CF gene was found by the technique of *positional cloning* which is appropriate to find a gene for which the protein product is unknown. Positional cloning involves identifying a gene by its position and thereby deciphering the molecular connection between phenotype and genotype. In 1985, a linkage marker (met) was found, localizing the CF gene in the long arm of chromosome 7, at band position 7q22 (a black square in Figure 1.15), covering a region of about 500 kbp. The search

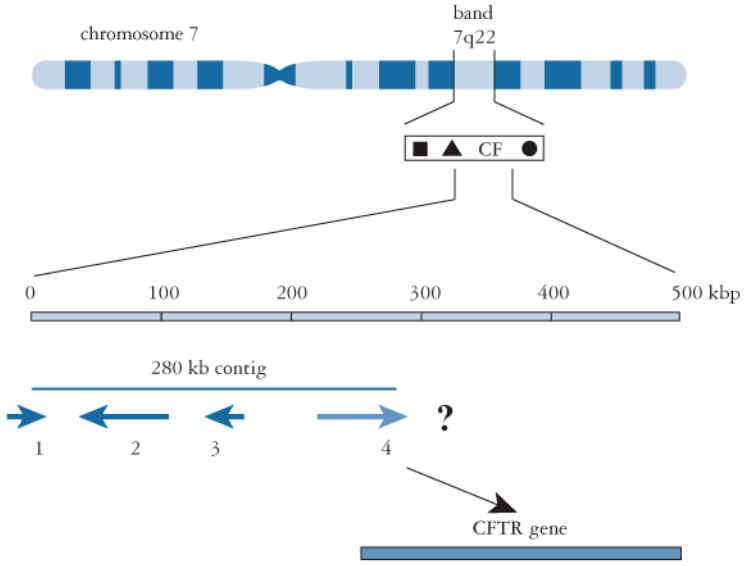


Figure 1.15: The hunt for the cystic fibrosis gene. Figure from [23]

for other markers uncovered two (triangle and circle in Figure 1.15) that showed

no recombination with the CF locus and which could be used to identify a contiguous sequence (clone "contig") of some 280 kbp of DNA. By further techniques ("chromosome walking" and "chromosome jumping"), four open reading frames (ORFs, labelled 1–4 in Figure 1.15). Further analysis of mRNA transcripts and DNA sequences, patterns of gene expression in CF patients being traced, eventually ORF 4 as the start of the "CF gene" coud be identified. It is now named CFTR or *cystic fibrosis transmembrane conductance regulator*.

Once having identified the CFTR gene, more detailed characterization of its normal gene product could be obtained. The gene is some 250 kbp in size and encodes 27 exons which produce a protein of 1 480 amino acids. The protein is similar to the ATP-binding cassette family of membrane transporter proteins. It was noted that around 70% of CF cases had a similar defective region in the sequence, namely a three-base-pair in exon 10. This causes the amino acid phenylalanine to be deleted from the protein sequence. This mutation is called the ΔF508 mutation. Here Δ stands for deletion, F is the single-letter code for phenylalanine, 508 is the position in the primary sequence of the protein. ΔF508 affects the folding of the CFTR protein, meaning that it cannot be processed and inserted correctly into the membrane after translation. Thus patients who carry two ΔF508 alleles

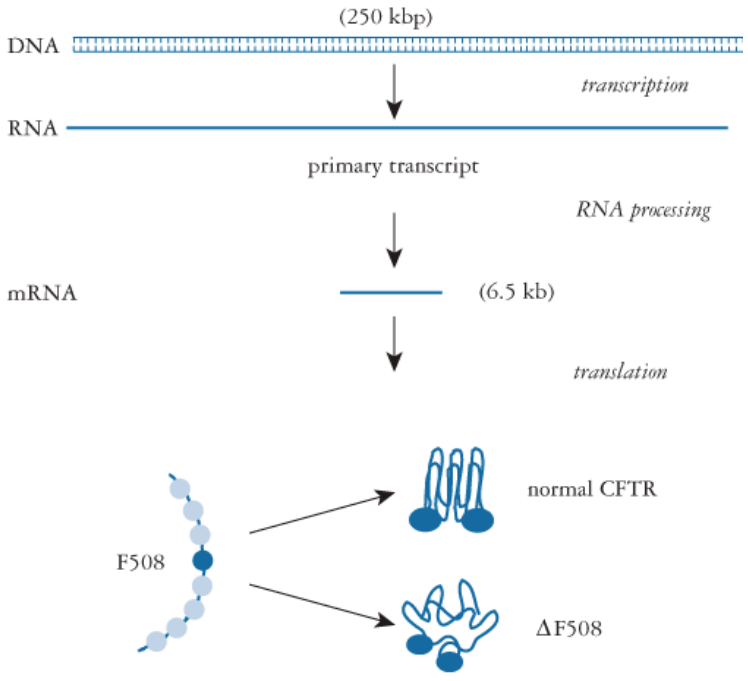


Figure 1.16: The ΔF508 mutation of the CFTR gene. Figure from [23]

do not produce any functional CFTR, causing the protein to fold incorrectly and preventing it reaching the site of incorporation into the membrane, with the associated disease phenotype arising as consequence. The normal and mutant gene sequences around position 508 (Phe) are as follows.

		Ile Ile Phe Gly
Normal:	5′ – GAA AAT ATC ATC TTT GGT GTT TCC – 3′	
Mutant:	5′ – GAA AAT ATC ATT GGT GTT TCC – 3′	
		Ile Ile Gly

The deleted three base pairs are framed in the normal sequence, causing the loss of phenylalanine.

Although the ΔF508 mutation is the most common cause of cystic fibrosis, to date around 1 500 mutations have been identified in the CFTR gene.

Chapter 2

Formal languages

The theory of formal languages deals with the systematic analysis, classification, and construction of sets of words generated by finite alphabets. The key ideas of formal languages originate in linguistics. Linguistic objects are structured objects, and a grasp of the meaning of a sentence depends crucially on the speaker's or listener's ability to recover its structure, an ability which is likely to be unconscious for a native speaker of the language in question. A computational device which infers structure from grammatical strings of words is known as a "parser." Formal languages, such as propositional calculus, Java, or Lisp, have well-defined unique grammars. Natural languages, however, are riddled with ambiguities at every level of description, from the phonetic to the sociological. Jokes for instance usually exploit ambiguities, in particular word plays: *"Time flies like an arrow, fruit flies like a banana"* puns on the different meanings of the words flies and like. So, formal languages do not seem appropriate for telling jokes.

Definition 2.1. *(Alphabet, word, language)* An *alphabet* Σ is a nonempty finite set of symbols. With $\Sigma^0 := \{\varepsilon\}$ denoting the set of the *empty word* ε, and $\Sigma^1 := \Sigma$ we define

$$\Sigma^{n+1} := \{xy |\ x \in \Sigma,\ y \in \Sigma^n\} \qquad (n > 0) \tag{2.1}$$

$$\Sigma^+ := \bigcup_{k=1}^{\infty} \Sigma^k \tag{2.2}$$

$$\Sigma^* := \bigcup_{k=0}^{\infty} \Sigma^k = \Sigma^+ \cup \{\varepsilon\}. \tag{2.3}$$

An element $w \in \Sigma^*$ is called a *word* over the alphabet Σ. The *length* of a word w, written $|w|$, is the number of symbols it contains. A subset $L \subset \Sigma^*$ is called a *language* over the alphabet Σ. [10, §4.1], [27, pp 13] $\lhd$

Remark 2.2. A usual data type of programming languages has as underlying alphabet Σ the set of all ASCII symbols or all Unicode symbols, where usually the symbols are called *characters* and the words over these alphabets are called *strings*. In bioinformatics, important alphabets are $\Sigma_{\text{DNA}} = \{\text{A, C, G, T}\}$ representing the four DNA nuclebases, $\Sigma_{\text{RNA}} = \{\text{A, C, G, U}\}$ representing the four RNA nuclebases, or $\Sigma_{\text{Amin}} = \{\text{A, C, D, }\ldots\}$ representing the 22 amino acids. The words over these alphabets are usually called *sequences*. $\Diamond$

Example 2.3. For the alphabet $\Sigma = \{a, b\}$ we have

$$\begin{aligned}
\Sigma^0 &= \{\varepsilon\} \\
\Sigma^1 &= \{a, b\} \\
\Sigma^2 &= \{aa, ab, ba, bb\} \\
&\vdots \\
\Sigma^+ &= \{a, b, aa, ab, ba, bb, aaa, aab, aba, \ldots\} \\
\Sigma^* &= \{\varepsilon, a, b, aa, ab, ba, bb, aaa, aab, aba, \ldots\}
\end{aligned}$$

Then $L = \{aa, ab\}$ is a language over Σ. ◊

Given a language L, is it possible to find a description by which all words can be systematically derived? If the formation of words of a language follow structured rules, they can be generated by a grammar.

Definition 2.4. A *grammar* G is a four-tuple (V, Σ, P, S) consisting of

- a finite set of *variables* V,
- a finite alphabet Σ of *terminals* satisfying $V \cap \Sigma = \emptyset$,
- a finite set P of *productions*, or *substitution rules*, each one having the form $l \to r$ with $l \in (V \cup \Sigma)^+$ and $r \in (V \cup \Sigma)^*$. Alternative options are listed with a logical OR sign $|$, e.g., $l \to a \mid b$ means that l may be substituted either by a or b.
- a start variable $S \in V$.

If two words $x, y \in (V \cup \Sigma)^*$ have the form $x = lur$ and $y = lvr$ with $l, r \in (V \cup \Sigma)^*$, then we say "$y$ can be *derived* from x" and write $x \Rightarrow y$ if the grammar contains a production of the form $u \to v$. Moreover, we write $x \overset{*}{\Rightarrow} y$ if y can be derived from x in finitely many steps. ◁

Definition 2.5. Any grammar G generates a language

$$L(G) = \{w \in \Sigma^* \mid S \overset{*}{\Rightarrow} w\} \tag{2.4}$$

consisting of the words which can be derived from the start variable S. ◁

Example 2.6. *(Dyck language)* Let be given $\Sigma = \{(,), [,]\}$, the variable set $V = \{S\}$ with the start variable S as its only element, and the production rules P

$$S \to \varepsilon \mid SS \mid [S] \mid (S). \tag{2.5}$$

Then $G_{\text{Dyck}} = (V, \Sigma, P, S)$ is a grammar. The language $D_2 := L(G_{\text{Dyck}})$ generated by it includes the word `()[()]()`, but `([)]` $\notin D_2$. ◊

Example 2.7. *(Simple English)* Let Σ be the Latin alphabet with the 26 lowercase letters, the space character and the full stop, the variables

$$V = \{\langle\text{sentence}\rangle, \langle\text{subject}\rangle, \langle\text{predicate}\rangle, \langle\text{object}\rangle, \langle\text{article}\rangle, \langle\text{adjective}\rangle, \langle\text{noun}\rangle\} \tag{2.6}$$

the start variable $\langle\text{sentence}\rangle$, and the production rules P

$$\begin{aligned}
\langle\text{sentence}\rangle &\to \langle\text{subject}\rangle\ \langle\text{predicate}\rangle\ \langle\text{object}\rangle\text{.} \\
\langle\text{subject}\rangle &\to \langle\text{article}\rangle\ \langle\text{adjective}\rangle\ \langle\text{noun}\rangle \\
\langle\text{article}\rangle &\to \texttt{a} \mid \texttt{the} \\
\langle\text{adjective}\rangle &\to \texttt{sweet} \mid \texttt{quick} \mid \texttt{small} \\
\langle\text{noun}\rangle &\to \texttt{duck} \mid \texttt{frog} \mid \texttt{mouse} \mid \texttt{hippopotamus} \\
\langle\text{predicate}\rangle &\to \texttt{likes} \mid \texttt{catches} \mid \texttt{eats} \\
\langle\text{object}\rangle &\to \texttt{cookies} \mid \texttt{chocolate} \mid \texttt{pizza}
\end{aligned} \tag{2.7}$$

Then $G_{\text{SE}} = (V, \Sigma, P, \langle\text{sentence}\rangle)$ is a grammar. The language $L(G_{\text{SE}})$ generated by it includes the sentences "`the small duck eats pizza.`" and "`a quick mouse catches cookies.`" ♢

Often a word of a given language is intended to represent a certain object other than a string, such as a graph, a polynomial, or a machine. However, it is easy to find an injective mapping from the set of objects to the strings. (This is what every computer software as a binary string does solving problems dealing with real-life objects such as addresses, persons, invoices, games, …). Our notation for the encoding of such an object A into its representation as a string is

$$\langle A \rangle \in \Sigma^*. \tag{2.8}$$

(Do not mix up this expression for encoding with the tag notation in the grammar of Example 2.7 above.) Usually, the underlying alphabet is $\Sigma = \{0, 1\}$. If we have several objects A_1, A_2, …, A_k, we denote their encoding into a single string by $\langle A_1, A_2, \ldots, A_k \rangle$. The encoding can be done in many reasonable ways. If the objects are well-defined, however, strings of different encodings can be translated into each other in a unique way [27, §3.3]. For instance, in the sequel we will denote the pair $\langle M, x \rangle$ for the encoding of a Turing machine M (an "algorithm") and an input string x.

2.1 The Chomsky hierarchy

In 1957 the US linguist Noam Chomsky postulated a framework by which formal grammars can be subdivided into four types, according to the following definitions.

Definition 2.8. (a) *Recursively enumerable*, or *phrase-structure grammars (type-0 grammars).* Any grammar according to Definition 2.4 is a *phrase-structure* or *recursively enumerable grammar*.

(b) *Context-sensitive grammars (type-1 grammars).* A grammar is *context-sensitive* if for any production $l \to r$ we have $|r| \geqq |l|$. In particular, a production can never lead to a shortening of a derived word.

(c) *Context-free grammars (type-2 grammars).* A grammar is *context-free* if for any production $l \to r$ we have $l \in V$. Therefore, the left hand side of a production always consists of a single variable.

(d) *Regular grammars (type-3 grammars).* A grammar is *regular* if for any production $l \to r$ we have $l \in V$ and $r \in \{\varepsilon\} \cup \Sigma V$. Therefore, a regular grammar is context-free and its right hand side is either empty or is a terminal followed by a variable. ◁

In a context-free grammar, the variable on the left side of the production can be rewritten by a certain word on the right regardless of the context it finds itself in the current sentential form: it is independent of its context. In contrast, in context-sensitive grammars a variable can be substituted by a certain variable only when it is in the context of other variables or terminals preceding or following it [Mo].

Definition 2.9. A language is called to be *of type n* if there exists a grammar of type n which generates L. The set of all type-n languages is denoted by $\mathscr{L}_n$, for $n = 0, 1, 2, 3,$ ◁

Lemma 2.10. *Let the following languages be given.*

$$L_{C3} := \{(ab)^n |\ n \in \mathbb{N}\}, \tag{2.9}$$
$$L_{C2} := \{a^n b^n |\ n \in \mathbb{N}\}, \tag{2.10}$$
$$L_{C1} := \{a^n b^n c^n |\ n \in \mathbb{N}\}, \tag{2.11}$$
$$L_{C0} := \{\langle M, x\rangle |\ M \text{ is a Turing machine halting on string } x\}, \tag{2.12}$$
$$L_{\mathrm{d}} := \{\langle M, x\rangle |\ M \text{ is a Turing machine not accepting } x\}. \tag{2.13}$$

For $k = 0, 1, 2, 3$ then L_{C_k} is a language of type k, but L_{Ck} is not a language of type $(k+1)$ for $k \leqq 2$. In other words, L_{C3} is regular, L_{C2} is context-free but not regular, L_{C1}

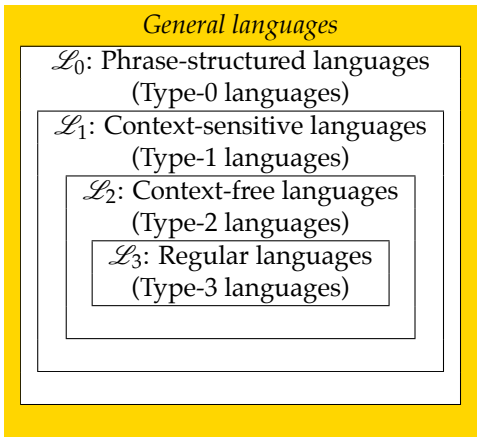


Figure 2.1: The Chomsky hierarchy

is context-sensitive but not context-free, and L_{C0} is phrase-structured but not context-sensitive. We thus have the strict set inclusions

$$\mathscr{L}_3 \subset \mathscr{L}_2 \subset \mathscr{L}_1 \subset \mathscr{L}_0, \tag{2.14}$$

Moreover, L_{d} is not phrase-structured, i.e., $L_{\mathrm{d}} \notin \mathscr{L}_0$.

Proof. The proof is given by the following examples 2.12–2.15, Lemma 2.16, and the corollaries 2.19, 2.28, 2.35 below. [10, §4] □

Remark 2.11. The language L_{C0} in (2.12) is often also called the "halting problem," and the language L_{d} in (2.13) the "diagonal language." ♢

Example 2.12. Let $G_3 = (\{B, C, S\}, \{a, b\}, P, S)$ be a grammar, with the productions P given by

$$\begin{aligned} S &\to aB \\ B &\to bC \\ C &\to \varepsilon \mid aB. \end{aligned}$$

Then G_3 is a regular grammar since the left hand side of each production consists of a single variable and each right hand side is either empty a terminal followed by a variable. Hence the language $L(G_3)$ generated by G_3 is given by

$$L(G_3) = \{ab, abab, ababab, \ldots\}, \tag{2.15}$$

i.e., $L(G_3) = L_{C3}$. In particular, $L_{C3} \in \mathscr{L}_3$, that is, L_{C3} is regular. ♢

Example 2.13. Let $G_2 = (\{S\}, \{a, b\}, P, S)$ be a grammar, with the productions P given by

$$S \to aSb \mid ab.$$

Then G_2 is a context-free grammar according to Definition 2.8. Hence the language $L(G_2)$ generated by G_2 is given by

$$L(G_2) = \{ab, aabb, aaabbb, \ldots\}, \tag{2.16}$$

i.e., $L(G_2) = L_{C2}$. In particular, $L_{C2} \in \mathscr{L}_2$, that is, L_{C2} is context-free. ♢

Example 2.14. Let $G_1 = (\{A,\ B,\ C,\ S\},\ \{a,\ b,\ c\},\ P,\ S)$ be a grammar, with the productions P given by

$$\begin{aligned} S &\to SABC \mid ABC, \\ BA &\to AB,\ \ CB \to BC,\ \ CA \to AC, \\ AB &\to ab,\ \ BC \to bc, \\ Aa &\to aa,\ \ Ab \to ab,\ \ bB \to bb,\ \ cC \to cc. \end{aligned}$$

Then G_1 is a context-sensitive grammar according to Definition 2.8. Hence the language $L(G_1)$ generated by G_1 is given by

$$L(G_1) = \{abc, aabbcc, aaabbbccc, \ldots\}, \tag{2.17}$$

i.e., $L(G_1) = L_{C1}$. In particular, $L_{C1} \in \mathscr{L}_1$, that is, L_{C1} is context-sensitive. ♢

Example 2.15. For the halting problem we have $L(G_0) \in \mathscr{L}_0$. For the proof we first have to show that there exists a Turing machine accepting the halting problem, namely the machine $\tilde{U}$ working on input $\langle M, x\rangle$ where M is a Turing machine and x a string, defined as

$$\tilde{U}(\langle M, x\rangle) = \textit{accept} \text{ if } M \text{ halts on } x. \tag{2.18}$$

($\tilde{U}$ is a slight modification of a universal Turing machine). Next we have to construct a grammar which simulates the actions of $\tilde{U}$ and generates a terminal string if $\tilde{U}$ accepts; such a grammar is given, for instance, in [12, Satz 9.4]. ◊

Lemma 2.16. *The diagonal language L_{d} in (2.13) is not phrase-structured.*

Proof. [11, §9.1.4] According to the definition, we have to prove that there exists no Turing machine which accepts L_{d}. One key property of Turing machines is that we can enumerate them such that M_1, M_2, … exhaust the set of all possible Turing machines, and assign a number string $w_i \in \{0,1\}^*$ to M_i for $i \in \mathbb{N}$, the *Gödel code*, [10, §6.1.5.6], [11, §9.1.2]. Thus L_{d} is the set of words w_i such that w_i is not in the language $L(M_i)$ of all strings accepted by M_i.

Assume that there exists a Turing machine M by which the diagonal language is accepted, i.e., $L_{\mathrm{d}} = L(M)$. Then there exists a number $i \in \mathbb{N}$ such that $M_i = M$ and a Gödel code w_i coding M_i. Now, is then $w_i \in L_{\mathrm{d}}$? (i) If $w_i \in L_{\mathrm{d}}$ then M_i excepts w_i; but then, by definition of L_{d}, w_i is not in L_{d} because L_{d} only contains those w_j such that M_j does *not* accept w_j. (ii) However, if $w_i \notin L_{\mathrm{d}}$, then M_i does not accept w_i and thus, by definition of L_{d}, w_i *is* in L_{d}. Since w_i thus can neither be in L_{d} nor fail to be in L_{d}, we conclude that there is a contradiction of our assumption that M exists. Hence L_{d} is not phrase-structured. □

2.2 Regular languages

Example 2.17. Let be $G_{abc} = (\{S, B, C\}, \{a, b, c\}, P, S)$ with the production P given by

$$\begin{aligned} S &\to aS \mid aB \\ B &\to bB \mid bC \\ C &\to cC \mid c. \end{aligned} \tag{2.19}$$

Then G_{abc} is a regular grammar. The language $L_{abc} := L(G_{abc})$ generated by it is

$$L_{abc} = \{a^i b^j c^k \mid i, j, k \in \mathbb{N}\}. \tag{2.20}$$

It is thus a regular language. ◊

The following theorem presents a technique for proving that a given language is *not* regular. The "pumping lemma" states that every regular language has a "pumping length" such that all longer strings in the language can be pumped by periodically repeating substrings.

Theorem 2.18 (Pumping lemma for regular languages). *A regular language L has a number $p \in \mathbb{N}$ where, if $s \in L$ is any word in L of length at least p, then it may be divided into three pieces*

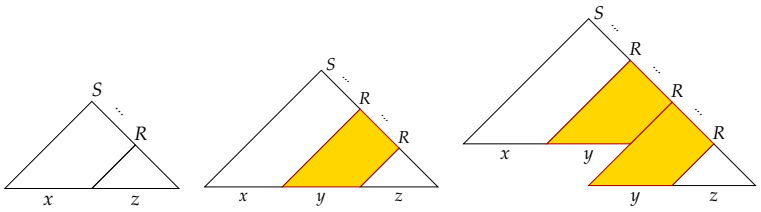
$$s = xyz \tag{2.21}$$

satisfying the conditions

$$xy^iz \in L \quad \text{for each } i \in \mathbb{N}_0, \tag{2.22}$$
$$|y| > 0, \tag{2.23}$$
$$|xy| \leqq p. \tag{2.24}$$

The number p is called pumping length *and is given as $p = |V| + 1$, where $|V|$ is the number of variables of a grammar G generating L.*

Proof. Regular languages have a certain word structure. If the number of derivation steps exceeds the limit p, then by the "pigeonhole principle"[1] at least one variable is repeated, say R. Then given the derivation chain $S \Rightarrow \ldots \Rightarrow R \Rightarrow \ldots \Rightarrow R \Rightarrow \ldots$, the derivation block between the two R's may be repeated arbitrarily often, as depicted in the following diagram for the words xz, xyz, xy^2z:



Therefore we may cut s into three pieces xyz and repeat the second piece and obtain a word still in the language, i.e., $xy^iz \in L$ for any $i = 0, 1, 2, \ldots$. For more details on the proof see [10, §4.3.2]. □

Corollary 2.19. *The language L_{C2} in Equation (2.10) is not regular.*

Proof. The proof is by contradiction. Assume that L_{C2} is regular. Then there exists a pumping length p according to the pumping lemma 2.18. Select the word $s = a^pb^p$. Clearly $s \in L_{C2}$ and $|s| > p$, and we can apply the pumping lemma to s. So we can find three words in L_{C2} such that $a^pb^p = xyz$, where condition (2.23) guarantees that y is nonempty. So there are three possible cases:

1. y consists only of a's: In this case the string $xyyz$ has more a's than b's and therefore is not in L_{C1}, contradiciting (2.22).

2. y consists only of b's: This case gives also a contradiction to (2.22).

3. y consists of both a's and b's: Then xy^2z may contain equal numbers of a's and b's, but not in the correct order since there are some b's before a's. Hence it cannot be in L_{C2}.

Therefore, it is impossible to subdivide s into xyz such that all three conditions (2.22) – (2.24) are satisfied. Hence our introductory assumption must be false, i.e., L_{C2} cannot be regular. □

[1] The *pigeonhole principle* is a fancy name for the rather obvious fact that if p pigeons are placed into fewer than p holes, some hole has to have more than one pigeon in it.

2.2.1 Regular expressions

Regular language can be elegantly described by regular expressions.

Definition 2.20. Given an arbitrary alphabet Σ, the set Regex_Σ of *regular expressions* is defined recursively by the following rules.

- $\emptyset, \varepsilon \in \mathrm{Regex}_\Sigma$, $\Sigma \subset \mathrm{Regex}_\Sigma$,
- If $r \in \mathrm{Regex}_\Sigma$ then $(r) \in \mathrm{Regex}_\Sigma$ and $r^* \in \mathrm{Regex}_\Sigma$.
- If $r, s \in \mathrm{Regex}_\Sigma$ then rs, $(r \mid s) \in \mathrm{Regex}_\Sigma$.

Here the operator $\mid$ is the logical OR operator, the operator * is the "Kleene star" or "zero-or-more" operator, where

$$r^* \;:=\; \varepsilon \mid r \mid rr \mid rrr \mid \;\ldots, \tag{2.25}$$

and the operation $(\cdot)$ is called "grouping." ◁

Lemma 2.21. *The neutral elements of* Regex_Σ *with respect to the* $\mid$ *operation are* $\emptyset$ *and* ε*, i.e.,*

$$r \mid \emptyset \;=\; \emptyset \mid r \;=\; r \mid \varepsilon \;=\; \varepsilon \mid r \;=\; r. \tag{2.26}$$

Moreover, for $r, s, t \in \mathrm{Regex}_\Sigma$ *we have the law of idempotence,*

$$r \mid r \;=\; r, \tag{2.27}$$

the law of commutativity,

$$r \mid s \;=\; s \mid r, \tag{2.28}$$

and the laws of distributivity,

$$r\,(s \mid t) \;=\; rs \mid rt, \tag{2.29}$$
$$(s \mid t)\,r \;=\; sr \mid tr. \tag{2.30}$$

Definition 2.22. A regular expression $r \in \mathrm{Regex}_\Sigma$ generates a language $L(r) = L_{\mathrm{Regex}_\Sigma}(r)$ by the following recursive definition:

$$\begin{aligned} L(\emptyset) &= \emptyset, \\ L(\varepsilon) &= \{\varepsilon\}, \\ L(a) &= \{a\} \qquad (a \in \Sigma), \\ L(rs) &= L(r)L(s), \\ L\,((r \mid s)) &= L(r) \,\cup\, L(s), \\ L(r^*) &= L(r)^*, \end{aligned}$$

where $r, s \in \mathrm{Regex}_\Sigma$. ◁

Theorem 2.23. *The language* $L(ab)$ *generated by a regular expression* $ab \in \mathrm{Regex}_{\{a,b\}}$ *over the alphabet* $\{a, b\}$ *is exactly* L_{C3} *in (2.9).*

Proof. $L(a, b) = ab(ab)^*$. □

2.3 Context-free languages

Example 2.24. Let be $G_{\text{alg}} = (\{A, O, S\}, \{x, y, z, +, -, \cdot, /, (,)\}, P, S)$ with the productions P

$$\begin{aligned} S &\to AOA \mid (S) \\ A &\to S \mid x \mid y \mid z, \;\; O \to + \mid - \mid \cdot \mid / \end{aligned} \tag{2.31}$$

Then G_{alg} is a context-free grammar. The language $L_{\text{alg}} := L(G_{\text{alg}})$ generated by it is the set of all algebraic formulas (in "infix" notation) made of the three variables x, y, z, of the operations $+$, $-$, $\cdot$ and $/$, and of the round brackets. In particular,

$$(x + y) \cdot x - z \cdot y / (x - y) \in L_{\text{alg}}.$$

L_{alg} is thus a context-free language. ◊

Example 2.25. *(Simple HTML)* Let be $G_{html} = (V, \text{ASCII}, P, S)$ with the set of variables

$$V = \{S, H, T, B, A\}$$

(where A represents plain text and the other variables "tags") and the production P

$$\begin{aligned} S &\to \texttt{<html>}HB\texttt{</html>} \\ H &\to \varepsilon \mid \texttt{<head>}T\texttt{</head>} \\ B &\to \varepsilon \mid \texttt{<body>}A\texttt{</body>} \\ T &\to \varepsilon \mid \texttt{<title>}A\texttt{</title>} \\ A &\to \varepsilon \mid \texttt{A}A \mid \texttt{B}A \mid \ldots \mid \texttt{Z}A \mid \texttt{a}A \mid \texttt{b}A \mid \ldots \mid \texttt{z}A \mid \ldots \end{aligned}$$

Then G_{html} is a context-free grammar, and the language $L(G_{html})$ is context-free. For a more detailed study of HTML and XML see [11, §§5.3.3, 5.3.4]. ◊

Example 2.26. *(Simple Java)* Let be $G_{Java} = (V, \text{UTF-8}, P, S)$ with the set of variables

$$V = \{S, M, T, I, N, A\}$$

and the production P

$$\begin{aligned} S &\to \texttt{public class}\ N\ \texttt{\{}\ M\ \texttt{\}} \\ M &\to \varepsilon \mid T\ N\texttt{;} \mid \texttt{public}\ T\ N\ \texttt{\{}I\texttt{\}} \mid \texttt{public static}\ T\ N\ \texttt{\{}I\texttt{\}} \\ T &\to \texttt{void} \mid \texttt{boolean} \mid \texttt{int} \mid \texttt{double} \\ I &\to \varepsilon \mid A\texttt{;}\ I \\ N &\to \texttt{A}A \mid \texttt{B}A \mid \ldots \mid \texttt{Z}A \mid \texttt{a}A \mid \texttt{b}A \mid \ldots \mid \texttt{z}A \\ A &\to \varepsilon \mid \texttt{A}A \mid \ldots \mid \texttt{Z}A \mid \texttt{a}A \mid \ldots \mid \texttt{z}A \mid \texttt{0}A \mid \texttt{1}A \ldots \end{aligned}$$

(Note that "`;`" is a terminal!) Here S represents a class, M a class member (attribute or method), T a data type, I an instruction, N a non-empty string, and A a string. Then G_{Java} is a context-free grammar, and the language $L_{Java} := L(G_{Java})$ therefore is context-free, too. ◊

Similarly to the pumping lemma 2.18 for regular languages, the following theorem supplies a technique for proving that a given language is *not* context-free. The "pumping lemma" states that every context-free language has a "pumping length" such that all longer strings in the language can be pumped with *two* substrings.

Theorem 2.27 (Pumping lemma for context-free languages). *A context-free language L has a number $p \in \mathbb{N}$ where, if $s \in L$ is any word in L of length at least p, then it may be divided into five pieces*

$$s = uvxyz \tag{2.32}$$

satisfying the conditions

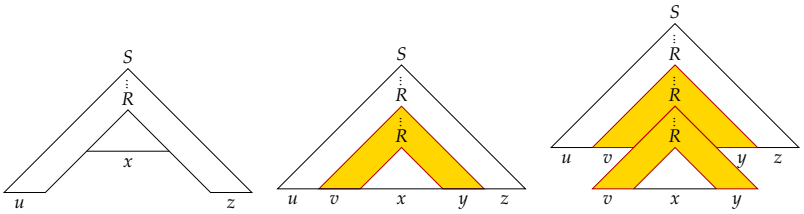
$$uv^i xy^i z \in L \quad \text{for each } i \in \mathbb{N}_0, \tag{2.33}$$
$$|vy| > 0, \tag{2.34}$$
$$|vxy| \leqq p. \tag{2.35}$$

The number p is called pumping length *and is given as*

$$p = k^{|V|+1} \tag{2.36}$$

where $k \geqq 2$ is the maximum number of symbols in the right-hand side of the production rules of a grammar G generating L, and $|V|$ is the number of variables of the grammar.

Proof. The main task of the proof is to show that any string s in L can be pumped and still remains in L. Because s is derivable from the grammar G, it has a parse tree. Since s is rather long, its parse tree is rather tall, that is, it must contain some long path from the from the start variable at its root to one of the terminal symbols at a leaf. On this long path then some variable symbol R must repeat because of the pigeonhole principle. As the following figure shows, this repetition allows us to replace the subtree under the second occurence of R and still get a legal parse tree for the words uxz, $uvxyz$, and uv^2xy^2z:



Therefore we may cut s into five pieces $uvxyz$ and repeat the second and fourth pieces and obtain a word still in the language, i.e., $uv^ixy^iz \in L$ for any $i = 0, 1, 2, \ldots$. For more details on the proof see [27, §2.3]. □

Corollary 2.28. *The language L_{C1} in Equation (2.11) is not context-free.*

Proof. The proof is by contradiction. We assume that L_{C1} is context-free. Then by the pumping lemma 2.27 there exists a pumping length p. Select the word $s = a^p b^p c^p$. Clearly $s \in L_{C1}$ and $|s| > p$, and we can apply the pumping lemma to s. So we can find five words in L_{C1} such that $a^p b^p c^p = uvxyz$. First, condition (2.34) stipulates that either v or y is nonempty. So there are two possible cases:

1. Both v and y contain only one type of alphabet symbol: Then v cannot contain both a's and b's or both b's and c's, and the same holds for y. In this case the string uv^2xy^2z cannot contain equal numbers of a's, b's, and c's, and therefore it is not in L_{C1}, contradiciting (2.33).

2. Either v or y contain more than one type of symbol: Then uv^2xy^2z may contain equal numbers of the three alphabet symbols a, b, c, but not in the correct order. Hence it cannot be in L_{C1}.

Therefore, it is impossible to subdivide s into $uvxyz$ such that all three conditions (2.33) – (2.35) are satisfied. Therefore, our introductory assumption must be false, i.e., L_{C1} cannot be context-free. □

More examples of languages for which the pumping lemma can be applied to prove that they are not context-free are $\{a^i b^j c^k |\ 0 \leqq i \leqq j \leqq k\}$ or $\{ww |\ w \in \{a, b\}^*\}$, as shown in [27, §2.3]. However, the pumping lemma only presents necessary conditions for non-context-free languages, but not sufficient conditions. That is, there may be languages which satisfy the conditions of the pumping lemma but are not context-free. An example is given in [10, §4.4]. However, a sharper necessary condition is known, called Ogden's lemma.

Theorem 2.29 (Ogden's lemma). *A context-free language L has a number $p \in \mathbb{N}$ where, if $s \in L$ is any word in L of length at least p, then it satisfies the following property: marking at least p symbols in s, the word s may be divided into five pieces*

$$s = uvwxy \tag{2.37}$$

satisfying the conditions

- *at least one symbol in vx is marked,*
- *at most p symbols are marked in vwx,*
- *$uv^i wx^i y \in L$.*

Proof. [13] □

Example 2.30. Let $G_{\text{Luk}} = (\{S\}, \{a, b\}, P, S)$ be a grammar, with the productions P given by

$$S \rightarrow aSS \mid b.$$

Then G_{Luk} is a comtext-free grammar according to Definition 2.8, but not regular. The language it generates is called the *Łukasiewicz language*. ♢

2.3.1 Linear languages

Definition 2.31. A context-free grammar is called *linear* if the right hand side of each of its productions contains at most one variable. A language is called *linear* if it can be generated by a linear grammar [12, p 109]. ◁

Example 2.32. *(Palindrome language)* The grammar $G_{\text{lin}} = (\{S\}, \{a, b\}, P, S)$ with the productions

$$S \rightarrow aSa \mid bSb \mid a \mid b \mid \varepsilon \tag{2.38}$$

is linear. It generates the palindrome language $G_{\text{pal}} = \{ww^R |\ w \in \{a, b\}^*\}$. ♢

The linear languages are not closed under the $*$ operation [12, p 150].

2.4 Context-sensitive languages

Nearly every imaginable language is context-sensitive. The only known proofs of the existence of not context-sensitive languages are based on the undecidability of the halting problem [11, §9.3].

Example 2.33. Let $G_{\text{exp}} = (\{A, B, C, D, E, F, G, S\}, \{a\}, P, S)$ be a grammar, with the productions P given by

$$S \to AB,\ \ B \to C \mid D,\ \ aC \to Ca,\ \ AC \to aa,\ \ AD \to AaaB,$$
$$aD \to EF,\ \ aE \to Ea,\ \ AE \to AaG,\ \ Ga \to aaG,\ \ GF \to aaaB.$$

Then G_{exp} is a context-sensitive grammar according to Definition 2.8. Hence the language $L(G_{\text{exp}})$ generated by G_0 is given by

$$L(G_{\text{exp}}) = \{aa, aaaa, a^8, a^{16}, \ldots\}, \tag{2.39}$$

i.e., $L(G_{\text{exp}}) \in \mathscr{L}_1$. [12, Ex. 9.4 adapted] ♢

Proof. First we prove by induction over n that we can always derive

$$S \overset{*}{\Rightarrow} Aa^{2^n-2}B \tag{2.40}$$

for $n \in \mathbb{N}$. For $n = 1$ we have clearly $S \to AB$ by the first production rule, and from (2.40) for $n \in \mathbb{N}$ we achieve the respective equation (2.40) for $n + 1$ by

$$\begin{aligned} Aa^{2^n-2}B &\Rightarrow Aa^{2^n-2}D \Rightarrow Aa^{2^n-3}EF \overset{*}{\Rightarrow} AEa^{2^n-3}F \\ &\Rightarrow AaGa^{2^n-3}F \overset{*}{\Rightarrow} Aa^{2^{n+1}-5}GF \to Aa^{2^{n+1}-2}B \end{aligned} \tag{2.41}$$

As a second derivation we directly see that

$$Aa^{2^n-2}B \Rightarrow Aa^{2^n-2}C \overset{*}{\Rightarrow} ACa^{2^n-2} \Rightarrow a^{2^n}. \tag{2.42}$$

With (2.40) this gives $S \overset{*}{\Rightarrow} a^{2^n}$ for any $n \in \mathbb{N}$. □

Example 2.34. [10, Abb. 4.26] Let $G'_{\text{exp}} = (\{D, L, S\}, \{a\}, P, S)$ be a grammar, with the productions P given by

$$S \to SD,\ \ SD \to LaD,\ \ aD \to Daa,\ \ LD \to L,\ \ L \to \varepsilon.$$

Then G'_{exp} is a phrase-structured grammar according to Definition 2.8, but not context-sensitive because of the second to last substitution rule. However, the language $L(G'_{\text{exp}})$ generated by G'_{exp} is given by

$$L(G'_{\text{exp}}) = \{a, aa, aaaa, a^8, a^{16}, \ldots\}, \tag{2.43}$$

i.e., $L(G'_{\text{exp}}) = L_{\text{exp}}$ in Example 2.33. Therefore, a non-context-sensitive grammar can generate a context-sensitive language. ♢

Corollary 2.35. *The language L_{C0} in Equation (2.12) is not context-sensitive.*

Proof. We prove by contradiction that $H := L_{C0}$ is not decidable ("decidable" being the same as "recursive" in [11]). Since every context-sensitive language is decidable [12, Satz 9.7], this proves the assertion. Assume that L_{C0} is decidable. Then in paricular the Turing machine $\tilde{U}$ in (2.18) exists, and we can construct the Turing machine D with input $\langle M \rangle$, where M is an arbitrary Turing machine, which runs H as a subroutine with the input $\langle M, \langle M \rangle \rangle$ and does the opposite of $\tilde{U}$, that is, D rejects if M accepts, and accepts if M rejects:

$$\begin{aligned} D = \ & \text{"On input } \langle M \rangle \text{, where } M \text{ is a Turing machine,} \\ & \quad 1. \text{ Run } \tilde{U} \text{ on input } \langle M, \langle M \rangle \rangle; \\ & \quad 2. \text{ Output the opposite of } \tilde{U}." \end{aligned}$$

Therefore,

$$D(\langle M \rangle) = \begin{cases} \textit{accept} & \text{if } M \text{ does not accept } \langle M \rangle, \\ \textit{reject} & \text{if } M \text{ accepts } \langle M \rangle. \end{cases} \tag{2.44}$$

and, applied to itself,

$$D(\langle D \rangle) = \begin{cases} \textit{accept} & \text{if } D \text{ does not accept } \langle D \rangle, \\ \textit{reject} & \text{if } D \text{ accepts } \langle D \rangle. \end{cases} \tag{2.45}$$

No matter what D does, applied on itself as input it is forced to output the opposite, which is a contradiction. Thus the halting problem L_{C0} cannot be decidable. □

For the solution of the decidability of the halting problem it was essential to apply $\tilde{U}(\langle M, \langle M \rangle \rangle)$, i.e., to run a Turing machine M on its own description, $M(\langle M \rangle)$. This is similar to running a program with itself as input, something which does occasionally occur in practice. For example, a compiler is a program which translates other programs; a compiler for the language Java may itself be written in Java, so it could be running on itself.

2.4.1 Indexed languages

Among the many proposed generalizations of context-free languages the "index languages," introduced by Aho in 1968, seem to be rather well adapted to natural languages [2], [GM, §4-1.6.3]. They use a set of indices working as a recursive stack.

Definition 2.36. [12, §14.3] An *indexed grammar* is a five-tuple (V, Σ, I, P, S) where V is a set of variables with the start variable $S \in V$, Σ is an alphabet, I is a set of indices, and the set P of productions having the form

$$A_{..} \to \alpha_{..}, \qquad A_{..} \to B_{f..}, \qquad A_{f..} \to \alpha_{..} \tag{2.46}$$

where $A, B \in V$, $f \in I$, $.. \in I^*$, and $\alpha \in (V \cup \Sigma)^*$. Whenever a production of the first kind is applied to A, the index string $..$ is attached to *each* variable in α (but not its terminals). For productions of the second kind, the index is added to the

front of the A's index string and attached to B. By productions of the third kind, the index is removed from the index string, and the index reminder is distributed over the variables, as before. A language is called *indexed* if it can be generated by an indexed grammar. ◁

Example 2.37. [12, §14.3] Let $G = (\{S, T, A, B, C\}, \{a, b, c\}, \{f, g\}, P, S\}$ be an indexed grammar with the productions

$$\begin{array}{llll} S \to T_g, & A_f \to aA, & A_g \to a, \\ T \to T_f, & B_f \to bB, & B_g \to b, \\ T \to ABC, & C_f \to cC, & C_g \to c. \end{array}$$

For instance we have

$$\begin{aligned} S \Rightarrow T_g \Rightarrow T_{fg} \Rightarrow A_{fg}B_{fg}C_{fg} \\ \Rightarrow aA_gB_{fg}C_{fg} \Rightarrow aaB_{fg}C_{fg} \Rightarrow aabB_gC_{fg} \\ \Rightarrow aabbC_{fg} \Rightarrow aabbcC_g \Rightarrow aabbcc, \end{aligned}$$

and thus in general

$$S \overset{*}{\Rightarrow} T_{f^ig} \Rightarrow A_{f^ig}B_{f^ig}C_{f^ig} \overset{*}{\Rightarrow} a^{i+1}b^{i+1}c^{i+1} \tag{2.47}$$

for $i \in \mathbb{N}_0$. Therefore, the language $L(G)$ generated by G is exactly L_{C1} in (2.11). According to Example 2.14 and Corollary 2.28, $L(G)$ is context-sensitive but not context-free. ◊

Theorem 2.38. *A context-free grammar is representable as an indexed grammar. In turn, an indexed grammar is context-sensitive.*

Proof. A context-free grammar $A \to \alpha$ directly induces an indexed grammar with an empty index set $I = \emptyset$. The second assertion is proved in [12, Theorem 14.7]. □

Global Index Languages

Global index grammars use the stack of indices as a global control structure during the entire derivation of a word. Global index grammars are a kind of regulated rewriting mechanism with global context and history of the derivation as the main characteristics of its regulating device [2, §1.2.2].

Definition 2.39. [2, §1.2.2] A *global index grammar* is a five-tuple (V, Σ, I, P, S) where V, Σ, I are pairwise disjoint sets, V denoting the set of variables, Σ the set of terminals, I the set of stack indices, $S \in V$ the start symbol, and P the set of productions of the form

$$A \to \alpha, \qquad A\cdot\cdot[..] \to a\alpha\cdot\cdot[i..] \qquad A\cdot\cdot[j..] \to \alpha\cdot\cdot[..]$$

with $A \in V$, $\alpha, \cdot\cdot \in (V \cup \Sigma)^*$, $a \in \Sigma$, $i \in I^*$, $j \in I$, and $.. \in I^*$. In contrast to a general indexed grammar as given by Definition 2.36, the global index $[\cdot\cdot]$ is *not* attached to each of its member variables. The second to last production is called a *push operation*, the last one a *pop operation*. The *derivation* $\Rightarrow$ is defined as in Definition 2.4. Then $L(G)$ is the language of G defined as $L(G) = \{w \in \Sigma^* \mid S[] \overset{*}{\Rightarrow} w[]\}$, analogously to (2.4). The empty stack often is simply omitted, i.e., $\alpha[] = \alpha$. ◁

Example 2.40. *(Multiple copies)* [2, §1.2.2] Let $G_{\text{mc}} = (\{S, R, A, B, L\}, \{a, b\}, \{i, j\}, P, S)$ with the productions

$$\begin{aligned} S &\to AS \mid BS \mid RS \mid L, \\ R{\cdot\cdot}[i..] &\to RA{\cdot\cdot}[..], \quad R{\cdot\cdot}[j..] \to RB{\cdot\cdot}[..], \quad R{\cdot\cdot}[] \to \varepsilon{\cdot\cdot}[], \\ A{\cdot\cdot}[..] &\to a{\cdot\cdot}[i..], \quad B{\cdot\cdot}[..] \to b{\cdot\cdot}[j..], \\ L{\cdot\cdot}[i..] &\to La{\cdot\cdot}[..], \quad L{\cdot\cdot}[j..] \to Lb{\cdot\cdot}[..], \quad L{\cdot\cdot}[] \to \varepsilon{\cdot\cdot}[]. \end{aligned}$$

For instance,

$$\begin{aligned} S[] &\Rightarrow AS[] \Rightarrow aS[i] \Rightarrow aBS[i] \Rightarrow abS[ji] \Rightarrow abRS[ji] \\ &\Rightarrow abRBS[i] \Rightarrow abRABS \Rightarrow abABS \Rightarrow abaBS[i] \\ &\Rightarrow ababS[ji] \Rightarrow ababL[ji] \Rightarrow ababLb[i] \Rightarrow ababLab[] \Rightarrow abababab[] \end{aligned}$$

Therefore, $L(G_{\text{mc}}) = \{ww^+ \mid w \in \{a, b\}^*\}$. $L(G_{\text{mc}})$ is context-sensitive, but not context-free as can be seen by the pumping lemma 2.27. The language $L(G_{\text{mc}})$ is relevant for coordination in natural languages. ◊

Example 2.41. *(Bach language)* Let $G_{\text{Bach}} = (\{S, D, F, L\}, \{a, b, c\}, \{i, j, k, l, m, n\}, P, S)$ with the productions

$$\begin{aligned} S &\to FS \mid DS \mid LS \mid \varepsilon, \\ F{\cdot\cdot}[..] &\to c{\cdot\cdot}[i..], \quad F{\cdot\cdot}[..] \to b{\cdot\cdot}[j..], \quad F{\cdot\cdot}[..] \to a{\cdot\cdot}[k..], \\ D{\cdot\cdot}[..] &\to aSb{\cdot\cdot}[l..] \mid bSa{\cdot\cdot}[l..], \quad D{\cdot\cdot}[..] \to aSc{\cdot\cdot}[m..] \mid cSa{\cdot\cdot}[m..], \\ D{\cdot\cdot}[..] &\to bSc{\cdot\cdot}[n..] \mid cSb{\cdot\cdot}[n..], \\ D{\cdot\cdot}[i..] &\to aSb{\cdot\cdot}[..] \mid bSa{\cdot\cdot}[..], \quad D{\cdot\cdot}[j..] \to aSc{\cdot\cdot}[..] \mid cSa{\cdot\cdot}[..], \\ D{\cdot\cdot}[k..] &\to bSc{\cdot\cdot}[..] \mid cSb{\cdot\cdot}[..], \\ L{\cdot\cdot}[l..] &\to c[..], \quad L{\cdot\cdot}[m..] \to b[..], \quad L{\cdot\cdot}[n..] \to a[..], \end{aligned}$$

For instance,

$$\begin{aligned} S &\Rightarrow FS \Rightarrow FDS \Rightarrow FDLS \Rightarrow FDL \Rightarrow cDL[i] \\ &\Rightarrow caScL[mi] \Rightarrow caScb[i] \Rightarrow caDScb[i] \Rightarrow caaSbScb \Rightarrow caabcb \end{aligned}$$

In total we have $L(G_{\text{Bach}}) = \{w \mid |w|_a = |w|_b = |w|_c\}$. (Here $|w|_a$ denotes the number of symbols a in the word w.) This is the *Bach language*, or *MIX language*, which is conjectured to be *not* an indexed language [2, §1.2.2]. ◊

Example 2.42. *(Dependent branches)* [2, §1.2.2] Let $G_{\text{sum}} = (\{S, R, F, L\}, \{a, b, c, d, e, f\}, \{i\}, P, S)$ with the productions

$$\begin{aligned} S{\cdot\cdot}[..] &\to aSf{\cdot\cdot}[i..] \mid R{\cdot\cdot}[i..], \quad R \to FL \mid F \mid L, \\ F{\cdot\cdot}[i..] &\to bFc{\cdot\cdot}[..] \mid bc{\cdot\cdot}[..], \quad L{\cdot\cdot}[i..] \to dLe{\cdot\cdot}[..] \mid de{\cdot\cdot}[..], \end{aligned}$$

For instance,

$$\begin{aligned} S &\Rightarrow aSf[i] \Rightarrow aaSff[ii] \Rightarrow aaRff[ii] \Rightarrow aaFLff[ii] \\ &\Rightarrow aabcLff[i] \Rightarrow aabcdeff \end{aligned}$$

Therefore, $L(G_{\text{sum}}) = \{a^n b^m c^m d^l e^l f^n \mid n = m + l \geqq 1\}$. This language cannot be generated by a linear indexed grammar, as introduced in Definition 2.43. ◊

Linear Indexed Languages

Definition 2.43. A *linear indexed grammar* is a five-tuple (V, Σ, I, P, S) where V is a set of variables, Σ a set of terminals, I a set of indices, $S \in V$ a start variable, and P a set of productions of the form

$$A[..] \to \alpha B[..]\gamma, \qquad A[i..] \to \alpha B[..]\gamma, \qquad A[..] \to \alpha B[i..]\gamma, \tag{2.48}$$

where $A, B \in V$, $\alpha, \gamma \in (V \cup \Sigma)^*$, $i \in I$, and $.. \in I^*$. ◁

Linear indexed grammars are therefore indexed grammars with the additional constraint in the productions that the stack of indices can be transmitted only to at most one variable. As a consequence they are "semilinear."

Example 2.44. [2, §1.2.1] Let $G_{wcw} := \{\{S, R\}, \{a, b, c\}, \{i, j\}, P, S\}$ with the productions P given by

$$S[..] \to aS[i..], \quad S[..] \to bS[j..], \quad S[..] \to cR[..],$$
$$R[i..] \to R[..]a, \quad R[j..] \to R[..]b, \quad R[] \to \varepsilon.$$

Then $L(G_{wcw}) = \{wcw |\, w \in \{ab\}^*\}$. ◊

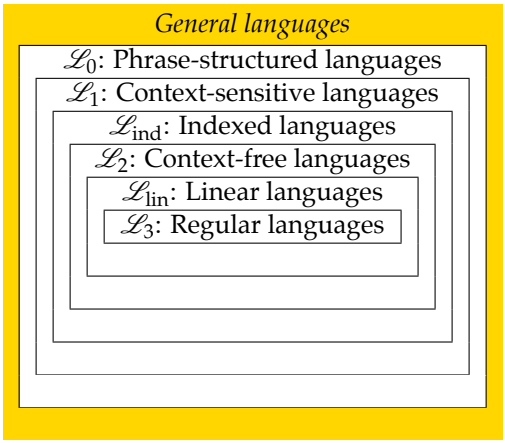


Figure 2.2: Hierarchy of languages, including the Chomsky hierarchy Figure 2.1.

2.5 Languages and machines

There is a close relationship between formal languages and finite automata which "accept" them. In this section we will briefly look at the big picture.

Definition 2.45. A *deterministic finite state machine*, often shortly called *machine* or *automaton*, is a five-tuple $(S, \Sigma, \delta, E, s_0)$ consisting of

- a finite set of states S,
- a finite input alphabet Σ,

- a state transition function $\delta : S \times \Sigma \to S$,
- a set of final states $E \subseteq S$,
- an initial state $s_0 \in S$.

For a given input word $w_0 w_1 \dots w_n \in \Sigma^*$ we write and two states s, $s' \in S$ we write

$$(s, w_0 w_1 \dots w_n) \to (s', w_1 \dots w_n) \tag{2.49}$$

if $s' = \delta(s, w_0)$. ◁

Initially, any automaton is in its initial state s_0. If it gets the input word $w = w_0 w_1 \dots w_n \in \Sigma^*$, it runs through the states $s_0, s_1, \dots, s_{n+1}$ with $s_{i+1} = \delta(s_i, w_i)$.

Definition 2.46. An automaton $A = (S, \Sigma, \delta, E, s_0)$ is said to *accept* an input word if the final state induced by it is an element of the set E of final states, i.e., if for $w = w_0 w_1 \dots w_n \in \Sigma^*$ we have $s_{n+1} = \delta(s_n, w_n) \in E$. The language $L(A)$ which is *accepted by* A is given by

$$L(A) = \{w_0 \dots w_n \in \Sigma^* |\ \delta(\dots(\delta(s_0, w_0), w_1), \dots), w_n) \in E\},$$

or equivalently

$$L(A) = \{w \in \Sigma^* |\ \exists s_f \in E \text{ with } (s_0, w) \to (s_f, \varepsilon)\}, \tag{2.50}$$

i.e., $L(A)$ contains all the words for which the final state is permissible. ◁

Definition 2.47. A *pushdown automaton* is a five-tuple $(S, \Sigma, \Gamma, \delta, s_0)$ consisting of

- a finite set of states S,
- a finite input alphabet Σ with $\varepsilon \notin \Sigma$,
- a finite *stack alphabet* Γ with $\perp \in \Gamma$,
- a transition relation $\delta : S \times (\Sigma \cup \{\varepsilon\}) \times \Gamma \to \mathscr{P}(S \times \Gamma^*)$,
- an initial state $s_0 \in S$.

Here $\mathscr{P}(A)$ for an arbitrary set A denotes the *power set* of A, i.e., the set of all possible subsets of A. In particular, $\emptyset$, $A \in \mathscr{P}(A)$, and $|\mathscr{P}(A)| = 2^{|A|}$, [9, 4.19]. For $\gamma \in \Gamma$, the words $w \in \Sigma^*$ and $\kappa \in \Gamma^*$ and two states s, $s' \in S$ we write

$$(s, w, \gamma\kappa) \to (s', w, \kappa' a \kappa) \tag{2.51}$$

if $(s', \kappa') \in \delta(s, \varepsilon, \gamma)$, and

$$(s, \sigma w, \gamma\kappa) \to (s', w, \kappa' a \kappa) \tag{2.52}$$

if $(s', \kappa') \in \delta(s, \sigma, \gamma)$ with $\sigma \in \Sigma$. ◁

Definition 2.48. A pushdown automaton $A = (S, \Sigma, \Gamma, \delta, s_0)$ is said to *accept* an input word if

$$L(A) = \{w \in \Sigma^* | \, (s_0, w, \bot) \to (s, \varepsilon, \varepsilon)\}, \tag{2.53}$$

i.e., $L(A)$ contains all the words for which the final state leaves an empty stack. $\lhd$

Although the definitions of the accepted language of a deterministic machine (2.50) and a pushdown automaton (2.53) formally resemble to one other, the final state in a pushdown automaton is *not* uniquely determined.

Definition 2.49. [1, p 12], [10, pp 267, 274] A *(deterministic) Turing machine M* is a seven-tuple $(S, \Sigma, \Gamma, \delta, s_0, \llcorner, E)$ consisting of

- a finite set S of possible states (in which M's "register" can be),
- a finite input alphabet Σ,
- a *tape alphabet* $\Gamma \supseteq \Sigma$ containing a blank symbol $\llcorner \notin \Sigma$,
- a transition function $\delta : S \times \Gamma \to S \times \Gamma \times \{\leftarrow, \to\}$ describing the rules the machine M uses in performing each step of M.
- an initial state $s_0 \in S$,
- a set of final states $E \subseteq S$.

There are two kinds of transition relations, the right translation $\delta(s, \sigma) = (s', \sigma', \to)$, which we write as

$$(v\rho, s, \sigma w) \to \begin{cases} (v\rho\sigma', s', w) & \text{if } w \neq \varepsilon, \\ (v\rho\sigma', s', \llcorner) & \text{if } w = \varepsilon, \end{cases} \tag{2.54}$$

and the left translation $\delta(s, \sigma) = (s', \sigma', \leftarrow)$, written as

$$(v\rho, s, \sigma w) \to \begin{cases} (v, s', \rho\sigma' w) & \text{if } v \neq \varepsilon, \\ (\llcorner, s', \rho\sigma') & \text{if } v = \varepsilon, \end{cases} \tag{2.55}$$

with the symbols $\rho, \sigma \in \Gamma$, the words $v, w \in \Gamma^+$, and the state $s \in S$. (Note the different meanings of the arrows "$\to$" as a controll symbol and as state transition sign.)

A *nondeterministic Turing machine M* is a seven-tuple $(S, \Sigma, \Gamma, \delta, s_0, \llcorner, E)$ consisting of the same elements as a determistic Turing machine, but with a transition relation

$$\delta : S \times \Gamma \to \mathscr{P}(S \times \Gamma \times \{\leftarrow, \to\}). \tag{2.56}$$

A Turing machine is called *linearly bounded* if only the symbols of the input word are changed, such that any output sequence cannot be longer than the input sequence. $\lhd$

Theorem 2.50. *Any language accepted by a deterministic finite automaton is regular, and for each regular language there exists a finite automaton accepting it.*

Proof. [10, §5.4] $\square$

Theorem 2.51. *Any language accepted by a pushdown automaton is context-free, and for each context-free language there exists a pushdown automaton accepting it.*

Proof. [10, §5.5.2] □

Theorem 2.52. *Any language accepted by a nondeterministic linearly bounded Turing machine is context-sensitive, and for each context-sensitive language there exists a nondeterministic lineraly bounded Turing machine accepting it.*

Proof. [10, §6.3] □

Theorem 2.53. *Any language accepted by a Turing machine is phrase-structured, and for each phrase-structure language there exists a Turing machine accepting it.*

Proof. [10, §6.3] □

Chapter 3

Grammar of DNA Strings

The molecules playing central roles in molecular biology and genetics are the so-called informational macromolecules DNA, RNA, and proteins. From the perspective of language theory, they exhibit a string behavior. One genetic phenomenon which has been studied from the view point of formal language theory is the splicing rule of DNA recombination. In the sequel this will be shown in more detail, based on the presentation in [8].

3.1 Searl's approach to DNA language

The following considerations of this section are based on [25] and [26]. The basis for the string representation of the molecular interactions both within and between nucleotides is the DNA alphabet

$$\Sigma_{\mathrm{DNA}} = \{\mathtt{A}, \mathtt{C}, \mathtt{G}, \mathtt{T}\}. \tag{3.1}$$

An important notion for molecular interactions is the complementarity of DNA words.

Definition 3.1. A *complementation* on a set M is a bijection $c : M \to M$ satisfying $c^2 = \mathrm{Id}_M$. ◁

Definition 3.2. Let the map $\bar{} : \Sigma_{\mathrm{DNA}} \cup \{\varepsilon\} \to \Sigma_{\mathrm{DNA}} \cup \{\varepsilon\}$ with $\bar{\varepsilon} = \varepsilon$, $\bar{A} = T$, $\bar{C} = G$, $\bar{G} = C$, $\bar{T} = A$ be given. For a word $w \in \Sigma^*_{\mathrm{DNA}}$ the complement $\bar{w}$ is then given by

$$\overline{w_1 w_2 \cdots w_n} = \bar{w}_n \cdots \bar{w}_2 \bar{w}_1 \tag{3.2}$$

with $w = w_1 w_2 \cdots w_n$, where $w_i \in \Sigma$. The map $\bar{} : \Sigma^*_{\mathrm{DNA}} \to \Sigma^*_{\mathrm{DNA}}$ thus defined is called *DNA-complementation* and $\bar{w}$ the *DNA-complement* of the word w. ◁

For a word $w \in \Sigma^*_{\mathrm{DNA}}$ of DNA, the complement $\bar{w}$ models the opposite helix strand of w. Complementary words can also be viewed as an operation corresponding to the process of DNA replication, since a new strand of DNA is laid down in the direction opposite to that of the template, taking the complement of each nucleobase.

Theorem 3.3 (Watson and Crick 1953). *Replication of the opposite strand of a string of DNA yields back the original string.*

Proof. Since DNA replication corresponds to DNA complementation, the theorem follows from $\bar{\bar{w}} = w$. □

Lemma 3.4 (Dyad symmetry). *For $w \in \Sigma^*_{\mathrm{DNA}}$ we have $w = \bar{w}$ if and only if $w = u\bar{u}$ for some $u \in \Sigma^*_{\mathrm{DNA}}$. In particular, w has even length.*

Proof. $\Leftarrow$: If $w = u\bar{u}$, we have $\bar{w} = \overline{u\bar{u}} = \bar{\bar{u}}\bar{u} = u\bar{u} = w$.

$\Rightarrow$: w must have even length, else it had a centermost letter $x \in \Sigma_{\mathrm{DNA}}$ satisfying $\bar{x} = x$, which is impossible by Definition 3.2. Thus we can divide any such w into equal halves u, v, i.e., $w = uv$ and $\bar{w} = \overline{uv} = \bar{v}\bar{u}$, hence $\bar{v} = u$. □

Lemma 3.5 (Copy property). *Let be $w \in \Sigma^*_{\mathrm{DNA}}$. If $w = \bar{w}$ then $w^n = \overline{w^n}$ for all $n \in \mathbb{N}$.*

Proof. Induction over n: $\overline{w^n} = \overline{ww^{n-1}} = \overline{w^{n-1}}\bar{w} = w^{n-1}w = w^n$. □

An important class of DNA strings are those connected to a *hairpin* molecule. This is a single strand which folds back on itself and base-pairs in the same fashion as two distinct strands of a double helix would. Hairpins form the language

$$L_{\mathrm{h}} = \{w \in \Sigma^*_{\mathrm{DNA}} \mid w = \bar{w}\}. \tag{3.3}$$

As a formal language, L_{h} resembles a palindrome language. In fact it can be generated by the context-free grammar $G_{\mathrm{h}} = (\{S\}, \Sigma_{\mathrm{DNA}}, P, S\}$ with the productions

$$S \to xS\bar{x} \mid \varepsilon \tag{3.4}$$

where $x \in \Sigma_{\mathrm{DNA}}$.

Example 3.6. The words $v = \mathtt{GATC}$ and $w = \mathtt{TGGCCA}$,

$$v = \boxed{\begin{array}{l} 5' \to \mathtt{GA} \to 3' \\ 3' \leftarrow \mathtt{CT} \leftarrow 5' \end{array}} \qquad w = \boxed{\begin{array}{l} 5' \to \mathtt{TGG} \to 3' \\ 3' \leftarrow \mathtt{GGT} \leftarrow 5' \end{array}} \tag{3.5}$$

are in L_{h} since the productions (3.4) yield $S \Rightarrow \mathtt{G}S\mathtt{C} \Rightarrow \mathtt{GA}S\mathtt{TC} \Rightarrow \mathtt{GATC}$ and $S \Rightarrow \mathtt{G}S\mathtt{C} \Rightarrow \mathtt{GG}S\mathtt{CC} \Rightarrow \mathtt{TGG}S\mathtt{CCA} \Rightarrow \mathtt{TGGCCA}$. By Lemma 3.5, the words $vv = \mathtt{GATCGATC}$ and $ww = \mathtt{TGGCCATGGCCA}$,

$$\boxed{\begin{array}{l} 5' \to \mathtt{GATC} \to 3' \\ 3' \leftarrow \mathtt{CTAG} \leftarrow 5' \end{array}} \qquad \boxed{\begin{array}{l} 5' \to \mathtt{TGGCCA} \to 3' \\ 3' \leftarrow \mathtt{ACCGGT} \leftarrow 5' \end{array}} \tag{3.6}$$

are then in L_{h}, too. ◊

Theorem 3.7. *L_{h} is context-free, but not regular.*

Proof. First, L_{h} is generated by the context-free grammar (3.4). However, by the pumping lemma 2.18 for regular languages it cannot be regular. □

Does Theorem 3.7 imply that the language of DNA is not regular? The answer of this question requires a precise definition of the "language of DNA." Usually [25, p 5] it is considered to be loosely specified by a series of important biological phenomena whose manifestations can be formalized linguistically.

The grammar G_{h} is idealized in several senses. First, base-pairs regions, or *stems*, need not be perfectly base-paired to form actual structures *in vivo*. In fact, in RNA those structures are more common, not only are occasional mismatches tolerated but certain other non-complementary pairings are found with an intermediate degree of preference. *"RNA folding is, overall, more a matter of thermodynamics than discrete mathematics."* [25, p 5]. Second, the hairpin structure in the exact form $w = \bar{w}$ of (3.3) is not realistic since steric hindrance restricts the radius of curvature at the turn of the hairpin so that at least three bases are unpaired. A more realistic grammar representing the hairpin structure should recognize the potential for an unpaired stretch of nucleobases at the turn, forming what is called a "stem-and-loop" structure

$$G_{\text{sl}}: \quad S \to xS\bar{x} \mid A, \quad A \to xA \mid \varepsilon. \qquad (x \in \Sigma_{\text{DNA}}) \tag{3.7}$$

Unfortunately, $L(G_{\text{sl}}) = \Sigma^*$, that is, the language generated by this grammar contains any possible word. G_{sl} thus has a weak generative capacity.

Example 3.8. *(Dumbbell language)* The words $v =$ `GATCGATC` and $w =$ `TGGCCATGGCCA` in (3.6) are in $L_{\text{db}} = \{vw \mid v, w \in L_{\text{h}}\}$, which is generated by the *dumbbell grammar*

$$S \to XX, \qquad X \to xX\bar{x} \mid \varepsilon. \qquad (x \in \Sigma_{\text{DNA}}) \tag{3.8}$$

E.g., $S \Rightarrow XX \Rightarrow$ `G`X`CG`X`C` $\Rightarrow$ `GA`X`TCGA`X`TC` $\Rightarrow$ `GATCGATC`. This language models adjacent stems. ◊

Example 3.9. *(Cloverleaf language)* The *cloverleaf language*

$$L_{\text{c}} = \{uv_1\bar{v}_1 \dots v_n\bar{v}_n\bar{u} \mid u, v_1, \dots, v_n \in \Sigma^*_{\text{DNA}},\ n \in \mathbb{N}\}, \tag{3.9}$$

is generated by the grammar

$$G_{\text{a}}: \quad S \to xS\bar{x} \mid X, \quad X \to XY \mid \varepsilon, \quad Y \to xY\bar{x} \mid \varepsilon. \tag{3.10}$$

$(x \in \Sigma_{\text{DNA}})$ Especially for $n = 3$, it typifies transfer RNA molecules. ◊

Example 3.10. *(Attenuator language)* The words $v =$ `GATCGATC` and $w =$ `TGGCCATGGCCA` in (3.6) are in the *attenuator language*

$$L_{\text{a}} = \{ww \mid w \in \Sigma^*_{\text{DNA}},\ w = \bar{w}\}, \tag{3.11}$$

◊

The following observations suggest that the DNA language is beyond context-free.

- Direct repeats are common in DNA, and copy languages such as $L_{\text{copy}} = \{ww \mid w \in \Sigma^*\}$ are not context-free.
- Crossing dependencies are observed in parallel interactions between strands, seen commonly in proteins.

- Pseudoknots are a form of secondary structure in which two stem-and-loop structures overlap, such that one of the loops contains half of the other stem. They can be represented by the pseudoknot language $L_\psi = \{uv\bar{u}\bar{v}|\ u, v \in \Sigma^*_{\text{DNA}}\}$.

- The common biological evolutionary rearrangements of a DNA string are duplication, inversion, transposition, and deletion

$$\text{DUP}(L) = \{xuuy|\ xuy \in L\} \tag{3.12}$$
$$\text{INV}(L) = \{x\bar{u}y|\ xuy \in L\} \tag{3.13}$$
$$\text{TRANS}(L) = \{xvuy|\ xuvy \in L\} \tag{3.14}$$
$$\text{DEL}(L) = \{xy|\ xuy \in L\} \tag{3.15}$$

  with $L \subset \Sigma_{\text{DNA}}$. Only the deletion lets a context-free language remain context-free. Under all other operations, a regular or context-free language is not closed. Evolution thus seems to tend to increasing linguistic complexity.

- The interaction of DNA macromolecules is fundamental to the workings of biological systems. By evolutionary selection, they thus have indirect affects on the formation of genetic strings. However, traditional formal linguistics capture dependencies *within* strings, but not *between* strings in a collection.

3.2 Gene regulation and inadequacy of context-free grammars

DNA molecules contain information that determines different regulatory networks responsible for differentiation and regulation of chemical activities. A central role is played by operons, functioning units of genomic DNA containing a cluster of genes under the control of a single regulatory signal, a promoter. It influences a certain region, called the operator, to which a regulatory protein (repressor or inducer) can bind and thereby can raise or lower the affinity for RNA polymerase. In this way, regulation of gene expression is thus executed by the promoters as molecular switches which can be on or off.

For a specific interaction, the three-dimensional protein structure and the DNA structure must fit into each other like lock and key. Let us call $X_1, \ldots X_n$ the sites of a regulatory proteins which bind to DNA operator sites $T_1, \ldots, T_n$, their respective targets. Then a change in the domain of the protein R_i to a different structure R'_i will impose a corresponding chance of the DNA operator site T_i to a different structure T'_i.

Example 3.11. [4] The *lac* repressor uses the amino acid `YQTVSRVV` for interacting with the *lac* operator `TGTGAGC`. If either the repressor is mutated to `VATVSRVV` or the operator is mutated to `TGTAAGC`, the recognition between these two structures is lost, but these specific repressor and operator mutants recognize each other. ◊

Let us name R_i a structural gene, i.e., a string of nucleotides, coding for a regulatory protein, and T_i the respective target recognized by this regulatory protein. Then a plausible subset of the "genetic language of regulation" can be formed, for $n \in \mathbb{N}$, by the string

$$(R_1, R_2, \ldots, R_n, T_1, T_2, \ldots, T_n) \in \Sigma^*_{\text{DNA}} \tag{3.16}$$

where each pair (R_i, T_i) for $1 \leqq i \leqq n$ has a dependency relation. Such dependency relations are equivalent to natural language expressions such as "John, Jenny and Mary are a doctor, a nurse and a philosopher, respectively." Such dependency relations cannot be handled by context-free grammars [4].

3.3 DNA splicing rule

To introduce DNA splicing let us look at a special example. Suppose an aquous solution with two DNA molecules, a ligase enzyme and the enzymes *Dpn*I and *Bal*I, two *restriction endonucleases*. Then a splicing and a recombination of the DNA molecules may happen in the following three steps. When *Dpn*I encounters a segment of a DNA molecule having the four letter word

$$\begin{array}{|c|}\hline \texttt{GATC} \\ \texttt{CTAG} \\ \hline \end{array} \tag{3.17}$$

of a DNA double strand (where the upper row denotes the 5′–3′ strand 5′–GATC–3′, and the lower one the 3′–5′ strand 3′–CTAG–5′), it cuts both strands of the DNA molecule between A and T. When *Bal*I encounters a segment of a DNA molecule having the six letter word

$$\begin{array}{|c|}\hline \texttt{TGGCCA} \\ \texttt{ACCGGT} \\ \hline \end{array} \tag{3.18}$$

of a DNA molecule, it analogously cuts it between G and C. The ligase enzyme then has the potential to bind together pairs of those of the cut molecules which have 5′ phosphates still attached, as will be the case where the cuts are fresh. For instance, if the two DNA molecules

$$\begin{array}{|c|}\hline \texttt{AAAAAAAGATCAAAAAAA} \\ \texttt{TTTTTTTCTAGTTTTTTT} \\ \hline \end{array} \qquad \begin{array}{|c|}\hline \texttt{CCCCCCCCCCCCTGGCCACCCCCCCCCC} \\ \texttt{GGGGGGGGGGGGACCGGTGGGGGGGGGG} \\ \hline \end{array}$$

are present in the aqueous solution, the action of the restriction endonucleases leave four DNA molecules in the solution:

$$\begin{array}{|c|}\hline \texttt{AAAAAAAGA} \\ \texttt{TTTTTTTCT} \\ \hline \end{array} \qquad \begin{array}{|c|}\hline \texttt{TCAAAAAAA} \\ \texttt{AGTTTTTTT} \\ \hline \end{array}$$

$$\begin{array}{|c|}\hline \texttt{CCCCCCCCCCCCTGG} \\ \texttt{GGGGGGGGGGGGACC} \\ \hline \end{array} \qquad \begin{array}{|c|}\hline \texttt{CCACCCCCCCCCC} \\ \texttt{GGTGGGGGGGGGG} \\ \hline \end{array}$$

The ligase enzyme then can combine the first and the fourth molecule producing the molecule

$$\begin{array}{|c|}\hline \texttt{AAAAAAAGACCACCCCCCCCC} \\ \texttt{TTTTTTTCTGGTGGGGGGGGG} \\ \hline \end{array}\,.$$

This recombination cannot be recut by the by either of the endonucleases. We can formalize the recombination as an operation on two words of a formal language L_{DNA} over the alphabet Σ_{DNA} given in (3.1). A word w in L_{DNA} is then given as the upper 5′–3′ strand in the above representation of a DNA molecule. For instance, the word GATC denotes the DNA segment (3.17).

Definition 3.12. Consider an alphabet Σ and two special symbols $\#, \$ \notin \Sigma$. Then a *splicing rule* r over Σ is a string

$$r = u_1 \# u_2 \$ u_3 \# u_4 \tag{3.19}$$

with $u_1, \ldots, u_4 \in \Sigma^*$. For a splicing rule r and words $x, y, z \in \Sigma^*$ we write

$$(x, y) \vdash_r z \tag{3.20}$$

if and only if $x = x_1 u_1 u_2 x_2$, $y = y_1 u_3 u_4 y_2$, and $z = x_1 u_1 u_4 y_2$, cf. Figure 3.1. We

Figure 3.1: The splicing operation (3.20).

then say that z is obtained by *splicing* x and y by the rule r, and $u_1 u_2$ and $u_3 u_4$ are called the *sites*. ◁

Theorem 3.13. *If the language of DNA sequences is not context-free, then it is even not context-sensitive.*

Proof. [8, Thm. 1], basing on closure properties of various language families under DNA splicing. □

Conjecture. *The language of DNA is not context-sensitive.*

This conjecture is not proved, at least to date (2011), but it is very probable to be true, for the following reason. The process of gene expression by RNA polymerase II needs promoters to regulate the transcription of the following gene, for instance the TATA box, and the termination by certain sequences such as AATAAA; hence by the evolutionary pressure only those DNA sequences should to have survived which respect this context-sensitive scheme. Thus the DNA grammar seems to be not context-free.

At first sight this conjecture seems to shatter all hopes to find a genome grammar, because a non-context-free grammar is far too complicated to be recognized easily. However, what actually is a complicated grammar? And what does it imply for the labguage it generates? What, in fact, does the Chomsky hierarchy tell about the difficulty to find a grammar for a given set of string sequences? At least, we must acknowledge that a simple *regular* grammar such as

$$S \to xS \mid \varepsilon \qquad (x \in \Sigma) \tag{3.21}$$

for some given alphabet Σ generates the most unstructured language

$$L_{\text{random}} = \Sigma^*, \tag{3.22}$$

consisting of random strings over Σ.

Appendix A

Mathematical Foundations

A.1 Notation

Mathematical and logical symbols

$:=$ is defined as
$\emptyset$ the empty set, $\emptyset = \{\}$
$\forall$ for all
$\exists$ there exists
$\exists_1$ there exists exactly one
$\Rightarrow$ implies, only if
$\Leftarrow$ follows from, if
$\Leftrightarrow$ if and only if

Number sets

$\mathbb{N}$ the natural numbers, $\mathbb{N} = \{1, 2, 3, \ldots\}$
$\mathbb{N}_0$ the natural numbers with zero, $\mathbb{N}_0 = \{0\} \cup \mathbb{N}$
$\mathbb{Z}$ the integers, $\mathbb{Z} = \{\ldots, -2, -1, 0, 1, 2, 3, \ldots\}$
$\mathbb{Z}^\times$ the nonvanishing integers, $\mathbb{Z}^\times = \mathbb{Z} \setminus \{0\}$
$\mathbb{Q}$ the rational numbers, $\mathbb{Q} = \{\frac{m}{n} |\ m \in \mathbb{Z},\ n \in \mathbb{N}\}$
$\mathbb{Q}^\times$ the nonvanishing rational numbers, $\mathbb{Q}^\times = \mathbb{Q} \setminus \{0\}$
$\mathbb{R}$ the real numbers, $\mathbb{R} = (-\infty, +\infty)$
$\mathbb{R}^\times$ the nonvanishing real numbers, $\mathbb{R}^\times = \mathbb{R} \setminus \{0\}$

A.2 Sets

The fundamental notion of mathematics is the set. Intuitive as at seems at first sight, this notion is rather subtle. It was a recognized in the beginning of the 20th century that a collection of objects need not be a set. A set is a mathematical concept which obeys strict axioms, the words "collection", "class", "family", or "system" of objects (which may also be sets) are often loosely used.

In mathematical logic, the notions "set" and "element x of a set S," denoted $x \in S$, are frequently used obeying the eight axioms of Zermelo-Fraenkel [5, §VII.3], [21, §3], [Ra, §2], [30, §4.4.3]:

(i) *Axiom of extensionality:* Two sets are equal if and only if they have the same elements.

(ii) *Axiom of separation:* For every set S and each property $\mathscr{A}$ there exists a set T of those elements of S with the property $\mathscr{A}$, written symbolically

$$T = \{x \in S \mid x \text{ has the property } \mathscr{A}\}. \tag{A.1}$$

(iii) *Axiom of pairing:* For every two sets S and T there exists the set $\{S, T\}$.

(iv) *Axiom of union:* For every set S the union $\bigcup S$ of sets in S is a set.

(v) *Axiom of power set:* For every set S there exists the power set $\mathscr{P}(S)$ of S, written $\mathscr{P}(S) = \{A \mid A \subset S\}$.

(vi) *Axiom of infinity:* There exists a set which contains the infinitely many sets $0 := \emptyset$, $1 := \{\emptyset\}$, $2 := \{\emptyset, \{\emptyset\}\}$, $3 := \{\emptyset, \{\emptyset, \{\emptyset\}\}\}$, … , or in other words, $1 = \{0\}$, $2 = \{0, 1\}$, and in general $n = \{0, 1, \ldots, n-1\}$.

(vii) *Axiom of replacement:* If the expression $\varphi(x, y, z_1, \ldots, z_n)$ for chosen parameters z_1, …, z_n defines an association $x \mapsto y = f(x)$, then the image $f(A)$ of a set is a set.

(viii) *Axiom of choice:* For a given finite or infinite index set I and every family $(S_i)_{i\in I}$ of nonempty sets S_i there exists a family $(x_i)_{i\in I}$ of elements $x_i \in S_i$ for every $i \in I$.

This system of axioms is also called *ZFC*, for Zermelo-Fraenkel with axiom of choice. The axiom of choice is the most famous of the ZFC axioms. There are a series of equivalent axioms, for instance Zorn's lemma or Zermelo's well-ordering theorem [21, §8.8], [29, p 5]. In contrast to the other axioms it does not tell constructively when general classes can be regarded as sets, but represents a mere existence postulate: It offers no recipe *how* an element of each set of a system of nonempty sets could be chosen.

Historically perhaps most important is Axiom (ii), the axiom of separation, which excludes Russell's antinomy. In 1901 Russel communicated this paradox to Frege, the leading mathematical logician of the period.[1] It shattered Cantor's (nowadays called naive) set theory. Since Frege at this time was going to publish his second edition of *Grundgesetze der Arithmetik*, which finally was released in 1903 and which heavily based on Cantor's set theory, he draw the bitter conclusion in the epilog:

> *Einem wissenschaftlichen Schriftsteller kann kaum etwas Unerwünschteres begegnen, als daß ihm nach Vollendung einer Arbeit eine der Grundlagen seines Baues erschüttert wird. In diese Lage wurde ich durch einen Brief des Herrn Bertrand Russell versetzt, als der Druck dieses Bandes sich seinem Ende näherte.*

[1] In 1879, Gottlob Frege (1848 – 1925) published his *Begriffsschrift, eine der arithmetischen nachgebildete Formelsprache des reinen Denkens* ("concept notation, a formula language, modelled on that of arithmetic, of pure thought"). The Begriffsschrift was arguably the most important publication in logic since Aristotle founded the subject more than 2000 years before.

Zermelo, who had discovered the paradox even a year before Russell but did not publish it, solved the problem in 1908 by introducing a set theory basing on the axiom of separation.

Theorem A.1 (**Exclusion of Russell's antinomy**). *There exists no set of all sets. (Or equivalently: The class of all sets is not a set.)*

Proof. Assume there exists a set M of all sets. By the axiom of separation, Axiom (ii), the class $R = \{x \in M \mid x \notin x\}$ then is a set. Hence there are two possibilities: (i) If $R \notin R$, then $R \in R$, by construction of R. (ii) If $R \in R$, then $R \notin R$ by construction of R. Hence we get the contradiction $R \in R \iff R \notin R$, i.e., the assumption must be wrong. □

Theorem A.2. *Given two sets A and B, for $x \in A$ and $y \in B$ the "Kuratowski pair" $(x, y) := \{\{x\}, \{x, y\}\}$, also called the "ordered pair" (x, y), satisfies*

$$(x, y) = (x', y') \iff x = x' \text{ and } y = y', \tag{A.2}$$

and the class

$$A \times B := \{(x, y) \mid x \in A \text{ and } y \in B\} \tag{A.3}$$

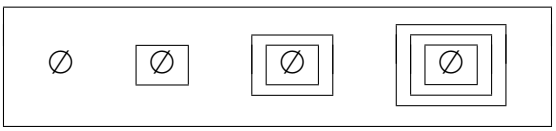
is a set, called the Cartesian product *of A and B.*

Proof. The property (A.2) of ordered pairs follows directly from the ZFC axioms (i) and (iii), the second assertion follows from axioms (ii) and (v). For details see [21, §4.3]. □

Why do we need the axiom of infinity? For instance, the set

$$4 = \{\emptyset, \{\emptyset\}, \{\{\emptyset\}\}, \{\{\{\emptyset\}\}\}\}$$

can be represented as boxes in boxes:



Each element of the set leaves us in a situation resembling the birthday child who receives the "disappointing gift" [21, p 180], namely a huge box which is opened to exhibit another box, which can be opened again to show another box in which is another box etc., to eventually present — an empty set.

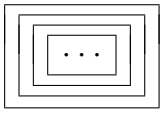
Up to the end of the 19th century it was commonplace belief among philosophers and mathematicians that the existence of infinite sets could be proved. In particular, one thought that the set of natural numbers could be constructed by logic. We know better now. *"Logic can codify the valid forms of reasoning but it cannot prove the existence of anything"* [21, p 27]. Zermelo was the first to recognize that the axiom of infinity has to be postulated to construct the set of natural numbers.

The axiom system of Zermelo-Fraenkel ZFC works well, that is, virtually all results in current mathematics can be derived by it.[2] However, there remains an uncomfortable property of the system to allow the set

$$\Omega = \{\Omega\}. \tag{A.4}$$

[2] It is a still open question, though, whether ZFC is logically consistent at all [5, pp 120].

We can think of Ω as the "ultimately frustrating gift" [21, p 181] since denoting Ω by a box $\square$, each box has exactly one box inside, identical with the one just opened, and you can keep opening the boxes without ever finding *anything* …



Not even an empty set. If such phenomena are desired to be banned, it is common to use the following additional axiom proposed by von Neumann in 1925:

(ix) *Axiom of Regularity:* Every non-empty set S contains an element x which is disjoint from S.

ZFC with the axiom of regularity thus forbids the set $\Omega = \{\Omega\}$, and hence in set theories based on this extension there exists no ultimately frustrating gift.

A.3 Maps

Definition A.3. Let A, B be two sets. A *map*, or a *function*, from A to B, written

$$f : A \to B, \tag{A.5}$$

denotes an association of any element $a \in A$ to an element $b = f(a)$ in B, also written $a \mapsto b$. Then A is called the *domain* of f, and B the *codomain* or *target* of f. For any subset $A' \subset A$ of A, the set

$$f(A') := \{f(a) |\, a \in A'\} \subset B \tag{A.6}$$

is called the *image* of A' under f. In particular, $f(A)$ is shortly called the image of f. For a subset $B' \subset B$ of B, the set

$$f^{-1}(B') := \{a \in A |\, f(a) \in B'\} \subset A \tag{A.7}$$

is called the *preimage* of B' under f. In particular, $f^{-1}(B)$ is shortly called the preimage of f. The map $f : A \to B$ is called …

- *injective* if $f(a) = f(a')$ implies $a = a'$ for all a, $a' \in A$, or equivalently, $|f^{-1}(\{b\})| = 1$ for all $b \in f(A)$;
- *surjective* if $f(A) = B$;
- *bijective* if it is both injective and surjective.

Two maps f: $A \to B$ and g: $C \to D$ are *equal*, if $A = C$, $B = D$, and $f(x) = g(x)$ for all $x \in A$. If for two maps A and B there exists a bijective map, we say that A and B have the same *cardinality* and denote $A \sim B$. ◁

Remark A.4. In set theory, a function is considered as a special binary relation. A *binary relation* R on two sets A and B is a subset $R \subset A \times B$. For instance, let be A the set of parents and B the set of children, given by

$$A = \{\text{Homer, Marge}\}, \qquad B = \{\text{Bart, Lisa, Maggie}\}.$$

Then the "Simpsons relation" $R =$ "is mother of" on A and B implies

$$(\text{Marge, Bart}) \in R, \qquad \text{but} \qquad (\text{Homer, Lisa}) \notin R.$$

A special class of binary relations are those $f \subset A \times B$ assigning to each element of A some element of B, formally

$$\forall x \in A \;\; \exists y \in B \text{ such that } (x, y) \in R. \tag{A.8}$$

Such a relation is called a *multivalued function*, written $f\colon A \to \mathscr{P}(B)$. Then a function $f \subset A \times B$ is a special relation assigning to each element of A *exactly one* element of B [21, §4.16],

$$\forall x \in A \;\; \exists_1 y \in B \text{ such that } (x, y) \in R. \tag{A.9}$$

Note that the above defined "Simpsons relation" on A and B is neither a multivalued function nor a function. ◊

Remark A.5. From the view point of algorithmics, the *computation* of a map is important. In set theory the functions $f, g : \mathbb{R}^2 \to \mathbb{R}$,

$$f(x, y) = (x + y)^2 \qquad g(x, y) = x^2 + 2xy + y^2 \tag{A.10}$$

are equal, but their computation is rather different. Algorithmically, f is more efficient than g (since there are only two arithmetical operations to compute f, but six for g). Whether the intuitive notion of "function as a computation" can be represented faithfully in set theory is an unsolved problem [21, p 41]. ◊

Example A.6. *(Bijection between* $\mathbb{N}$ *and* $\mathbb{Z}$*)* Define the function $f : \mathbb{N} \to \mathbb{Z}$,

$$f(n) = (-1)^n \, \frac{2n - 1}{4} \; + \; \frac{1}{4}. \tag{A.11}$$

Then for all $n \in \mathbb{N}$ we have $f(2n) = n$ and $f(2n - 1) = 1 - n$. In particular, this implies immediately that $f(m) \neq f(n)$ if $m \neq n$ for $m, n \in \mathbb{N}$, i.e., f is injective. Moreover, f is surjective since for any $y \in \mathbb{Z}$ there exists an element $x \in \mathbb{N}$ such that $f(x) = y$, namely $x = 2y$ if $y > 0$ and $x = 1 - 2y$ if $y \leqq 0$. Thus, f is a bijection and $\mathbb{N} \sim \mathbb{Z}$, cf. [7, p 305]. ◊

Remark A.7. In a proof that two general sets A and B have the same cardinality, it is usually a tedious task to construct a surjective map. By the Cantor-Bernstein Theorem, often also called Schröder-Bernstein Theorem, it suffices to find two injective maps $f : A \to B$ and $g : B \to A$ [7, §3.5.1]. ◊

Theorem A.8 (Pigeonhole Principle). *Every map $f : A \to A$ on a finite set A into itself is injective if and only if it is surjective.*

Proof. The "only if" part ("f injective $\Rightarrow f(A) = A$") is shown in [21, 5.25]. Let $I_n = \{0, 1, \ldots, n-1\}$. The proof bases on the property (A.14) of a map $g : I_n \to I_n$, since A is finite, there exists a bijective permutation $\pi : A \to I_n$, and we can define $g : I_n \to I_n$ by the equation

$$g(i) = \pi(f(\pi^{-1}(i))) \tag{A.12}$$

for $i < n$ so that

$$f(x) = \pi^{-1}(g(\pi(x))) \tag{A.13}$$

for all $x \in A$. Now if f is injective then also g is injective, as a composition of injections (A.12); but then by (A.14) g is bijective, and hence f as a compositions of bijections is bijective. By the same reasoning, if f is surjective, g is surjective as a compositions of surjections, i.e. injective by (A.14), and thus f is injective as a composition of injections. □

Lemma A.9. *Let $I_n = \{0, 1, \ldots, n-1\}$. For all maps $g : I_n \to I_n$ and an arbitrary natural number n we have*

$$g(x) \neq g(y) \quad \forall x \neq y \qquad \Longleftrightarrow \qquad g(I_n) = I_n, \tag{A.14}$$

with $x \in y \in I_n$.

Proof. The proof is by induction over n. For $n = 0$ we have only one map $g : \emptyset \to \emptyset$, and for $n = 1$ only one function $g : \{1\} \to \{1\}$, and both are bijective. For the induction step $n \to n+1$ we assume (A.14) for *all* maps $h : I_n \to I_n$. Since $I_{n+1} = I_n \cup \{n\}$, for any map $g : I_{n+1} \to I_{n+1}$ and its restriction $h := g|_{I_n}$, defined by $h(k) = g(k)$ for $0 \leqq k < n$, we have one of the following three cases.

Case (1): $g(n) = n$. Then g is clearly injective if and only if h is injective, and g is surjective if and only if h is surjective.

Case (2): $n \notin g(I_{n+1})$. This means that g is not surjective, and since $g(n) = h(k) = g(k)$ for some k, i.e., g then is also not injective. On the other hand, the assumption that g is injective implies that $\nexists k \in [0, k)$ with $h(k) = g(n) \in [0, n)$, i.e., h is not injective, hence g is not injective, which is a contradiction to the assumption.

Case (3): There exist numbers $u, v < n$ such that $g(u) = n$, $g(n) = v$. Defining $h' : I_n \to I_n$,

$$h'(k) = \begin{cases} h(k) & \text{if } k < n \text{ and } k \neq u, \\ v & \text{if } k = u. \end{cases}, \tag{A.15}$$

i.e., a function h' agreeing with g at all arguments except u, we see that h' is injective if and only if g is injective, and that h' is surjective if and only if g is surjective. □

Remark A.10. Theorem A.8 is equivalently stated as [6, p 130]: *If n pigeons are put into n pigeonholes, then there is an empty pigeonhole if and only if there is a hole with more than one pigeon.* Here an empty pigeonhole means "not surjective," and a hole with more pigeons means "not injective." It implies the pigeonhole principle in its usual formulation refering to Dirichlet's *Schubfachprinzip* ("box principle") [7, §2.4]: *If more than n objects are distributed over n containers, then some container must contain more than one object.* Here "contain more than one object" means "is mapped injectively." ♢

Remark A.11. *(Hilbert's Hotel)* Theorem A.8 cannot be generalized for an infinite set A. For instance, the map $f : \mathbb{N} \to \mathbb{N}$,

$$f(n) = n + 1 \tag{A.16}$$

is injective, but not surjective. The fact that the pigeonhole principle is not valid for infinite sets is the reason for the possibility of Hilbert's Hotel, a hotel with infinitely many rooms completely numbered by 1, 2, …, all of which are occupied by a guest. However, if a new guest arrives and wishes to be accommodated in the hotel, each hotel guest can be moved from his room number n to the next room with number $n + 1$. Thus it is possible to make room for any finite number of new guests. ♢

A.4 Algebra

A main subject of abstract algebra is group theory. It studies possible binary operations on elements of a set, often called *structures*, which in the end generalize the addition and multiplication operations of numbers. This section gives a brief overview to the basic algebraic notions needed in our considerations here. For more details see any standard textbook on algebra, e.g., [16] or [17]. However, differing from most standard treatments the emphasis here is laid on properties of semigroups and monoids which, in the theory of formal languages, play a more important role than groups.

A.4.1 Semigroups, monoids, and groups

Definition A.12. A *semigroup* $(H, \circ)$ is a nonempty set H with the binary operation $\circ : H \times H \to H$ obeying the *law of associativity*

$$(x \circ y) \circ z = x \circ (y \circ z) \tag{A.17}$$

for all $x, y, z \in H$. If all elements $x, y \in H$ satisfy the *law of commutativity*

$$x \circ y = y \circ x, \tag{A.18}$$

then H is called *Abelian* or *commutative*. The number $|H|$ of elements of H is called the *order* of the semigroup. If it is clear from the context, one often speaks of the semigroup H instead of $(H, \circ)$. ◁

Definition A.13. A semigroup $(M, \circ)$ is called a *monoid* if it contains an element $e \in M$ satisfying

$$e \circ x = x \circ e = x \tag{A.19}$$

for all $x \in M$. In this case, e is called *neutral element*. ◁

Lemma A.14. *The neutral element in a monoid is unique.*

Proof. Assume that $e' \in M$ is another neutral element in the monoid. Then by (A.19) we have $e' \circ e = e'$ (since e is neutral), as well as $e' \circ e = e$ (since e' is neutral), i.e., $e' = e$. □

Examples A.15. *(Natural numbers)* The pair $(\mathbb{N}, +)$ is a semigroup but not a monoid, but $(\mathbb{N}_0, +)$ is a monoid with the neutral element 0. Moreover, $(\mathbb{N}, \cdot)$ is a monoid with the neutral element 1. ♢

Examples A.16. *(Maps)* Let X be a set. Then the set of all maps $X \to X$ is a monoid with respect to composition of maps, with the identity map as neutral element. ♢

Examples A.17. *(Words)* Let Σ be an alphabet. Then the set Σ^+ of all words over Σ, i.e., all strings of letters from Σ, is a semi-group with respect to concatenation [3, §6.1]. The set Σ^* containing the empty string ε is a monoid with the empty string as neutral element. ♢

Definition A.18. A *group* $(G, \circ)$ is a nonempty set G with the binary operation $\circ : G \times G \to G$ obeying the three group axioms, namely: the law of associativity

$$(x \circ y) \circ z = x \circ (y \circ z) \tag{A.20}$$

for all $x, y, z \in G$; there exists an element $e \in G$ satisfying

$$e \circ x = x, \tag{A.21}$$

the *neutral element*; for each $x \in G$ there exists an element $x^{-1} \in G$ such that

$$x^{-1} \circ x = e. \tag{A.22}$$

It is called the *inverse* of x (with respect to the operation $\circ$). If all elements $x, y \in G$ satisfy the *law of commutativity*

$$x \circ y = y \circ x, \tag{A.23}$$

then G is called *Abelian* or *commutative*. ◁

Lemma A.19. *Let be G a group and $x \in G$ with the neutral element $e \in G$. Then we have*

$$x \circ e = x, \tag{A.24}$$
$$x \circ x^{-1} = e \tag{A.25}$$

for all $x \in G$. In particular, a group is a monoid. Moreover, for each $x \in G$ the inverse $x^{-1} \in G$ is unique.

Proof. For $x \in G$ there exists an element $y = x^{-1} \in G$ with $y \circ x = e$, and an element $z = y^{-1} \in G$ with $z \circ y = e$. Thus

$$x = e \circ x = (z \circ y) \circ x = z \circ (y \circ x) = z \circ e.$$

Substituting e by $(e \circ e)$ this gives $x = z \circ (e \circ e) = (z \circ e) \circ e = x \circ e$, i.e., (A.24). In addition, $z = z \circ e = x$ and therefore $x \circ y = z \circ y = e$, i.e., (A.25). If $y' \in G$ is another element being an inverse of x, then we have $y' = y' \circ e = y' \circ (x \circ y) = (y' \circ x) \circ y = e \circ y = y$. □

Examples A.20. *(Numbers)* The monoids $(\mathbb{N}_0, +)$ and $(\mathbb{N}, \cdot)$ of Example A.15 are not groups. However, $(\mathbb{Z}, +)$ and $(\mathbb{Q}^\times, \cdot)$ with $\mathbb{Q}^\times = \mathbb{Q} \setminus \{0\}$ are commutative groups, as well as $(\mathbb{R}, +)$ and $(\mathbb{R}^\times, \cdot)$ with $\mathbb{R}^\times = \mathbb{R} \setminus \{0\}$. $(\mathbb{R}, \cdot)$ is an Abelian monoid but not a group, $(\mathbb{R}^\times, +)$ is not even a semigroup, ◊

Examples A.21. *(Matrices)* For $n \in \mathbb{N}$, let denote $M_n(\mathbb{K})$ the set of all $(n \times n)$ matrices over a field $\mathbb{K}$, for instance $\mathbb{Q}$, $\mathbb{R}$, or $\mathbb{C}$. Then $(M_n(\mathbb{K}), \cdot)$ with the matrix multiplication $\cdot$ and the identity matrix as neutral element. is a monoid, but not a group. However, $(M_n(\mathbb{K}), +)$ with the matrix addition is a commutative group, and the set $GL(n, \mathbb{K})$ of invertible matrices in $M_n(\mathbb{K})$ is a non-commutative group $(GL(n, \mathbb{K}, \cdot)$ with the identity matrix as neutral element. $GL(n, \mathbb{K})$ is called the *general linear group* over $\mathbb{K}$. ◊

Examples A.22. *(Unit fractions)* Let $F = \{1, \frac{1}{2}, \frac{1}{3}, \ldots\}$ denote the set of unit fractions $\frac{1}{n}$ with $n \in \mathbb{N}$. Then $(F, \cdot)$ and $(F \cup \{0\}, \cdot)$ are monoids, each with neutral element 1. ◊

Remark A.23. There are two usual ways to write the group operation, the additive notation and the multiplicative notation. In the additive notation we write $x + y$ instead of $x \circ y$, 0 is the neutral element, and the inverse of x is written as $-x$. It usually denotes commutative operations. In the multiplicative notation, xy is written instead of $x \circ y$, 1 is the neutral element, and x^{-1} is the inverse of x. ◊

Definition A.24. Let G be a semigroup. Then a subsemigroup $H \subset G$ being in G is called *normal* if

$$xH = Hx \qquad \text{for all } x \in G, \tag{A.26}$$

where $xH = \{xh |\ h \in H\}$ and $Hx = \{hx |\ h \in H\}$. Analogously, a submonoid H of a monoid G is called *normal* if (A.26) holds literally, and a subgroup H of a group G is called *normal* if (A.26) holds. ◁

Theorem A.25. *Is H a subgroup of a group G, then the following assertions are equivalent:*

(i) $xH = Hx$ *for all* $x \in G$*;*

(ii) $x^{-1}Hx = H$ *for all* $x \in G$*;*

(iii) $x^{-1}hx \in H$ *for all* $x \in G$ *and* $h \in H$*.*

Proof. The implications (i) ⇒ (ii) ⇒ (iii) are clear. To show (iii) ⇒ (i), let $x \in G$ and $h \in H$ be given; then $xh = (x^{-1})^{-1}hx^{-1}x \in Hx$ and $hx = x(x^{-1}hx) \in xH$. □

Theorem A.26. *All subgroups of an Abelian group are normal.*

Example A.27. *(Matrices)* Let $\mathbb{K}$ be $\mathbb{Q}$, $\mathbb{R}$, or $\mathbb{C}$ (or an arbitrary field), and denote the *special linear group* $SL(n, \mathbb{K})$ the set of all $(n \times n)$ matrices over $\mathbb{K}$ with determinant 1. Then $SL(n, \mathbb{K})$ is a normal subgroup of the general linear group $GL(n, \mathbb{K})$ from Example A.21. ◊

A.4.2 Homomorphisms

Structure-preserving maps between algebraic structures are called homomorphisms and play an important role in algebra.

Definition A.28. Let G and H be two semigroups. Then a map $f : G \to H$ is called a *homomorphism* if

$$f(xy) = f(x)\,f(y) \tag{A.27}$$

for all $x, y \in G$. The set of all homomorphisms from G to H is denoted by $\mathrm{Hom}(G, H)$. The set

$$\mathrm{im}f := f(G) := \{f(x) |\, x \in G\} \tag{A.28}$$

is called the *image* of f. If G and H are monoids and $e' \in H$ is the neutral element in H, the set

$$\ker f := f^{-1}(\{e'\}) := \{x \in G |\, f(x) = e'\} \tag{A.29}$$

is called the *kernel* of f. ◁

Lemma A.29. *If $f \in \mathrm{Hom}(G, H)$ for two monoids G and H with neutral elements $e \in G$ and $e' \in H$, then $f(e) = e'$.*

Proof. We have $f(e) = f(ee) = f(e)f(e)$, i.e., $f(e) = e'$. □

Theorem A.30. *If $f \in \mathrm{Hom}(G, H)$ for two groups G and H, then*

$$f(x^n) = f(x)^n \tag{A.30}$$

for all $n \in \mathbb{Z}$. If G and H are monoids, the equation holds for all $n \in \mathbb{N}_0$, and if they are semigroups, it holds for all $n \in \mathbb{N}$.

Proof. For $n \in \mathbb{N}$ Equation (A.30) follows by induction from (A.27). In case of monoids, $x^0 = e$ and $f(x^0) = f(x)^0 = e'$ by Lemma A.29. Finally, if G and H are groups, $e' = f(e) = f(xx^{-1}) = f(x)f(x^{-1})$, hence $f(x)^{-1} = f(x^{-1})$. □

Corollary A.31. *Let $f \in \mathrm{Hom}(G, H)$, with G and H specified as follows.*

(i) If G and H are semigroups then $\mathrm{im}f$ is a subsemigroup in H.

(ii) If G and H are monoids or groups then $\mathrm{im}f$ is a submonoid of H, $\ker f$ is a normal submonoid of G. $f(\ker f y) = f(yx)$ if x

(iii) If G and H are groups then $\mathrm{im}f$ is a subgroup of H and $\ker f$ is a normal subgroup of G.

Proof. The assertions for the image set $\mathrm{im}f$ follow directly from Equation (A.27) and Theorem A.29. In case (ii) and (iii) we have for any $y \in G$, $x \in K := \ker f$ and the neutral element $e' \in H$ that $f(xy) = f(x)f(y) = e'f(y) = f(y)e' = f(y)f(x) = f(yx)$, i.e., $f(Ky) = f(yK)$, and hence $Ky = f^{-1}(f(Ky)) = f^{-1}(f(yK)) = yK$. □

An important special class of homomorphisms of a semigroup G are are those which are bijective, and in particular those mapping G bijectively into G itself.

Definition A.32. For semigroups G and H, a bijective homomorphisms from G to H is called an *isomorphism*. The set of isomorphisms from G to G is given by

$$\mathrm{Aut}(G) := \mathrm{Hom}(G, G) \tag{A.31}$$

and is called the set of *automorphisms* of G. ◁

Lemma A.33. *For a semigroup G and the map composition $\circ$: $\mathrm{Aut}(G) \times \mathrm{Aut}(G) \to \mathrm{Aut}(G)$, the pair $(\mathrm{Aut}(G), \circ)$ is a group, with the identity map Id_G as neutral element.*

Proof. By definition of the composition $\circ$ the law of associativity (A.20) is satisfied. For any $f \in \mathrm{Aut}(G)$ and $x \in G$ we have $(\mathrm{Id}_G \circ f)(x) = \mathrm{Id}_G(f(x))) = f(x)$, i.e., $\mathrm{Id}_G \circ f = f$ as required by (A.21). Let $f \in \mathrm{Aut}(G)$ and $x, y \in G$ such that $y = f(x)$. Since $\mathrm{Aut}(G)$ consists of *all* homomorphisms from G to G, there exists one that maps y to x, i.e., $f^{-1} \in \mathrm{Aut}(G)$, satisfying (A.22). □

Definition A.34. Let G be a group and $g \in G$ an invertible element. Then the map

$$C_g : G \to G, \qquad x \mapsto gxg^{-1} \tag{A.32}$$

is called the *conjugation* with g. ◁

Lemma A.35. *For a semigroup G and an invertible element $g \in G$, the conjugation C_g with g is an automorphism, i.e., $C_g \in \mathrm{Aut}(G)$.*

Proof. For $x, y \in G$ we have $C_g(xy) = gxyg^1 = (gxg^1)(gyg^1) = C_g(x)\, C_g(y)$, and $C_e = \mathrm{Id}_G$ for the neutral element $e \in G$. Hence C_g is a homomorphism. Since for two elements $x, y \in G$ the equality $gxg^{-1} = gyg^{-1}$ implies $x = y$ after multiplying with g^{-1} from the left and with g from the right, C_g is injective. Since then the order of G is the same then the order of $C_g(G)$, i.e., $|C_g(G)| = |G|$, C_g is also surjective. □

Theorem A.36. *There are only five non-isomorphic semigroups of order two, given by the following multiplication tables ("Cayley tables"):*

O_2			LO_2			RO_2			$(\{0,1\}, \wedge)$			$\mathbb{Z}_2$		
$\cdot_0$	0	1	$\cdot_l$	0	1	$\cdot_r$	0	1	$\wedge$	0	1	$\oplus$	0	1
0	0	0	0	0	0	0	0	1	0	0	0	0	0	1
1	0	0	1	1	1	1	0	1	1	0	1	1	1	0
semigroups									*monoid*			*group*		

(A.33)

Proof. A complete enumeration of all $2^{2^2} = 16$ possible binary operations on the set {0, 1}, i.e., all combinations of filling the Cayley table

$\circ$	0	1
0	a_{00}	a_{01}
1	a_{10}	a_{11}

with $a_{ij} \in \{0,1\}$. Regarding isomorphims yields the assertion.[3] O_2, LO_2 and RO_2 each are no monoids since they do not have a neutral element. Instead, in O_2 the element 0 is an "absorbing" element, in LO_2 both 0 and 1 are left-absorbing, and in RO_2 both are right-absorbing. The semigroup $(\{0,1\},\wedge)$ is a monoid with the neutral element 1, and $\mathbb{Z}_2 = (\{0,1\},\oplus)$ is a group with neutral element 0 and 1 being its own inverse. □

Remark A.37. The semigroup O_2 is called the *null semigroup*, LO_2 the *left zero semigroup*, and RO_2 the *right zero semigroup*. RO_2 and LO_2 are not isomorphic but antiisomorphic, and hence "equivalent." Moreover, $(\{0,1\},\wedge)$ is isomorphic to $(\{0,1\},\vee)$, but here the neutral element is 0. An isomorphic matrix representation of $(\{0,1\},\wedge)$ is $(\{I,A\},\cdot)$ with

$$I = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}, \qquad A = \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix}, \tag{A.34}$$

and the operation $\cdot$ being the usual matrix multiplication. In $\mathbb{Z}_2 = (\{0,1\},\oplus)$ the binary operation $\oplus$ is a logical XOR. ♢

Remark A.38. In general there are n^{n^2} possible binary operations on a set with n elements. Under *oeis.org/A027851* the On-Line Encyclopedia of Integer Sequences, lists the number of nonisomorphic semigroups with n elements, and under *oeis.org/A001423* the number of nonequivalent semigroups. ♢

[3] *http://en.wikipedia.org/wiki/Semigroup_with_two_elements* [2011-05-17]

Bibliography

[1] S. Arora and B. Barak. *Computational Complexity. A Modern Approach*. Cambridge University Press, Cambridge, 2009.

[2] J. M. Castaño. 'Global index grammars and descriptive power'. In R. T. Oehrle and J. Rogers, editors, *Proceedings of Mathematics of Languages 8*, pages 1–12. molweb.org, 2003. *http://molweb.org/mol8/papers_mol8/castano.pdf*.

[3] A. M. Cohen, H. Cuypers, and H. Sterk. *Algebra Interactive!* Springer-Verlag, Berlin Heidelberg, 1999.

[4] J. Collado-Vides. 'The search for a grammatical theory of gene regulation is formally justified by showing the inadequacy of context-free grammars'. *Computer Applications in the Biosciences*, 7(3):321–326, 1991. doi: 10.1093/bioinformatics/7.3.321.

[5] H.-D. Ebbinghaus, J. Flum, and W. Thomas. *Einführung in die mathematische Logik*. Spektrum Akademischer Verlag, Heidelberg Berlin, 1996.

[6] R. L. Graham, D. E. Knuth, and O. Patashnik. *Concrete Mathematics*. Addison-Wesley, Upper Saddle River, NJ, 2nd edition, 1994.

[7] J. M. Harris, J. L. Hirst, and M. J. Mossinghoff. *Combinatorics and Graph Theory*. Springer, New York, 2nd edition, 2008.

[8] T. Head, G. Păun, and D. Pixton. 'Language Theory and Molecular Genetics: Generative Mechanisms Suggested by DNA Recombination'. In G. Rozenberg and A. Salomaa, editors, *Handbook of Formal Languages. Vol.2. Linear Modeling: Background and Application*, pages 295–360, Berlin Heidelberg, 2010. Springer-Verlag.

[9] S. Hedman. *A First Course in Logic. An Introduction to Model Theory, Proof Theory, Computability, and Complexity*. Oxford University Press, Oxford New York, 2004.

[10] D. W. Hoffmann. *Theoretische Informatik*. Carl Hanser Verlag, München, 2009.

[11] J. E. Hopcroft, R. Motwani, and J. D. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Pearson, Boston, 2007.

[12] J. E. Hopcroft and J. D. Ullman. *Einführung in die Automatentheorie, Formale Sprachen und Komplexitätstheorie*. Addison-Wesley, Bonn, 1988.

[13] J. E. Hopcroft, J. D. Ullman, and R. Motwani. *Einführung in die Automatentheorie, Formale Sprachen und Komplexitätstheorie*. Pearson Studium, München, 2003.

[14] S. Hovmöller, T. Zhou, and T. Ohlson. 'Conformations of amino acids in proteins'. *Acta Cryst.*, D58(Pt 5):768–776, 2002.

[15] IUPAC. *Compendium of Chemical Terminology*. Blackwell Scientific Publications, Oxford, 2nd edition, 1997. *doi: 10.1351/goldbook*.

[16] C. Jantzen and J. Schwermer. *Algebra*. Springer-Verlag, Berlin Heidelberg, 2006.

[17] O. Körner. *Algebra*. Akademische Verlagsgesellschaft, Frankfurt, 1974.

[18] S. C. Lovell, I. W. Davis, W. B. Arendall III, P. I. W. de Bakker, J. M. Word, M. G. Prisant, J. S. Richardson, and D. C. Richardson. 'Structure validation by Cα geometry: φ, ψ and Cβ deviation'. *Proteins 50 (3): 437–450*, 50(3):437–450, 2003. doi: 10.1002/prot.10286.

[19] R. Merkl and S. Waack. *Bioninformatik interaktiv. Grundlagen, Algorithmen, Anwendungen*. Wiley VCH, Weinheim, 2. edition, 2009.

[20] C. E. Mortimer and U. Müller. *Chemie*. Thieme, Stuttgart, 10. edition, 2010.

[21] Y. N. Moschovakis. *Notes on Set Theory*. Springer-Verlag, New York, 1994.

[22] P. Nelson. *Biological Physics. Energy, Information, Life*. W. H. Freeman, New York, 2004.

[23] D. S. T. Nicholl. *An Introduction to Genetic Engineering*. Cambridge University Press, Cambridge, 3rd edition, 2008.

[24] E. Sackmann and R. Merkel. *Lehrbuch der Biophysik*. Wiley-VCH, Weinheim, 2010.

[25] D. B. Searls. 'Formal Language Theory and Biological Macromolecules'. In *Series in Discrete Mathematics and Theoretical Computer Science*, pages 117–140, 1999. *http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.42.3056*.

[26] D. B. Searls. 'The language of genes'. *Nature*, 420(6912):211–217, 11 2002. DOI 10.1038/nature01255.

[27] M. Sipser. *Introduction to the Theory of Computation*. Thomson Course Technology, Boston, 2006.

[28] P. Turner, A. McLennan, A. Bates, and M. White. *Molecular Biology*. Taylor & Francis, New York, 3rd edition, 2005.

[29] B. von Querenburg. *Mengentheoretische Topologie*. Springer-Verlag, Berlin Heidelberg, 3. edition, 2001.

[30] E. Zeidler, editor. *Teubner Taschenbuch der Mathematik. Teil 1*, Leipzig, 1996. B. G. Teubner.

Sources

[GM] *http://www.informatics.sussex.ac.uk/research/groups/nlp/gazdar/nlp-in-lisp/* – Gerald Gazdar, Chris Mellish (1996): "Natural Language Processing in Lisp." [last access 2011-04-26]

[Mo] *http://www.staff.ncl.ac.uk/hermann.moisl/ell236/* – Hermann Moisl: "Computational Linguistics." [last access 2011-04-06]

[Ra] *http://page.mi.fu-berlin.de/raut/Mengenlehre/m.pdf* – Wolfgang Rautenberg (2008): "Grundkurs Menegnlehre." Lecture notes of Freie Universität Berlin [last access 2011-05-15]

Index

Title page graphics adapted from *http://upload.wikimedia.org/wikipedia/commons/b/b1/A-DNA%2C_B-DNA_and_Z-DNA.png*