DNA Computing Models

Zoya Ignatova · Israel Martínez-Pérez · Karl-Heinz Zimmermann

DNA Computing Models

Springer

Zoya Ignatova
Cellular Biochemistry
Max Planck Institute of Biochemistry
82152 Martinsried by Munich
Germany
ignatova@biochem.mpg.de

Karl-Heinz Zimmermann
Institute of Computer Technology
Hamburg University of Technology
21071 Hamburg
Germany
k.zimmermann@tuhh.de

Israel Martínez-Pérez
Institute of Computer Technology
Hamburg University of Technology
21071 Hamburg
Germany
martinez-perez@tuhh.de

ISBN: 978-0-387-73635-8 e-ISBN: 978-0-387-73637-2
DOI: 10.1007/978-0-387-73637-2

Library of Congress Control Number: 2007943572

Cover illustration:

Printed on acid-free paper

9 8 7 6 5 4 3 2 1

springer.com

Preface

Biomolecular computing was invented by Leonard Adleman, who made headlines in 1994 demonstrating that DNA – the double-stranded helical molecule that holds life's genetic code – could be used to carry out computations. DNA computing takes advantage of DNA or related molecules for storing information and biotechnological operations for manipulating this information. A DNA computer has extremely dense information storage capacity, provides tremendous parallelism, and exhibits extraordinary energy efficiency. Biomolecular computing has an enormous potential for in vitro analysis of DNA, assembly of nanostructures, and in vivo calculations.

The aim of this book is to introduce the beginner to DNA computing, an emerging field of nanotechnology based on the hybridization of DNA molecules. The book grew out of a research cooperation between the authors and a graduate-level course and several seminars in the master's program in Computer Engineering taught by the third author at the Hamburg University of Technology during the last few years. The book is also accessible to advanced undergraduate students and practitioners in computer science, while students, researchers, and practitioners with background in life science may feel the need to catch up on some undergraduate computer science and mathematics. The book can be used as a text for a two-hour course on DNA computing with emphasis on mathematical modelling.

This book is designed not as a comprehensive reference work, but rather as a broad selective textbook. The first two chapters form a self-contained introduction to the foundations of DNA computing: theoretical computer science and molecular biology. Chapter 2 concisely describes the abstract, logical, and mathematical aspects of computing. Chapter 3 briefly summarizes basic terms and principles of the transfer of the genetic information in living cells. The remaining chapters contain material that for the most part has not previously appeared in textbook form. Chapter 4 addresses the problem of word design for DNA computing. Proper word design is crucial in order to successfully conduct DNA computations. Chapter 5 surveys the first

generation of DNA computing. The DNA models are laboratory-scaled and human-operated, and basically aim at solving complex computational problems. Chapter 6 addresses the second generation of DNA computing. The DNA models are molecular-scaled, autonomous, and partially programmable, and essentially target the in vitro analysis or synthesis of DNA. Chapter 7 is devoted to the newest generation of DNA computing. The DNA models mainly aim at performing logical calculations under constraints found in living cells.

We have not tried to trace the full history of the subjects treated – this is beyond our scope. However, we have assigned credits to the sources that are as readable as possible for one knowing what is written here. A good systematic reference for the material covered are the Proceedings of the Annual International Workshop on DNA Based Computers.

First of all, we would like to thank Professor Volker Kasche and Professor Rudi Müller for valuable support and for providing laboratory facilities for our experimental work. We are grateful to Dr. Boris Galunsky, Stefan Goltz, Margaret Parks, and Svetlana Torgasin for proofreading, and we express our thanks to Wolfgang Brandt and Stefan Just for technical support. Finally, we thank our students for their attention, their stimulating questions, and their dedicated work, in particular, Atil Akkoyun, Gopinandan Chekrigari, Zhang Gong, Sezin Nargül, Lena Sandmann, Oliver Scharrenberg, Tina Stehr, Benjamin Thielmann, Ming Wei, and Michael Wild.

Hamburg, Munich
December, 2007

Zoya Ignatova
Israel Martínez-Pérez
Karl-Heinz Zimmermann

Contents

Acronyms

Mathematical Notation

$\emptyset$	empty set
$\mathbb{N}$	set of natural numbers
$\mathbb{N}_0$	set of non-negative integers
$\mathbb{Z}$	set of integers
$\mathbb{R}$	set of real numbers
$\mathbb{R}_0^+$	set of non-negative real numbers
$\mathbb{R}^+$	set of positive real numbers
$P(S)$	power set of a set S
δ_{ij}	Kronecker delta
$\circ$	function composition
$\lfloor x \rfloor$	largest integer $\leq x$
$\lceil x \rceil$	least integer $\geq x$
ϵ	empty string
Σ	alphabet
Δ	DNA alphabet
Σ^n	set of all length-n strings over Σ
Σ^*	set of all strings over Σ
Σ^+	set of all non-empty strings over Σ
$\Sigma^\bullet$	set of all circular strings over Σ
$\Sigma^\star$	set of all signed strings over Σ
$\|x\|$	length of string x
x^R	mirror image of string x
x^C	complement of string x
x^{RC}	reverse complement of string x
$\overline{x}$	reverse complement of string x
$\bullet x$	circular word
$\overline{f}$	negation of Boolean function f
$\|M\|$	size of automaton M

$L(M)$	language of automaton M
$L(G)$	language of grammar G
$\mathbb{B}$	set $\{0, 1\}$
$\mathbb{F}_n$	nth Boolean algebra
nt	number of nucleotides
bp	number of base pairs
aa	number of amino acids
ΔG°	Gibbs free energy
ΔH°	enthalpy
ΔS°	entropy
T	temperature
T_m	melting temperature
$[X]$	concentration of reactant X
V	volume
d_H	Hamming distance
d_H^ϕ	ϕ-Hamming distance
σ_λ	similarity function
σ_β	block similarity function
$\mathrm{com}_U(G)$	U-complement of G
$\mathrm{loc}_v(G)$	local complement of G at v

Physical Units

Angstrom	$1\,\text{Å} = 10^{-10}\,\mathrm{m}$
Atomic mass	$1\,\mathrm{Da} = 1.661 \cdot 10^{-27}\,\mathrm{kg}$
Avogadro number	$N_A = 6.022 \cdot 10^{23}\,1/\mathrm{mol}$
Boltzmann constant	$k_B = 1.38 \cdot 10^{-23}\,\mathrm{J/K}$
Dielectric constant of vacuum	$\epsilon_0 = 8.854 \cdot 10^{-12}\,\mathrm{F/m}$
Dipole moment	$1\,\mathrm{D} = 3.34 \cdot 10^{-30}\,\mathrm{Cm}$
Electron charge	$e = 1.602 \cdot 10^{-19}\,\mathrm{C}$
Electron mass	$m_e = 9.109 \cdot 10^{-31}\,\mathrm{kg}$
Gas constant	$R = 1.987\,\mathrm{cal/(K\ mol)}$
Planck constant	$h = 6.626 \cdot 10^{-34}\,\mathrm{Js}$
Reduced Planck constant	$\hbar = h/(2\pi)\,\mathrm{Js}$
Mole	$1\,\mathrm{mol} = 6.022 \cdot 10^{23}\,\mathrm{molecules}$
Molarity	$1\,\mathrm{M} = 6.022 \cdot 10^{23}\,\mathrm{mol/l}$

Chemical Notation

`H`	hydrogen atom
`O`	oxygen atom
`C`	carbon atom
`N`	nitrogen atom
`S`	sulfur atom
`P`	phosphor atom
`A`	adenine
`C`	cytosine
`G`	guanine
`T`	thymine
`U`	uracil

Chapter 1
Introduction

Abstract This introductory chapter envisions DNA computing from the perspective of molecular information technology, which is brought into focus by three confluent research directions. First, the size of semiconductor devices approaches the scale of large macromolecules. Second, the enviable computational capabilities of living organisms are increasingly traced to molecular mechanisms. Third, techniques for engineering molecular control structures into living cells start to emerge.

Nanotechnology

Nanotechnology focuses on the design, synthesis, characterization, and application of materials and devices at the nanoscale. Nanotechnology comprises near-term and molecular nanotechnology. Near-term nanotechnology aims at developing new materials and devices taking advantage of the properties operating at the nanoscale. For instance, nanolithography is a top-down technique aiming at fabricating nanometer-scale structures. The most common nanolithography technique is electron-beam-directed-write (EBDW) lithography in which a beam of electrons is used to generate a pattern on a surface.

Molecular nanotechnology aims at building materials and devices with atomic precision by using a molecular machine system. Nobel Prize-winner R. Feynman in 1959 was the first who pointed towards molecular manufacturing in his talk "There's plenty of room at the bottom," in which he discussed the prospect of maneuvering things around atom by atom without violating physical laws. The term nanonechnology was coined by N. Taniguchi in 1974, while in the 1980s E. Drexler popularized the modelling and design of nanomachines, emphasizing the constraints of precision, parsimony, and controllability, performing tasks with minimum effort. Eric Drexler's nanomachines include nano-scale manipulators to build objects atom by atom, bearings and axles built of diamond-like lattices of carbon, waterwheel-like pumps

Z. Ignatova et al., *DNA Computing Models*,
DOI: 10.1007/978-0-387-73637-2_1,

to extract and purify molecules, and tiny computers with moving parts whose size is within atomic scale.

Nanotechnology relies on the fact that material at the nanoscale exhibits quantum phenomena, which yield some extraordinary bonuses. This is due to the effects of quantum confinement that take place when the material size becomes comparable to the de Broglie wavelength of the carries (electrons and holes behaving as positively charged particles), leading to discrete energy levels. For instance, quantum dots are semiconductors at the nanoscale consisting of 100 to 100,000 atoms. Quantum dots confine the motion of (conduction band) electrons and (valency band) holes in all three spatial directions. Quantum dots are particularly useful for optical applications due to their theoretically high quantum yield (i.e., the efficiency with which absorbed light produces some effect). When a quantum dot is excited, the smaller the dot, the higher the energy and intensity of its emitted light. These optical features make quantum dots useful in biotechnological developments as well. Recently, D. Lidke and colleagues (2004) successfully employed quantum dots to visualize the movement of individual receptors on the surface of living cells with unmatched spatial and temporal resolution.

Biotechnology

Modern biotechnology in the strong sense refers to recombinant DNA technology, the engineering technology for bio-nanotechnology. Recombinant DNA technology allows the manipulation of the genetic information of the genome of a living cell. It facilitates the alteration of bio-nanomachines within the living cells and leads to genetically modified organisms. Manipulation of DNA mimics the horizonal gene transfer (HGT) in the test tube.

HGT played a major role in bacterial evolution. It is thought to be a significant technique to confer drug-resistant genes. Common mechanisms for HGT between bacterial cells are transformation, the genetic alteration of a cell resulting from introducing foreign gene material, transduction, in which genetic material is introduced via bacterial viruses (bacteriophages), and bacterial conjugation, which enables transfer of genetic material via cell-to-cell contact. HGT appears to have some significance for unicellular eukaryotes, especially for protists, while its prevalence and importance in the evolution of multicellular eukaryotes remains unclear. Today, the HGT mechanisms are used to alter the genome of an organism by exposing cells to fragments of foreign DNA encoding desirable genes, including those from another species. This DNA can be either transiently internalized into the cell or integrated into the recipient's chromosomes. Thus, it can be replicated and inherited like any other part of the genome. HGT holds promising applications in health care and in industrial and environmental processing.

Bio-Nanotechnology

Nanotechnology was invented more than three billion years ago. Indeed, nanoscale manipulators for building molecule-sized objects were required in the earliest living cells. Today, many working examples of bio-nanomachines exist within living cells. Cells contain molecular computers, which recognize the concentration of surrounding molecules and compute the proper functional output. Cells also host a large collection of molecule-selective pumps that import ions, amino acids, sugars, vitamins and all of the other nutrients needed for living. By evolutionary search and modification over trillions of generations, living organisms have perfected a plethora of molecular machines, structures, and processes (Fig. 1.1).

Bio-nanomachines are the same size as the nanomachines that are designed today. But they hardly resemble the machines of our macroscopic world and they are less familiar than E. Drexler's manipulators built along familiar rigid, rectilinear designs. D. Goodsell recently claimed that the organic, flexible forms of bio-nanomachines can only be understood by looking at the forces that made possible the evolution of life. The process of evolution by natural selection places strong constraints on biological molecules, their structure and their function. As a consequence of the evolution of life, all living organisms on earth are made of four basic molecular building blocks: proteins, nucleic acids, polysaccharides, and lipids. Proteins and nucleic acids are built in modular form by stringing subunits (monomers) together based on genetic information. These polymers may be formed in any size and with

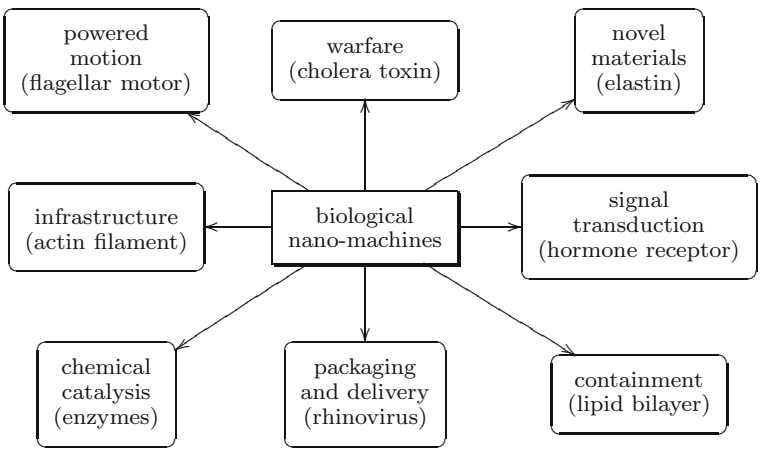

Fig. 1.1 Molecular bio-nanomachines in living cells.

monomers in any order so that they are remarkably flexible in structure and function. On the other hand, lipids and polysaccharides are built by dedicated bio-machines. Each type of new lipid or polysaccharide requires an entirely new suite of synthetic machines. Consequently, lipids and polysaccharides are less diverse in structure and more limited in function than proteins are.

The principles of protein structure and function may yield insight into nanotechnological design and fabrication. Proteins are synthesized in a modular and information-driven manner by the translation machinery of the cell, and the design of proteins is limited by a dedicated modular plan given by the genetic code. Proteins can aggregate in larger complexes due to errors in the protein-synthetic machinery or changes in the environmental conditions, so the size of proteins that may be consistently synthesized is limited. These aggregates can be built accurately and economically by protein-protein interactions based on many weak interactions (hydrogen bonds) and highly complementary shapes of interacting surfaces. Proteins are synthesized in cells and are transported to their ultimate destinations or diffuse freely in a crowded collection of competitors. A typical protein will come into partial contact with many other types of proteins and must be able to discriminate its unique target from all others. Proteins constantly flex at physiological temperatures, with covalent bonds remaining connected, and reshaped hydrogen bonds and salt bridges linking portions of the molecule or aggregate. Proteins even breathe, switching between different conformations and allowing atoms or small molecules to pass.

Synthetic Biology

The term synthetic biology was introduced by E. Kool and other speakers at the annual meeting of the American Chemical Society in 2000. Synthetic biology in broader terms aims at recreating the properties of living systems in unnatural chemical systems. That means, assembling chemical systems from unnatural components so that the systems support Darwinian evolution and are thus biological. Thus, synthetic biology may provide a way to better understand natural biology.

DNA and RNA are the molecular structures that support genetic systems on earth. Synthetic biology partially shows that the DNA and RNA backbone is not a simple scaffold to hold nucleobases but has an important role in molecular recognition, and the repeating charge provides the universal feature of genetic molecules that they work in water. Recently, S. Benner and coworkers (2003) constituted a synthetic genetic system by eight nucleotides that were generated from the natural nucleobases by shuffling hydrogen-bond donating and accepting groups. This system is part of the Bayer VERSANT branched DNA diagnostic assay which provides a reliable method to quantify HIV-1 RNA in human plasma.

Molecular Self-Assembly

Molecular self-assembly is an autonomous process of nanofabrication in which molecules or aggregates are formed without the influence of an outside source. The physicist H.R. Crane (1950) provided two basic design concepts required for molecular self-assembly. First, the contact or combined spots on the components must be multiple and weak. Thus, an array of many weak interactions is considered preferable to a few very strong interactions because the latter may lead to interactions with wrong candidates. Second, the assembled components must be highly complementary in their geometrical arrangement so that tightly packed aggregates can result. These two concepts can be observed in numerous protein-protein structures, as already mentioned.

Molecular self-assembly can theoretically create a wide range of aggregates. However, a major inherent difficulty is that the exact set of components and interactions that will construct the aggregate is difficult to determine. Recent advances in biotechnology and nanotechnology provided tools necessary to consider engineering at the molecular level. DNA computation introduced by L. Adleman in 1994 blazed a trail for the experimental study of programmable biochemical reactions, the self-assembly of DNA structures.

DNA Nanotechnology

DNA nanotechnology was initiated by N. Seeman in the 1980s. It makes use of the specificity of Watson-Crick base pairing and other DNA properties to make novel structures out of DNA. The techniques used are also employed by DNA computing and thus DNA nanotechnology overlaps with DNA computing. A key goal of DNA nanotechnology is to construct periodic arrays in two and three dimensions. For this, DNA branched junctions with specific sticky ends are designed that self-assemble to stick figures whose edges are double-stranded DNA. Today, this technology provides cubes, truncated octahedrons, and two-dimensional periodic arrays, while three-dimensional periodic arrays are still lacking. One ultimate goal is the rational synthesis of DNA cages that can host guest molecules whose structure is sought by crystallography. This would overcome the weakness of the current crystallization protocol and provide a good handle on the crystallization of all biological molecules.

Computing

A digital computer can be viewed as a network of digital components such as logic gates. The network consists of a finite number of components and the components can take on a few states. Thus, the network has only a finite number of states, and hence any realizable digital computer is a finite state

machine, although with a vast number of states. Today, these machines are realized by digital electronic circuits mainly relying on transistor technology. The success of digital electronic circuits is based on low signal-to-noise ratio, inter-connectability, low production costs, and low power dissipation. Digital electronic circuits scaled predictably during the last 30 years, with unchanged device structure and operability. Another decade of scaling appears to be feasible.

Digital computers excel in many areas of applications, while other interesting information processing problems are out of reach. The limitations are of both a theoretical and physical nature. Theoretical limitions are due to the nature of computations. The first model of effective computation was introduced by the Turing machine, which is essentially a finite state machine with an unlimited memory. In view of the generally accepted Church's thesis, the model of computation provided by the Turing machine is equivalent to any other formulation of effective computation. A machine capable of carrying out any computation is called a universal machine. Universal Turing machines exist, and every personal computer is a finite-state approximation of a univeral machine. A general result in computability reveals the existence of problems that cannot be computed by a universal machine despite potentially unlimited resources. Efficient computations can be carried out on practical computers in polynomial time and space. However, there are computational problems that can be performed in exponential time and it is unknown whether they can be performed in polynomial time and space. A prototype example is the travelling salesman problem that seeks to find a route of minimal length through all cities in a road map.

Biomolecular Computing

Current attempts to implement molecular computing fall into two categories. In the first are studies to derive molecular devices that mimic components of conventional computing devices. Examples are transistors from carbon-based semiconductors and molecular logic gates. The second includes investigations to find new computing paradigms that exploit the specific characteristics of molecules. Examples that fall into this category are computions based on diffusion-reaction or self-assembly.

A physical computation in a digital computer evolves over time. Information is stored in registers and other media, while information is processed by using digital circuits. In biomolecular computing, information is stored by biomolecules and processing of information takes place by manipulating biomolecules. The concept of biomolecular computing was theoretically discussed by T. Head in 1987, but L. Adleman in 1994 was the first to solve a small instance of the travelling salesman problem with DNA. Adleman's experiment attracted considerable interest from researchers hoping that the massive parallelization of DNA molecules would one day be the basis to

outperform electronic computers, when it comes to the computation of complex combinatorial problems. However, soon thereafter, researchers realized some of the drawbacks related to this incipient technology: a growing number of error-prone, time-consuming operations, and exponential growth of DNA volume with respect to problem size. Although some new concepts like molecular self-assembly counteracted these difficulties, no satisfactory solution to these problems has been found so far questioning the feasibility of this technology for solving intractable problems.

Therefore, molecular computing should not be viewed as a competitor for conventional computing, but as a platform for new applications. Progress in molecular computing will depend on both novel computing concepts and innovative materials. The goal of molecular information processing is to find computing paradigms capable of exploiting the specific characteristics of molecules rather than requiring the molecules to conform to a given specific formal specification.

References

1. Adleman LM (1994) Molecular computation of solutions of combinatorial problems. Science 266:1021–1023
2. Benner SA, Sismour AM (2005) Synthetic biology. Nature Rev Genetics 6: 533–543
3. Crane HR (1950) Principles and problems of biological growth. Sci Monthly 70:376–389
4. Carbone A, Seeman NC (2004) Molecular tiling and DNA self-assembly. LNCS 2340:219–240
5. Drexler KE (1992) Nanosystems, molecular machines, manufacturing and computation. Wiley and Sons, New York
6. Feynman RP (1961) Miniaturization. In: Gilbert DH (ed.) Reinhold, New York
7. Geyer C, Battersby T, Benner SA (2003) Nucleobase pairing in expanded Watson-Crick-like genetic information systems. Structure 11:1485–1498
8. Goodsell DS (2000) Biotechnology and nanotechnology. Sci Amer 88:230–237
9. Head T (1987) Formal language theory and DNA: an analysis of the generative capacity of specific recombination behaviors. Bull Math Biol 47:737–759
10. Kendrew J (1998) Encyclopedia of molecular biology. Blackwell Sci, Oxford
11. Lidke DS, Nagy P, Heintzmann R, Arndt-Jovin DJ, Post JN, Grecco H, Jares-Erijman EA, Jovin TM (2004) Quantum dot ligands provide new insights into erbB/HER receptor-mediated signal transduction. Nat Biotech 22:198–203
12. Leavitt D (2006) The man who knew too much: Alan Turing and the invention of the computer. Norton, London
13. Rawls R (2000) Synthetic biology makes its debut. Chem Eng News 78:49–53
14. Seeman N (1982) Nucleic acid junctions and lattices. J Theor Biol 99:237–247
15. Taniguchi N (1974) On the basic concept of nanotechnology. Proc Intl Conf Prod Eng Tokyo, Japan Soc Prec Eng
16. Wu R, Grossman L, Moldave K (1980) Recombinant DNA. Vol 68 Academic Press New York

Chapter 2
Theoretical Computer Science

Abstract This chapter provides a self-contained introduction to a collection of topics in computer science that focusses on the abstract, logical, and mathematical aspects of computing. First, mathematical structures called graphs are described that are used to model pairwise relations between objects from a certain collection. Second, abstract machines with a finite number of states called finite state automata are detailed. Third, mathematical models of computation are studied and their relationships to formal grammars are explained. Fourth, combinatorial logic is introduced, which describes logic circuits whose output is a pure function of the present input only. Finally, the degrees of complexity to solve a problem on a computer are outlined.

2.1 Graphs

Graph theory provides important tools to tackle complex problems in different parts of science.

2.1.1 Basic Notions

A *graph* is a pair $G = (V, E)$, consisting of a non-empty set V and a set E of two-element subsets of V. The elements of V are called *vertices* and the elements of E are termed *edges*. An edge $e = \{u, v\}$ is also written as $e = uv$ (or $e = vu$). If $e = uv$ is an edge, then u and v are *incident* with e, u and v are *adjacent*, and u and v form the *end-vertices* of e. In the following, we consider *finite* graphs (i.e., graphs with finite vertex sets). The number of vertices and edges of a graph G is called the *order* and *size* of G, respectively.

A graph is described by a *diagram*, in which the vertices are points in the drawing plane and the edges are line segments.

Z. Ignatova et al., *DNA Computing Models*,
DOI: 10.1007/978-0-387-73637-2_2,

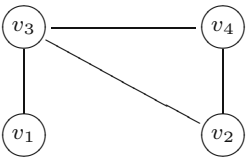

Fig. 2.1 Diagram of the graph in Example 2.1.

Example 2.1. The graph G with vertex set $V = \{v_1, \ldots, v_4\}$ and edge set $E = \{v_1v_3, v_2v_3, v_2v_4, v_3v_4\}$ is given by the diagram in Figure 2.1. ♢

A graph $G = (V, E)$ has neither *loops* nor *multiple edges*. Loops are one-element subsets of V (i.e., edges incidenting with only one vertex). Multiple edges are multisets over the two-element subsets of V. A *multiset* over a set M is a mapping $f : M \to \mathbb{N}_0$ assigning to each element m in M the number of occurrences $f(m)$ in the multiset.

Let $G = (V, E)$ be a graph. The number of edges which are incident with a vertex $v \in V$ is called the *degree* of v and is denoted by $d(v)$. A vertex v in G is called *isolated* if $d(v) = 0$. If all vertices in G have the same degree k, then the graph G is called *k-regular*.

Lemma 2.2. (Handshaking) *For each graph $G = (V, E)$,*

$$\sum_{v \in V} d(v) = 2|E| \, . \tag{2.1}$$

Proof. On the left hand side, each edge in the sum is counted twice, once for each vertex. □

Corollary 2.3. *In each graph, the number of vertices of odd degree is even.*

Example 2.4. Can 333 phones be connected so that each phone is connected with three phones? The answer is no, because the sum of degrees in this network would be odd ($333 \cdot 3$), contradicting the handshaking lemma. ♢

The *degree sequence* of a graph G is given by the decreasing list of degrees of all vertices in G. For instance, the graph in Figure 2.1 has the degree sequence $(3, 2, 2, 1)$. On the other hand, not every decreasing sequence of natural numbers is the degree sequence of a graph, such as $(5, 3, 2, 2, 2, 1)$, since the sum of degrees is odd.

Subgraphs

Let $G = (V, E)$ be a graph. A *subgraph* of G is a graph $G' = (V', E')$ with $V' \subseteq V$ and $E' \subseteq E \cap \binom{V'}{2}$, where $\binom{V'}{2}$ is the set of 2-element subsets of V'. The subgraph G' is considered to be *induced* from its edge set E'. If $E' = E \cap \binom{V'}{2}$, the subgraph G' is *induced* from its vertex set V'.

Fig. 2.2 Two subgraphs, G_1 and G_2, of the graph G in Figure 2.1.

Example 2.5. In view of the graph G in Figure 2.1, two of its subgraphs G_1 and G_2 are illustrated in Figure 2.2. The subgraph G_2 is induced from the vertex set $\{v_2, v_3, v_4\}$, while the subgraph G_1 is not because the edge v_2v_3 is missing. ♢

Isomorphisms

Let $G = (V, E)$ and $G' = (V', E')$ be graphs. A mapping $\phi : V \to V'$ is called an *isomorphism* from G onto G', if ϕ is bijective and for all vertices $u, v \in V$, $uv \in E$ if and only if $\phi(u)\phi(v) \in E'$. Two graphs G and G' are termed *isomorphic* if there is an isomorphism from G onto G'. Clearly, isomorphic graphs have the same order, size, and degree sequence.

Example 2.6. The graphs in Figure 2.3 are isomorphic. An isomorphism is given by $\phi(v_i) = u_i$ for $1 \le i \le 4$. ♢

2.1.2 Paths and Cycles

Let $G = (V, E)$ be a graph. A sequence $W = (v_0, \ldots, v_k)$ of vertices $v_i \in V$ is called a *path* in G, if for each i, $1 \le i \le k$, we have $v_{i-1}v_i \in E$. The vertex v_0 is the *initial vertex* and the vertex v_k the *final vertex* in W. The *length* of W equals n, the number of edges in W. A path W is called *simple* if W contains each vertex at most once.

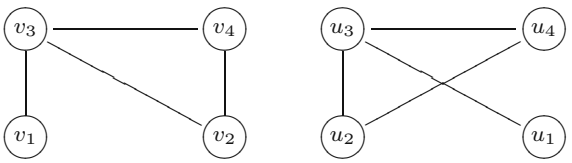

Fig. 2.3 Two isomorphic graphs.

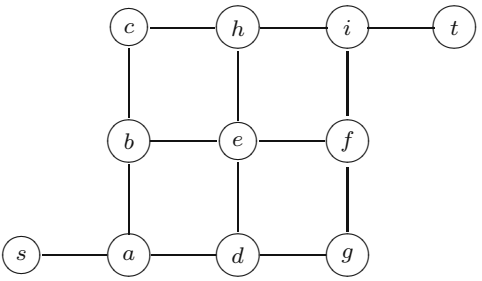

Fig. 2.4 A Manhattan network.

Example 2.7. The graph in Figure 2.4 contains several simple paths of length 6 such as (s, a, d, g, f, i, t) and (s, a, b, e, h, i, t). ♢

A *cycle* in G is a path in G, in which the initial and final vertex are identical. A cycle is called *simple* if it contains each vertex at most once (apart from the initial and final vertex). Each edge uv provides a simple cycle (u, v, u) of length 2.

Example 2.8. The graph in Figure 2.4 contains several simple cycles of length 6 such as (a, b, c, h, e, d, a) and (a, b, e, f, g, d, a). ♢

Connectedness

Let $G = (V, E)$ be a graph. Two vertices $u, v \in V$ are called *connected* in G, briefly $u \equiv_G v$, if $u = v$ or there is a path from u to v in G. If any two vertices in G are connected, then G is termed *connected*. For each vertex v in G, define the set of vertices connected to v as $C_G(v) = \{u \in V \mid u \equiv_G v\}$.

Theorem 2.9. *Let $G = (V, E)$ be a graph. The set of connected sets $C_G(v)$, $v \in V$, of G is a partition of V (i.e., the sets are non-empty and their union provides the overall set V, and any two sets are either equal or disjoint).*

A subgraph induced by a connected set of G is called a *component* of G. If G is connected, then there is only one component.

Example 2.10. The graph in Figure 2.5 consists of two components: $\{a, b\}$ and $\{c, d, e\}$. ♢

Theorem 2.11. *Let $G = (V, E)$ be a connected graph and let K be a simple cycle in G. If an edge $e \in G$ on the cycle K is deleted from G, the resulting subgraph of G is still connected.*

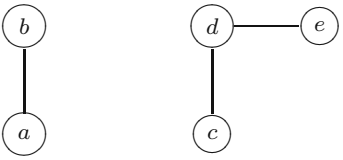

Fig. 2.5 A graph with two components.

2.1.3 Closures and Paths

A *directed graph* is a pair $G = (V, E)$, consisting of a non-empty set V and a subset E of $V \times V$ (Fig. 5.1). Each undirected graph can be assigned a directed graph so that each edge $e = uv$ is replaced by the edges (u, v) and (v, u). The edge set of a directed graph forms a binary relation on V. The *indegree* of a vertex v in G is the number of incoming edges (u, v), $u \in V$, and the *outdegree* of v is the number of outcoming edges (v, w), $w \in V$.

Let R be a binary relation on a set A (i.e., $R \subseteq A \times A$). Define the powers of R inductively as follows:

- $R^0 = \{(a, a) \mid a \in A\}$,
- $R^{n+1} = R \circ R^n = \{(a, c) \mid (a, b) \in R, (b, c) \in R^n, b \in A\}$ for all $n \geq 0$.

Clearly $R^1 = R^0 \circ R = R$. The definition implies the following:

Theorem 2.12. *Let $G = (V, E)$ be a directed graph and let $n \geq 0$ be an integer. The nth power E^n provides all paths of length n between any two vertices in G.*

Define $R^+ = \bigcup_{n \geq 1} R^n$ and $R^* = \bigcup_{n \geq 0} R^n = R^+ \cup R^0$.

Theorem 2.13. *Let R be a binary relation on a set A. The relation R^+ is the smallest transitive relation containing R. The relation R^* is the smallest reflexive, transitive relation that contains R.*

Proof. Let $R' = \bigcup_{n \geq 1} R^n$. Claim that R' is transitive. Indeed, let $a, b, c \in A$ with $(a, b) \in R'$ and $(b, c) \in R'$. Then there are non-negative integers m and n so that $(a, b) \in R^m$ and $(b, c) \in R^n$. Thus, $(a, c) \in R^{m+n}$ and hence $(a, c) \in R'$. Moreover, $R = R^1$ and so $R \subseteq R'$.

Finally, let R'' be a transitive relation on A, which contains R. Claim that $R' \subseteq R''$. Indeed, let $a, b \in A$ with $(a, b) \in R'$. Then there is a non-negative integer n so that $(a, b) \in R^n$. Consequently, there are elements $a_1, \ldots, a_{n-1}$ in A so that $(a, a_1) \in R$, $(a_i, a_{i+1}) \in R$ for all $1 \leq i \leq n-2$, and $(a_{n-1}, b) \in R$. But R is a subset of R'' and so $(a, a_1) \in R''$, $(a_i, a_{i+1}) \in R''$ for all $1 \leq i \leq n-2$, and $(a_{n-1}, b) \in R''$. As R'' is transitive, it follows that $(a, b) \in R''$ and so the claim is established.

The second assertion is similarly proved. □

The relation R^+ is called the *transitive closure* of R, while the relation R^* is termed the *reflexive, transitive closure* of R.

Distances

Let $G = (V, E)$ be a graph and let $u, v \in V$. Define the *distance* between u and v in G as follows:

$$d_G(u,v) = \begin{cases} 0 & \text{if } u = v, \\ \infty & \text{if } u \text{ and } v \text{ are not connected,} \\ l & \text{if } l \text{ is the length of a shortest path in } G \text{ from } u \text{ to } v. \end{cases} \tag{2.2}$$

Theorem 2.14. *Let $G = (V, E)$ be a graph. The distance d_G defines a metric on G. That is, for all $u, v, w \in V$, $d_G(u, v) = 0$ if and only if $u = v$, $d_G(u, v) = d_G(v, u)$, and $d_G(u, w) \leq d_G(u, v) + d_G(v, w)$.*

Notice that each metric d_G satisfies $d_G(u, v) \geq 0$ for all $u, v \in V$, because $0 = d_G(u, u) \leq d_G(u, v) + d_G(v, u) = 2d_G(u, v)$.

2.1.4 Trees

A graph is called *cycle-free* or a *forest* if it contains no simple cycles of length at least 3. A connected forest is called a *tree* (Fig. 2.6).

Theorem 2.15. *Each tree contains at least two vertices of degree 1.*

Proof. Let G be a tree. Let u und v be vertices in G so that their distance $d_G(u, v)$ is maximal. Let $W = (u, v_1, \ldots, v_{k-1}, v)$ be a shortest path in G from u to v. Suppose that u has two adjacent vertices, v_1 and w. Then by hypothesis, $d_G(w, v) \leq d_G(u, v)$. Thus there is a shortest path from w to v not using u. So G contains a simple cycle of length at least 3. A contradiction. Consequently, u has degree 1 and, by symmetry, also v. □

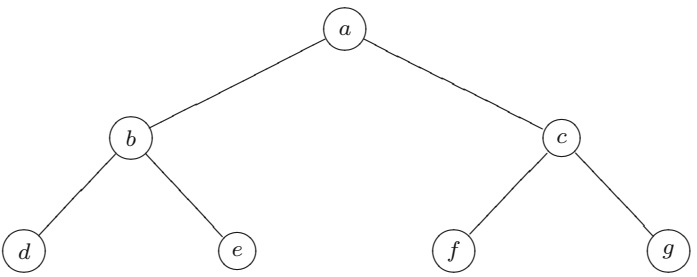

Fig. 2.6 A tree.

Theorem 2.16. *For each tree $G = (V, E)$, we have $|E| = |V| - 1$.*

Proof. The case $|V| = 1$ is clear. Let G be a tree with $|V| > 1$ vertices. In view of Theorem 2.15, the graph G contains a vertex of degree 1. If this vertex is deleted, the resulting subgraph $G' = (V', E')$ of G is a tree, too. By induction hypothesis, $1 = |V'| - |E'| = (|V| - 1) - (|E| - 1) = |V| - |E|$. □

Let $G = (V, E)$ be a graph. A *spanning tree* of G is a subgraph of G, which forms a tree and contains each vertex of G (Fig. 2.7).

Theorem 2.17. *Each connected graph contains a spanning tree.*

Proof. Let $G = (V, E)$ be a connected graph. If $|E| = 1$, then the assertion is clear. Let $|E| > 1$. If G is a tree, then G is its own spanning tree. Otherwise, there is a simple cycle of length at least 3 in G. Delete one edge from this cycle. The resulting subgraph G' of G has $|E| - 1$ edges and is connected by Theorem 2.11. Thus by induction hypothesis, G' has a spanning tree, and this spanning tree is also a spanning tree of G. □

Theorem 2.18. *A connected graph $G = (V, E)$ is a tree if and only if $|E| = |V| - 1$.*

Proof. Let $|E| = |V| - 1$. Suppose G is not a tree. Then G contains a simple cycle of length at least 3. Delete one edge from this cycle. The resulting subgraph $G' = (V, E')$ of G is connected by Theorem 2.11. The edge set in G' fulfills $|E'| < |V| - 1$. On the other hand, Theorems 2.16 and 2.17 imply that G contains a spanning tree with $|V| - 1$ edges, which lies in E'. A contradiction. The reverse assertion was proved in Theorem 2.16. □

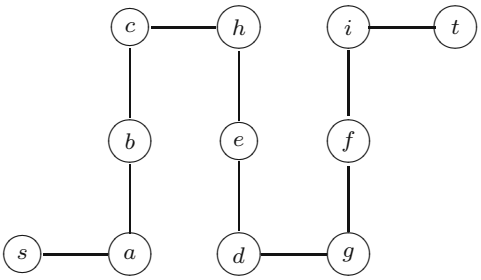

Fig. 2.7 A spanning tree of the graph in Figure 2.4.

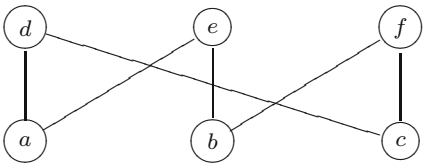

Fig. 2.8 A bipartite graph with partition $\{\{a, b, c\}, \{d, e, f\}\}$.

2.1.5 Bipartite Graphs

A graph $G = (V, E)$ is called *bipartite* if there is a partition of V into subsets V_1 und V_2 so that every edge in G has one end-vertex in V_1 and one end-vertex in V_2 (Fig. 2.8).

Theorem 2.19. *A connected graph G is bipartite if and only if G contains no cycles of odd length.*

Proof. Let $G = (V, E)$ be a bipartite graph with partition $\{V_1, V_2\}$. Let $K = (v_0, v_1, \ldots, v_k)$ be a cycle in G. If $v_0 \in V_1$, then $v_1 \in V_2$, $v_2 \in V_1$, and so on. Thus, $v_k = v_0 \in V_1$ and hence the cycle K has even length. If $v_0 \in V_2$, then the result is the same.

Conversely, assume that G contains no cycles of odd length. Let $v \in V$ and define

$$V_1 = \{u \in V \mid d_G(v, u) \equiv 1 \bmod 2\}$$

and

$$V_2 = \{u \in V \mid d_G(v, u) \equiv 0 \bmod 2\} .$$

Clearly, $\{V_1, V_2\}$ is a partition of V. Suppose that there is an edge uw in G with $u, w \in V_1$. Then there is a cycle, consisting of the edge uw, a path of length $d_G(w, v)$ from w to v, and a path of length $d_G(v, u)$ from v to u. This cycle has total length $1 + d_G(w, v) + d_G(v, u)$, which is odd by definition of V_1 and V_2. A contradiction. Similarly, there exists no edge uw in G with $u, w \in V_2$. □

2.2 Finite State Automata

Finite state automata are a simple type of machine studied first in the 1940s and 1950s. These automata were originally proposed to model brain functions. Today, finite state automata are mainly used to specify various kinds of hardware and software components.

2.2.1 Strings and Languages

Let Σ be a finite set and let n be a non-negative integer. A *word* or *string* of length n over Σ is a sequence $x = a_1 \dots a_n$ so that $a_i \in \Sigma$ for each $1 \leq i \leq n$. The *length* of a string x is denoted by $|x|$. The set Σ is termed *alphabet* and the elements of Σ are called *characters* or *symbols*. The *empty string* corresponds to the empty sequence and is denoted by ε. For instance, the strings of length at most 2 over $\Sigma = \{a, b\}$ are ε, a, b, aa, ab, ba, and bb.

Define Σ^n as the set of all strings of length n over Σ. In particular, $\Sigma^0 = \{\varepsilon\}$ and $\Sigma^1 = \Sigma$. Moreover, let Σ^* be the set of all strings over Σ (i.e., Σ^* is the disjoint union of all sets Σ^n, $n \geq 0$). Write Σ^+ for the set of all non-empty strings over Σ (i.e., Σ^+ is the disjoint union of all sets Σ^n, $n \geq 1$). Any subset of Σ^* is called a *(formal) language* over Σ.

The *concatenation* of two strings x and y is the string xy formed by joining x and y. Thus, the concatenation of the strings "home" and "work" is the string "homework". Let x be a string over Σ. A *prefix* of x is a string u over Σ so that $x = uv$ for some string v over Σ. Similarly, a *postfix* of x is a string v over Σ so that $x = uv$ for some string u over Σ.

A *monoid* is a set M which is closed under an associative binary operation, denoted by '$\cdot$', and has an *identity element* $\varepsilon \in M$. That is, for all x, y, and z in M, $(x \cdot y) \cdot z = x \cdot (y \cdot z)$, and $x \cdot \varepsilon = x = \varepsilon \cdot x$. This monoid is written as a triple $(M, \cdot, \varepsilon)$. In particular, the set Σ^* forms a monoid with the operation of concatenation of strings and with the empty string as the identity element. For any two languages L_1 and L_2 over Σ, write $L_1 L_2 = \{xy \mid x \in L_1, y \in L_2\}$ to denote their *concatenation*.

Let $(M, \cdot, \varepsilon)$ and $(M', \circ, \varepsilon')$ be monoids. A *homomorphism* from M to M' is a mapping $\phi : M \to M'$ so that for all $x, y \in M$, $\phi(x \cdot y) = \phi(x) \circ \phi(y)$ and $\phi(\varepsilon) = \varepsilon'$. An *anti-homomorphism* from M to M' is a mapping $\phi : M \to M'$ so that for all $x, y \in M$, $\phi(x \cdot y) = \phi(y) \circ \phi(x)$ and $\phi(\varepsilon) = \varepsilon'$. A homomorphism $\phi : M \to M$ is called a *morphic involution* if ϕ^2 is the identity mapping. The simplest morphic involution is the identity mapping. An anti-homomorphism $\phi : M \to M$ so that ϕ^2 is the identity mapping is termed an *anti-morphic involution*.

Let Σ be an alphabet. Each mapping $f : \Sigma \to \Sigma$ can be extended to a homomorphism $\phi : \Sigma^* \to \Sigma^*$ so that $\phi(a) = f(a)$ for each $a \in \Sigma$. To see this, put $\phi(a_1 \dots a_n) = f(a_1) \dots f(a_n)$ for each string $a_1 \dots a_n \in \Sigma^*$. Similarly, each mapping $f : \Sigma \to \Sigma$ can be extended to an anti-homomorphism $\phi : \Sigma^* \to \Sigma^*$. For this, define $\phi(a_1 \dots a_n) = f(a_n) \dots f(a_1)$ for each string $a_1 \dots a_n \in \Sigma^*$.

Single strands of DNA are quaternary strings over the DNA alphabet $\Delta = \{\mathtt{A}, \mathtt{C}, \mathtt{G}, \mathtt{T}\}$. Strands of DNA are oriented (e.g., `AACG` is distinct from `GCAA`). An orientation is introduced by declaring that a DNA string begins with the 5'-end and ends with the 3'-end. For example, the strands `AACG` and `GCAA` are denoted by `5'-AACG-3'` and `5'-GCAA-3'`, respectively. Furthermore,

in nature DNA is predominantly double-stranded. Each natural strand occurs with its reverse complement, with reversal denoting that the sequences of the two strands are oppositely oriented, relative to one other, and with complementarity denoting that the allowed pairings of letters, opposing one another on the two strands, are the *Watson-Crick pairs* $\{\mathtt{A},\mathtt{T}\}$ and $\{\mathtt{G},\mathtt{C}\}$. A double strand results from joining reverse complementary strands in opposite orientations:

```
5'-AACGTC-3'
3'-TTGCAG-5' .
```

DNA strands that differ by orientation are mapped onto each other by the *mirror involution* $\mu : \Delta^* \to \Delta^*$, which is the anti-homomorphism extending the identity mapping. For example, $\mu(\mathtt{AACG}) = \mathtt{GCAA}$. The mirror image of a DNA string x is denoted by $x^R = \mu(x)$. Moreover, the *complementarity involution* is the morphic involution $\phi : \Delta^* \to \Delta^*$ that extends the *complementarity mapping* $f : \Delta \to \Delta$ given by $f(\mathtt{A}) = \mathtt{T}$, $f(\mathtt{C}) = \mathtt{G}$, $f(\mathtt{G}) = \mathtt{C}$, and $f(\mathtt{T}) = \mathtt{A}$. For example, $\phi(\mathtt{AACG}) = \mathtt{TTGC}$. The complementary image of a DNA string x is denoted by $x^C = \phi(x)$. Finally, reverse complementary strands are obtained by the *reverse complementarity involution* or *Watson-Crick involution* $\tau = \mu\phi$ $(= \phi\mu)$, which is composed of the mirror involution μ and the complementarity involution ϕ (in any order). For example, $\tau(\mathtt{AACG}) = \mathtt{CGTT}$. The reverse complementary image of a DNA string x is denoted by $x^{RC} = \tau(x)$.

2.2.2 Deterministic Finite State Automata

A finite state automaton can be thought of as a processing unit reading an input string and accepting or rejecting it. A *(deterministic) finite state automaton* is a quintuple $M = (\Sigma, S, \delta, s_0, F)$ so that Σ is an alphabet, S is a finite set of *states* with $S \cap \Sigma = \emptyset$, $s_0 \in S$ is the *initial state*, $F \subseteq S$ is the set of *final states*, and $\delta : S \times \Sigma \to S$ is the *transition function*, where the transition $\delta(s, a) = s'$ is also graphically written as $s \xrightarrow{a} s'$. The *size* of a finite state automaton M, denoted by $|M|$, is the number $|S| + |\delta|$.

Example 2.20. Consider the finite automaton M with state set $S = \{s_0, s_1\}$, input alphabet $\Sigma = \{a, b\}$, initial state s_0, final state set $F = \{s_0\}$, and transition function δ given by the transition graph in Figure 2.9. ♢

A finite state automaton M computes a string $x = a_1 \dots a_n$ as follows: M starts in the initial state s_0, reads the first symbol a_1 and enters the state $s_1 = \delta(s_0, a_1)$. Then it reads the next symbol a_2 and enters the state $s_2 = \delta(s_1, a_2)$ and so on. After reading the last symbol a_n, the automaton enters the state $s_n = \delta(s_{n-1}, a_n)$. Therefore, the processing of an input string x can be traced by the associated path $(s_0, \dots, s_n)$ in the transition graph. If the last state s_n is a final state, then M *accepts* the string x; otherwise,

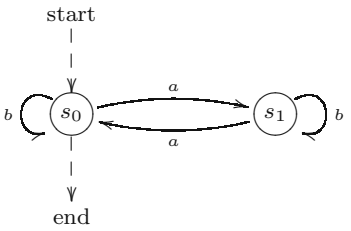

Fig. 2.9 Transition graph of finite state automaton.

M rejects the string x. The *language* of M is the set of all strings accepted by M,

$$L(M) = \{x \in \Sigma^* \mid M \text{ accepts } x\} \,. \tag{2.3}$$

The multi-step behavior of a finite state automaton M can be formally described by the *extended transition function* $\delta^* : S \times \Sigma^* \to S$, which is inductively defined as follows:

- $\delta^*(s, \varepsilon) = s$,
- $\delta^*(s, ax) = \delta^*(\delta(s, a), x)$ for all $s \in S$, $a \in \Sigma$, and $x \in \Sigma^*$.

In particular, $\delta^*(s, a) = \delta(s, a)$ for all $s \in S$ and $a \in \Sigma$. The language of M is thus given by

$$L(M) = \{x \in \Sigma^* \mid \delta^*(s_0, x) \in F\} \,. \tag{2.4}$$

If $L = L(M)$ is a finite language, the size of the accepting automaton M is in the worst case proportional to the total length of all strings in L.

Example 2.21. Consider the finite state automaton M in Example 2.20. The language of M consists of all strings over Σ which contain an even number of a's. For instance, $\delta^*(s_0, abab) = s_0$ and $\delta^*(s_0, bbab) = s_1$. ♢

2.2.3 Non-Deterministic Finite State Automata

Non-deterministic machines may provide several next states for each pair of state and input symbol. A *non-deterministic finite state automaton* is a quintuple $M = (\Sigma, S, \delta, S_0, F)$ so that Σ is an alphabet, S is a finite set of *states* with $S \cap \Sigma = \emptyset$, $S_0 \subseteq S$ is the set of *initial states*, $F \subseteq S$ is the set of *final states*, and $\delta : S \times \Sigma \to P(S)$ is the *transition function*.

A non-deterministic finite state automaton M computes a string $x = a_1 \dots a_n$ similar to its deterministic counterpart. However, M can start in any initial state, and if it happens to enter the state s and reading symbol a, then it can enter any state in $\delta(s, a)$. Therefore, the processing of the input string x can be traced by *all* paths $(s_0, \dots, s_n)$ in the corresponding transition graph so that $s_0 \in S_0$ and $s_i \in \delta(s_{i-1}, a_i)$ for all $1 \le i \le n$.