Emergence, Complexity and Computation ECC

Georgios Ch. Sirakoulis
Andrew Adamatzky *Editors*

Robots and Lattice Automata

Springer

Georgios Ch. Sirakoulis · Andrew Adamatzky
Editors

Robots and Lattice Automata

Springer

Editors
Georgios Ch. Sirakoulis
Department of Electrical and Computer Engineering
Democritus University of Thrace
Xanthi
Greece

Andrew Adamatzky
Unconventional Computing Centre
University of the West of England
Bristol
UK

ISSN 2194-7287
ISBN 978-3-319-10923-7
DOI 10.1007/978-3-319-10924-4

ISSN 2194-7295 (electronic)
ISBN 978-3-319-10924-4 (eBook)

Library of Congress Control Number: 2014951740

Springer Cham Heidelberg New York Dordrecht London

Printed on acid-free paper

Springer is part of Springer Science+Business Media (www.springer.com)

Preface

Robots and Automata are notionally related. In this context, Automata (originated from the latinization of the Greek word "αυτόματον") as self-operating autonomous machines, invented from ancient years can be easily considered as the first steps of these robotic-like efforts. On other words, an Automaton is a self-operating machine, while a robot is a hardware agent with role(s) to operate usually without an immediate human operator. Automata are useful tools for formal descriptions of robots. Automata themselves are formally represented by final state machines: the abstract machines which take finite number of states and change their state while triggered by certain conditions. Authors of the book bring together concepts, architectures and implementations of Lattice Automata and Robots. Lattice Automata are minimal universal instantiation of space and time. A Lattice Automaton is either a regular array of finite state machines or collectives of mobile finite state machines inhabiting a discrete space. In both cases the finite states machines, or Automata, update their states by the same rules depending on states of their immediate neighbours. Automata and Robots often share the same notional meaning: Automata are mathematical models of robots and also they are integral parts of robotic control systems.

The book opens with inspiring text by Rosenberg—Chap. 1—on computational potential of groups of identical finite-state machines. The chapter lays somewhat foundational theoretical background for the rest of the book.

Modular robots are kinematic machines of many units capable for changing its topology by dynamically updating connections between the units. To develop efficient algorithms of reconfiguration, we represent the robotic units by configurations of Lattice Automata and study Automaton transition rules corresponding to reconfiguration. The topic is studied in full details in three chapters: Chap. 2 by Stoy introduces the reader to the theoretical and general aspects of modular reconfigurable robots in Lattice Automata; Chap. 3 by Eckenstein and Yim reproduces all the up-to-date related works and corresponding modular reconfigurable robotic systems; while in Chap. 4, Tomita and co-authors provide full details for some of these modular systems, namely Fractum and M-Tran in every possible aspect and discuss the general problems of Lattice-based robotic systems.

Motion control and path planning are amongst key problems of robotics, they put high demands on detailed knowledge of environment and consume substantial computational resources. Five chapters explicitly deal with these problems. Thus, Arena and co-authors, in Chap. 5, use Automaton networks to control locomotion of the fly-inspired robot. Efficient ways of routing, an abstract version of path planning, are designed and analysed by Hoffman and Désérable in Chap. 6. Marchese proposes to use particular families of Cellular Automata to provide an optimal representation of space and maps in precise parallel motion planning, in Chap. 7. Charalampous and co-authors in Chap. 8 adapt classical designs of Cellular Automaton based shortest path finders to undertake autonomous collision-free navigation. Moreover, Ioannidis and co-authors proposed the employment of Cellular Automata advanced with Ant Colony Optimization techniques resulting to Cellular Robotic Ants synergy coordination for tackling the path planning problem for robotic teams in Chap. 9.

Further applications of Lattice Automata in Robotics are presented in the following chapters. A novel method of map representation is proposed in Chap. 10 by Kapoutsis and his co-authors. There, a configuration of elevation heights is converted to cells' states; thus, an entire map is represented by a Cellular Automaton configuration. Cellular Automata have been a classical tool in image processing community since mid-1970s, yet, there is still vast lands of unexplored features and algorithms. In his Chap. 11, Nalpantidis demonstrates practical, real-life implementation of Cellular Automaton algorithms onboard of a mobile robot.

The last two chapters deal with cooperative actions in large-scale robotic collectives. In both chapters, robots are oscillating mechanisms arranged on a two-dimensional array: their aim is to adjust their oscillations or states to produce a specified vibration pattern. Silva and co-authors, in Chap. 12 provide modelling and analysis of the space–time behaviour of such collectives and transitions between different modes of behaviour. Application of the vibrating automaton array to physical manipulator of objects in real life is studied by Georgilas and co-authors in Chap. 13. They show how Automaton model of an sub-excitable medium can be used to purposefully transport objects.

All chapters are written in an accessible manner and lavishly illustrated. The book will help computer and robotic scientists and engineers to understand mechanisms of decentralised functioning of robotic collectives and to design future and emergent reconfigurable, parallel and distributed robotic systems.

Georgios Ch. Sirakoulis
Andrew Adamatzky

Contents

Chapter 1
Algorithmic Insights into Finite-State Robots

Arnold L. Rosenberg

Abstract Modern technology has enabled the deployment of small computers that can act as the "brains" of mobile robots. Multiple advantages accrue if one can deploy simpler computers rather than more sophisticated ones: For a fixed cost, one can deploy more computers, hence benefit from more concurrent computing and/or more fault-tolerant design—both major issues with assemblages of mobile "intelligent" robots. This chapter explores the capabilities and limitations of computers that execute simply structured *finite-state programs*. The robots of interest operate within constrained physical settings such as warehouses or laboratories; they operate on tesselated "floors" within such settings—which we view formally as meshes of tiles. The major message of the chapter is that teams of (identical) robots whose "intellects" are powered by finite-state programs are capable of more sophisticated algorithmics than one might expect, even when the robots must operate: (*a*) *without the aid of centralized control* and (*b*) using algorithms that are *scalable*, in the sense that they work in meshes/"floors" of arbitrary sizes. A significant enabler of robots' algorithmic sophistication is their ability to use their host mesh's edges—i.e., the walls of the warehouses or laboratories—when orchestrating their activities. The capabilities of our "finite-state robots" are illustrated via a variety of algorithmic problems that involve path planning and exploration, in addition to the rearranging of labeled objects.

1.1 Introduction

Modern technology has enabled the deployment of small computers that can act as the "brains" of mobile robots. Multiple advantages accrue if one can deploy simpler computers rather than more sophisticated ones: For a fixed cost, one can deploy more computers, hence benefit from more concurrent computing and/or more fault-tolerant design—both major issues with assemblages of mobile "intelligent" robots. This chapter explores the capabilities and limitations of computers that execute simply

A.L. Rosenberg (✉)
Northeastern University, Boston, MA 02115, USA
e-mail: rsnbrg@ccs.neu.edu

G.Ch. Sirakoulis and A. Adamatzky (eds.), *Robots and Lattice Automata*,
Emergence, Complexity and Computation 13, DOI 10.1007/978-3-319-10924-4_1

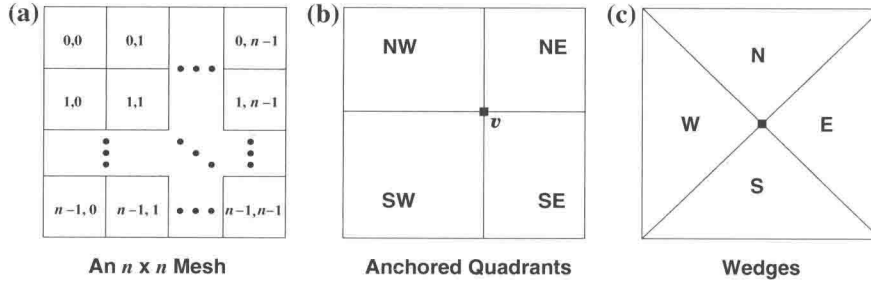

Fig. 1.1 **a** The $n \times n$ mesh $\mathcal{M}_n$; **b** $\mathcal{M}_n$ partitioned into the four *quadrants* determined by *anchor tile* v. **c** $\mathcal{M}_n$ partitioned into its four *wedges*

structured *finite-state programs*, perhaps the simplest type of program that one can expect to enable sophisticated robot behavior. The robots of interest—which we call *finite-state machines* (*FSMs*, for short), to emphasize the finite-state property—operate within constrained physical settings such as warehouses or laboratories; they operate on tesselated "floors" within such settings—which we view formally as instances of the $n \times n$ *mesh of tiles* $\mathcal{M}_n$ (Fig. 1.1a).

The major message of the chapter is that teams of (identical) FSMs are capable of more sophisticated algorithmics than one might expect, even when the FSMs must operate: (*a*) *without the aid of centralized control* and (*b*) using algorithms that are *scalable*, in the sense that they work in meshes/"floors" of arbitrary sizes. A significant enabler of robots' algorithmic sophistication is their ability to exploit the *edges* of the meshes they operate in—i.e., the walls of the warehouses or laboratories—when orchestrating their activities. The capabilities of finite-state robots are illustrated here via a variety of algorithmic problems that involve path planning and exploration, in addition to the rearranging of labeled objects (that sit within some tiles of the home mesh). We note again the major points that FSMs operate *without centralized control* while executing algorithms that are *scalable*.

Our study focuses on algorithmic problems that emerge from complementary avenues of investigation with histories that span several decades. The literature on automata theory and its applications contains studies such as [3, 5, 8, 10, 24, 27] that focus on the (in)ability of FSMs to explore graphs with goals such as finding "entrance"-to-"exit" paths or exhaustively visiting all of a graph's nodes or all of its edges. Other studies, e.g., [4, 15, 17, 23, 26, 36], focus on algorithms that enable FSMs that populate the tiles of (multidimensional) meshes—*cellular automata*—to tightly synchronize, a crucial component of many activities that must be performed without centralized control; the cellular automaton model dates back a half-century [38] but remains of interest today [14, 39]. Yet other automata-theoretic studies update the historical string-recognition work of classical finite-automata theory—cf., [25, 28]—to more ambitious domains such as graphics [7, 13, 19, 20, 29]. The robotics literature contains numerous studies—e.g., [1, 2, 11, 18, 35]—that explore ants as a metaphor for simple robots that collaborate to accomplish complex tasks;

the interesting topic of "virtual pheromones" within this metaphor is studied in [11, 18, 31, 35]. Cellular automata appear in many studies of robotic applications of automata-theoretic concepts: application- and implementation-oriented studies as well as theoretical ones [6, 11, 16, 22, 32, 35, 37]. The current chapter melds the automata-theoretic and robotic points of view by studying FSMs that operate within square meshes; most of the problems we discuss are more closely motivated by robotics than automata theory, although a few emerge from the world of language-oriented studies.

The specific algorithmic challenges that we study are inspired by our earlier work on FSMs, which itself emerged from our work on the Cellular ANTomaton model [32], a marriage of robotics and cellular automata. All of our studies demand algorithms that are *scalable* in the sense that they work in meshes $\mathcal{M}_n$ of arbitrary size, i.e., for arbitrarily large values of n. Our first study involving FSMs was [31], which focused on the *Parking Problem* for FSMs; this problem requires each FSM in a mesh to go to its closest corner and has FSMs within each corner organize into a maximally compact formation (i.e., one that minimizes the FSMs' aggregate distance to their nearest corners). A central component of parking is to have each FSM determine which *quadrant* of $\mathcal{M}_n$ it resides in (cf. Fig. 1.1b); because the home-quadrant determination problem is treated in detail in [31], we focus here on a kindred, but rather different problem that requires FSMs to determine their home *wedges* (cf. Fig. 1.1c). Our next study of FSMs, in [33], allowed the tiles of $\mathcal{M}_n$ to be labeled from a given repertoire. The study required FSMs to move to a specified tile $v_{\varphi,\psi} = \langle \lfloor \varphi n \rfloor, \lfloor \psi n \rfloor \rangle$ of $\mathcal{M}_n$, identified by a prespecified pair of positive rational numbers $\langle \varphi, \psi \rangle$.

> Note that: (a) the rational numbers $\varphi = a/b$ and $\psi = c/d$ are *fixed for each specific problem-instance*, and (b) the FSM $\mathcal{F}$ that solves each instance of the problem is designed so that its state-memory "contains" the four integers a, b, c, d; i.e., $\mathcal{F} = \mathcal{F}^{(a,b,c,d)}$. The scalability in our problem solutions refers only to the mesh-size paramater n.

Every tile $v_{\varphi,\psi}$ can serve as an *anchor* to induce a partition of $\mathcal{M}_n$ into quadrants, as in Fig. 1.1b. The added challenge in [33] is to have the FSMs *sweep* the quadrants induced by $v_{\varphi,\psi}$ to check that each of the mesh's tiles contains a quadrant-specific label. In the third of our studies, [34], FSMs do more than plan application-specific trajectories and seek specified goal-tiles. They now rearrange objects that occupy $\mathcal{M}_n$'s tiles in various prespecified ways while transporting the objects from $\mathcal{M}_n$'s top row to its bottom row. The specific rearrangments include: (1) reversing the order in which objects appear, (2) cyclically rotating the objects, and (3) sorting the objects by their (ordered) "types." In addition to being scalable, the algorithms we describe in [34] are *pipelineable* in a way that achieves parallel speedup that is asymptotically linear in the number of FSMs (even as that number approaches n). The pipelining that we refer to here has a team of identical copies of an FSM $\mathcal{F}$ march one after the other, performing different instances of the chores to be performed; see, e.g., [34] for details.

The problems we discuss in this chapter describe, and in several places extend or improve, the material in [32–34]. The problems we discuss require FSMs:

- to determine where they are within $\mathcal{M}_n$;
  We focus on having FSMs determine which *wedge* of $\mathcal{M}_n$ they reside in (cf. Fig. 1.1c). (Recall that we treat the analogous problem for quadrants in [31].)
- to seek various target tiles of $\mathcal{M}_n$;
  We recapitulate the study in [33], wherein target tiles are specified via pairs of positive rational numbers, specifically using the rational pair $\langle \varphi, \psi \rangle$ to specify tile $\langle \lfloor \varphi(n-1) \rfloor, \lfloor \psi(n-1) \rfloor \rangle$ of $\mathcal{M}_n$.
- to transport the objects residing in $\mathcal{M}_n$'s top row to $\mathcal{M}_n$'s bottom row while rearranging the objects in prespecified ways;
  Excerpting from our study in [34], we have FSMs (1) reverse the objects' original order, (2) cyclically rotate the original order, and (3) sort the objects by their (ordered) types.
- to determine whether the objects residing in certain of $\mathcal{M}_n$'s rows of tiles have certain patterns.
  We complement the study in [34] by having FSMs check the pattern of objects along $\mathcal{M}_n$'s rows rather than effect the pattern. We have FSMs identify *palindromes* (words that read the same forwards and backwards), *perfect squares* (even-length words whose first and second halves are identical), and *rotations* (a pair of words one of which is a cyclic rotation of the other).

The algorithmic tools employed by our FSMs extend to myriad other problems.

A final word of introduction. We noted earlier that various sources—e.g., [11, 18, 35]—discuss "virtual pheromones" as a control mechanism for robotic "ants." This mechanism assigns registers within each robot's internal computer to maintain levels of intensity of an array of pheromones, thereby implementing a digital analogue of the volatile organic compounds that are used by nature's ants. We largely ignore "virtual pheromones" because FSMs do not need them to execute the algorithms we discuss. We note only that in [31] we have shown that "virtual pheromones" do not enhance the power of a single FSM—although they can sometimes be used to decrease the required size of an FSM, as measured in number of states.

1.2 Technical Background

1.2.1 FSM "Robots" and Their Domains

Our formal model of *FSM-robots* (*FSM*s, for short) is obtained by augmenting the capabilities of standard finite-state machines (sources such as [30] provide formal details) with the ability to travel around square *meshes* of *tiles*, possibly transporting *objects* from one tile to another (empty) one. We flesh out this informal description.

Meshes. We index the n^2 tiles of the $n \times n$ mesh $\mathcal{M}_n$ by the set[1] $[0, n-1] \times [0, n-1]$; see Fig. 1.1a. The set of tiles of $\mathcal{M}_n$ that share the first index-coordinate i, i.e., $R_i \stackrel{\text{def}}{=} \{\langle i, j\rangle \mid j \in [0, n-1]\}$, is the ith *row* of $\mathcal{M}_n$; the set of tiles that share the second index-coordinate j, i.e., $C_j \stackrel{\text{def}}{=} \{\langle i, j\rangle \mid i \in [0, n-1]\}$, is the jth *column* of $\mathcal{M}_n$. Tile $\langle i, j\rangle$ of $\mathcal{M}_n$ is:

- a *corner* tile if $i, j \in \{0, n-1\}$;
- an *(internal) edge* tile if it is one of:
  - a *bottom* tile Meaning that $i = 0$ and $j \in [1, n-2]$;
  - a *top* tile Meaning that $i = n-1$ and $j \in [1, n-2]$;
  - a *left* tile Meaning that $i \in [1, n-2]$ and $j = 0$;
  - a *right* tile Meaning that $i \in [1, n-2]$ and $j = n-1$;
- an *internal* tile if $i, j \in [1, n-2]$.

We employ the *King's move* adjacency model for meshes, so named for the chess piece (also known as the Moore model). Under this model, each tile $\langle i, j\rangle$ of $\mathcal{M}_n$ has up to 8 neighbors, one in each compass direction, abbreviated (in clockwise order) $N, NE, E, SE, S, SW, W, NW$. Accordingly, each internal tile of $\mathcal{M}_n$ has 8 neighbors; each (internal) edge tile has 5 neighbors; and each corner tile has 3 neighbors. Clerical modifications allow any *fixed finite* set of adjacencies, each specified by a pair of signed positive integers $\langle \pm a, \pm b\rangle$; each such pair, $\langle c, d\rangle$, indicates that every tile $\langle i, j\rangle$ of $\mathcal{M}_n$ has a neighbor at index-point $\langle i + c, j + d\rangle$, *as long as this point is a valid index for* $\mathcal{M}_n$, meaning that both $i + c$ and $j + d$ are in the range $[0, n-1]$. One opts for program compactness at the cost of algorithmic efficiency by choosing a smaller repertoire of adjacencies, such as *NEWS* moves: N, E, W, S (which are also known as the von Neumann model); one opts for increased efficiency at the cost of larger programs by choosing a larger repertoire of adjacencies, such as the 16 *Knight's + King's* moves. These three alternatives are illustrated in Fig. 1.2, which depicts the world from the viewpoint of an FSM. Whichever adjacency model is implemented: *every edge of every tile v of $\mathcal{M}_n$ is labeled to indicate which of v's potential neighbors actually exist.* (This enables FSMs to avoid "falling off" $\mathcal{M}_n$ or "banging into a wall.")

$\mathcal{M}_n$'s four *quadrants* are determined by lines that cross at an *anchor* tile v and are perpendicular to $\mathcal{M}_n$'s edges (Fig. 1.1b). The "standard" quadrants—which are anchored at $\mathcal{M}_n$'s "center" tile $\langle \lfloor \frac{1}{2}(n-1) \rfloor, \lfloor \frac{1}{2}(n-1) \rfloor \rangle$, hence are as close to equal in number of tiles as the parity of n allows—comprise the following sets of tiles.

[1] For positive integers i and $j > i$, we denote by $[i, j]$ the set $\{i, i+1, \ldots, j\}$.

Quadrant	Name	Tile-set
SOUTHWEST	$\mathcal{Q}_{SW}$	$\{\langle x, y\rangle \mid x \geq \lfloor \frac{1}{2}(n-1) \rfloor;\ \ y \leq \lfloor \frac{1}{2}(n-1) \rfloor\}$
NORTHWEST	$\mathcal{Q}_{NW}$	$\{\langle x, y\rangle \mid x < \lfloor \frac{1}{2}(n-1) \rfloor;\ \ y \leq \lfloor \frac{1}{2}(n-1) \rfloor\}$
SOUTHEAST	$\mathcal{Q}_{SE}$	$\{\langle x, y\rangle \mid x \geq \lfloor \frac{1}{2}(n-1) \rfloor;\ \ y < \lfloor \frac{1}{2}(n-1) \rfloor\}$
NORTHEAST	$\mathcal{Q}_{NE}$	$\{\langle x, y\rangle \mid x < \lfloor \frac{1}{2}(n-1) \rfloor;\ \ y \leq \lfloor \frac{1}{2}(n-1) \rfloor\}$

$\mathcal{M}_n$'s four *wedges* are determined by passing lines with slopes ± 1 through $\mathcal{M}_n$'s "center" tile; see Fig. 1.1c. These lines come as close to connecting $\mathcal{M}_n$'s corners as the parity of n allows. $\mathcal{M}_n$'s wedges comprise the following sets of tiles.

Wedge	Name	Tile-set
NORTH	$\mathcal{W}_N$	$\{\langle x, y\rangle \mid [x \leq y] \text{ and } [x + y \leq n-1]\}$
SOUTH	$\mathcal{W}_S$	$\{\langle x, y\rangle \mid [x > y] \text{ and } [x + y \geq n]\}$
EAST	$\mathcal{W}_E$	$\{\langle x, y\rangle \mid [x \leq y] \text{ and } [x + y \geq n]\}$
WEST	$\mathcal{W}_W$	$\{\langle x, y\rangle \mid [x > y] \text{ and } [x + y \leq n-1]\}$

Rounding ensures that each tile has a unique home quadrant and home wedge.

Objects. Each tile v of $\mathcal{M}_n$ can be empty—i.e., v contains 0 FSMs and 0 objects—or it can hold at most one FSM and at most one object—i.e., v contains 0 FSMs and 1 object *or* 1 FSM and 0 objects *or* 1 FSM and 1 object. Each object has a *type* chosen from some *fixed finite ordered* set. Because the number of objects can be commensurate with n while the number of object-types must be fixed independent of n, perforce, many objects can have the same type.

FSMs. At any moment, an FSM $\mathcal{F}$ occupies a single tile of $\mathcal{M}_n$, possibly sharing that tile with an object but *not* with another FSM. At each step, $\mathcal{F}$ can move to any neighbor v' of its current tile v (cf. Fig. 1.2), providing that v' contains no other FSM. Additionally, if v' contains no object, the $\mathcal{F}$ can convey the object that resides on v (if there is one) to v'.

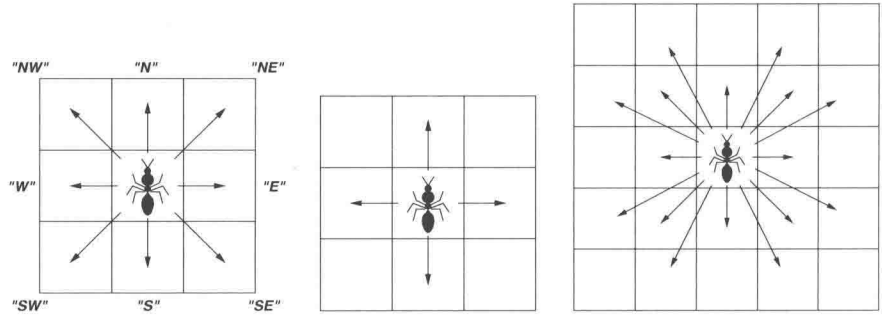

Fig. 1.2 Single-step move repertoires for FSMs. (*left*) The *King's-move* repertoire; (*center*) the *NEWS* (North-East-West-South) repertoire; (*right*) the *Knight's-move* + *King's-move* repertoire