Cellular ANTomata: A Tool for Early PDC Education

Arnold L. Rosenberg(✉)

Computer and Information Science, Northeastern University,
Boston, MA 02115, USA
rsnbrg@cs.umass.edu

Abstract. The thesis of this essay is that the *Cellular ANTomaton* (CAnt) computational model—obtained by deploying a team of mobile finite-state machines (the model's "Ants") upon a *cellular automaton* (CA)—can be a highly effective platform for introducing early undergraduate students to a broad range of concepts relating to *parallel and distributed computing* (*PDC*). CAnts permit many sophisticated PDC concepts to be taught within a unified, perspicuous model and then experimented with using the many easily accessed systems for simulating CAs and CAnts. Space restrictions limit us to supporting the thesis via only three important PDC concepts: synchronization, (algorithmic) scalability, and leader election (symmetry breaking). Having a single versatile pedagogical platform facilitates the goal of endowing *all* undergraduate students with a level of computational literacy adequate for success in an era characterized increasingly by ubiquitous parallel and/or distributed computing devices.

Keywords: Cellular automata and ANTomata
Teaching PDC to early undergrads

1 Introduction

1.1 Our Overall Goal

A. Computational literacy for all. The current era is characterized by ubiquitous computational devices. As such devices proliferate, they also become more sophisticated, containing multiple processors and/or cores. Indeed, we employ *parallel and distributed computing* (*PDC*) when we drive cars, use household appliances, go shopping, ... It is now widely recognized (cf. [15]) that *all* undergraduate students—all the more so those who aspire to a career in a computation-related field—must achieve a level of computational literacy adequate to succeed in our computing-rich society—and such literacy must encompass PDC and its enabling technologies. The thesis of this essay is that the *Cellular ANTomaton* (CAnt) computation model [18]—obtained by deploying a team of mobile finite-state machines (the model's "Ants") atop a *cellular automaton* (CA)—has traits that recommend it as a conceptual platform for introducing early undergraduate

© Springer International Publishing AG, part of Springer Nature 2018
D. B. Heras and L. Bougé (Eds.): Euro-Par 2017 Workshops, LNCS 10659, pp. 252–265, 2018.
https://doi.org/10.1007/978-3-319-75178-8_21

students to a broad range of sophisticated notions relating to *PDC*. We support this thesis by discussing three sophisticated PDC concepts that CAnts render accessible to early students. With more space, we could easily expand this list.

Benefit #1 of CAnt-based Pedagogy. CAnts *provide a single perspicuous platform for many core PDC concepts. Thereby, students need not master a range of platforms as they strive to master a variety of core concepts.*

B. What is computational literacy? We define "computational literacy" via three main features. For a *core concept* as defined in, e.g., [15], we expect a student to provide a:

1. *precise definition*—to a degree of rigor commensurate with the student's level;
2. *rudimentary implementation*—on a "reasonably simplified" computing platform;
3. *rudimentary analysis of an implementation on a "reasonably simplified" platform.*

"Reasonable simplifications" include, e.g., assuming that a key constant is a power of 2 or a perfect square or assuming that CAnts' constituent agents[1] act (perfectly) synchronously, rather than only approximately synchronously (cf. [5,22]).

1.2 Illustrating CAnt-Pedagogy via Some "Core" Concepts

A. The illustrative core concepts. We defend our pedagogical thesis by discussing "reasonably simplified" versions of three core PDC concepts. We chose these concepts because they invoke different strengths and features of the CAnt model.

1. *Synchronization.* The *Firing Squad Synchronization Protocol* (*FSSP*)—see [8,13]—allows the agents of arbitrarily large CAnts to initiate a process at the same step. A "reasonably simplified" FSSP should be accessible to early students.
   Enrichment opportunity #1. *A more advanced discussion could also address synchronicity—how to control* clock skew [5,22] *so that agents in neighboring cells "hear" temporally proximate clock signals almost simultaneously.*
2. *Scalability.* For many problems, one can craft a single algorithm that works on CAnts having arbitrarily many agents.
3. *Symmetry breaking (leader election).* Initiating concurrent procedures is more challenging for *distributed* agents than for *parallel* ones. CAnts admit a simple, efficient "leader-election" protocol for their (distributed) Ants.

With more space we could easily expand this list. As but two examples, the mesh structure underlying CAs and CAnts provides access to the following important topics. (*a*) The observation that many genres of computation can

[1] "Agents" comprise the *parallel* FSMs within a CA $\mathcal{C}$ and the *distributed* Ants atop $\mathcal{C}$.

be orchestrated as waves of data that pass though an array of identical computing agents spawned the elegant notion of *systolic array*: a highly structured form of *data flow* [11]. This topic has since advanced along several fronts [1,16]. (*b*) The advent of *massively* parallel computers via fragile VLSI-based technology heightened awareness of the importance of *fault tolerance*. Elegant and effective schemes have been developed for tolerating both faults and failures in mesh-based systems; cf. [7,10]. The details within the five just-cited sources are too sophisticated for beginning students, but the underlying ideas are readily accessible.

B. Our goal. We strive to help instructors appeal to a range of students, from the nonspecialist to the aspiring professional, as they teach our illustrative PDC-related concepts.

- Striving to serve the entire target range of students, we discuss *synchronization* in Sect. 3 via a verbally described synchronization algorithm, together with a small simulation of the algorithm and a simplified timing analysis.
- We discuss *scalability* in Sect. 4 via two examples. One is treated via an elementary verbally described algorithm. The other accompanies a verbally described algorithm with a program in pseudo-code and a proof of validity.
- We discuss *symmetry breaking/leader election* in Sect. 5 via a verbally described algorithm accompanied by a proof of validity and timing analysis.

1.3 Platforms for Implementing PDC Concepts

Implementing concepts helps students assimilate often-subtle details. The following tools provide quite distinct "programming" styles for simulating CAnts.

- *NETLOGO* [14] employs a rather general agent-based approach.
- *CARPET* [21] specifies Agents via *case-statement* programs.
- *MATLAB*® processes array-structured data using declarative programming.

Additionally, certain *systolic* computations can be specified perspicuously; cf. [1,16].

Benefit #2 of CAnt-based Pedagogy. *Several well-developed tools enable students to craft implementations of core concepts and experiment with them.*

1.4 A (Very) Brief History of CAs and CAnts

CAs have been studied since at least the 1960s [13] and continue to be of interest to this day [6,8]. They provide an attractive alternative to other formal models of computers [23], combining mathematical simplicity with levels of efficiency that make them feasible candidates for many real computational tasks. Indeed, CAs are remarkably efficient for a broad range of tasks that require the tight coordination of many simple agents [2,3,8,13]. In [2], CAs implement an ant-inspired clustering algorithm; in [3], they support an ant-inspired

algorithm for a genre of flow problem. In [12], a CA-like model greedily preplans a route for a single robot to a single goal, by having the goal broadcast its position. Several recent CA-based robotics-motivated studies appear in [20]. CAnts are introduced in [18], and algorithms are developed for some robotics-inspired problems. In contrast to CAnts, the models in the preceding sources support algorithms that are: • fully synchronous (there is a single clock that is "heard" by all agents); • centrally controlled (there is a central planner); • not scalable (the central planner knows and exploits the size of the system). Some models are centrally programmable, using systems such as CARPET [21]; their global name spaces preclude scalability. CAs have also been used for rather general suites of parallel-computing applications in [21] and related sources. Algorithms for (bio-inspired) pattern matching appear in [9] for one-dimensional CAs and in [19] for (two-dimensional) CAnts.

2 A Technical Introduction to CAs and CAnts

2.1 Overview of the Models

A *cellular automaton* (CA) is obtained by placing a copy of a single *finite-state machine* (*FSM*) at each cell of a *square mesh*. A *Cellular ANTomaton* (CAnt) is obtained by deploying a team of mobile FSMs (*Ants*) atop a CA, at most one Ant per cell. Each FSM communicates at each step with the FSMs within cells that are adjacent along the eight compass directions ($E, SE, S, SW, W, NW, N, NE$); it also communicates with an Ant that resides in its cell. Ants communicate with their host FSM and with any Ants that reside in adjacent cells. FSMs detect when the mesh-cell they reside in is on an edge or at a corner; thereby, a CAnt can ensure that Ants never "fall off" the mesh. The preceding informal definition will suffice for many styles of early introductory course. For other styles, one could add detail and formalism, as found in, e.g., [17,18].

Enrichment Opportunity #2. *One can distinguish Ants as physical devices (say, robots) or as virtual algorithmic devices (which can simplify subprocessing). In the former case, one could discuss inter-cellular message-transmission speeds: the electronic propagation of signals vs. the electro-mechanical movement of Ants.*

2.2 Pedagogically Useful Details

A. (Orthant) Meshes. We enable teaching opportunities by building the world of CAs and CAnts atop the (infinite) 2-*dimensional orthant mesh*[2], whose *cells* are labeled by all nonnegative integer-pairs, $\{\langle i,j\rangle \mid i,j \geq 0\}$. Each mesh-cell $\langle i,j\rangle$ has $\leq$8 types of *neighbors* (or, *adjacencies*), corresponding to the 8 compass directions; see Fig. 1(left).

[2] Our 2-*dimensional* (orthant) mesh is easily restricted to one dimension or extended to three.

Fig. 1. A "prefix" of: (left) a mesh $\mathcal{M}_n$; (center) a cellular automaton [CA] whose cells contain copies of an FSM F; (right) a Cellular ANTomaton [CANT] with three Ants.

B. Finite-State Machines. As their name suggests, finite-state machines (*FSM*s) were historically viewed as abstract machines (such as, say, elevators) whose behavior could be described and analyzed by characterizing "states" in which all "interesting" actions occurred. Myriad texts (e.g., [17]) adopt this view of FSMs. When teaching introductory computer science courses, though, students may be more receptive to viewing (and experimenting with) FSMs specified as *programs of case statements*, of the form indicated in Fig. 2.[3] One can simulate the operation of the FSM specified by such a program by iteratively cycling through the specified conditions until one finds one that applies.

$\underline{\text{LABEL}_1}$:	**if** INPUT_1 **then** $\text{OUTPUT}_{1,1}$ **and goto** $\text{LABEL}_{1,1}$
	⋮
	if INPUT_m **then** $\text{OUTPUT}_{1,m}$ **and goto** $\text{LABEL}_{1,m}$
LABEL_2:	**if** INPUT_1 **then** $\text{OUTPUT}_{2,1}$ **and goto** $\text{LABEL}_{2,1}$
	⋮
	if INPUT_m **then** $\text{OUTPUT}_{2,m}$ **and goto** $\text{LABEL}_{2,m}$
⋮	⋮
LABEL_s:	**if** INPUT_1 **then** $\text{OUTPUT}_{s,1}$ **and goto** $\text{LABEL}_{s,1}$
	⋮
	if INPUT_m **then** $\text{OUTPUT}_{s,m}$ **and goto** $\text{LABEL}_{s,m}$

Fig. 2. A finite-state machine (FSM) F specified via a program of case statements.

C. CA*s and* CANT*s.* One turns a mesh $\mathcal{M}$ into a CA $\mathcal{C}$ as follows.

- • Populate $\mathcal{M}$'s cells with copies of a single FSM F, one per cell (Fig. 1 (center)); we refer to the FSM at cell $\langle i, j\rangle$ as $\mathsf{F}^{\langle i,j\rangle}$.
  • Endow FSMs with bidirectional communication channels to FSMs in neighboring cells and to resident Ants (when they exist).

[3] The CARPET programming environment [21] employs a similar programming style.

- • Deploy $c \geq 0$ Ants on $\mathcal{M}$, at most one Ant per cell.
  • Endow each Ant $\mathcal{A}$ with bidirectional communication channels to Ants in neighboring cells and to the FSM in the cell that $\mathcal{A}$ is standing on.
- Endow FSMs and Ants with sensors: FSMs sense a resident Ant; FSMs and Ants sense mesh-edges, obstacles, and goal-objects (when relevant).

At each step: each copy of F polls the states of FSMs in neighboring cells and of any Ant that resides on F's cell; each Ant $\mathcal{A}$ polls the state of the FSM in its current cell plus the states of Ants on neighboring cells. Based on these polls, FSMs and Ants performs actions such as sending signals (an FSM may, e.g., tell its resident Ant to move). FSMs and Ants then change state—and the cycle repeats.

3 Synchronization

Synchronization in parallel/distributed systems seems at first blush to be an advanced topic that requires substantial background. In fact, for CAs and CANTS, the topic can be taught with varying levels of rigor to students having varying levels of preparation.

3.1 FSSP: The Firing Squad Synchronization Problem

We describe an algorithm for synchronizing Agents within CANTS, in a way that can "unfold" through a series of courses in the CS/CE curriculum, from a CS0-type course (e.g., "Computer Literacy") through a course in algorithm design/analysis.

- The motivation for and definition of the colorfully named *Firing Squad Synchronization Problem* (*FSSP*, for short) should be accessible to students even in a CS0-level course. This can whet students' appetites for more advanced courses by exposing them to a problem that is both interesting and non-"programmy."
- The solution to the FSSP sketched here should be accessible to students in any course that introduces recursion (as an algorithmic control structure). Students can observe a sophisticated recursion within a "reasonably simplified" framework, solving a problem that some students will initially doubt can even be solved.
- Our "simplified" analysis of the FSSP should be accessible to students whose algorithmic preparation includes the Master Theorem for Linear Recurrences [4].

The FSSP can be specified informally as follows. Start with n identical autonomous Agents standing (*physically or logically*) contiguously along row 0 of a mesh. Each Agent can communicate *only* with its immediate neighbors. (The two end Agents have one neighbor each; all others have two neighbors, one on each side.) The initially *dormant* Agents must enter an *active* state *at the exact*

same step when told to do so by the leftmost ("leader") Agent. The Agents' only tool for accomplishing the task is their limited ability to intercommunicate.

Solutions to the *one-dimensional* FSSP (the version just described) have been known since at least 1962 [13]. Easily, any solution requires at least $2n-2$ steps, just so a message can reach the farthest Agent and this Agent can respond to the leader. *There exist solutions that use only this number of steps—in fact, using only* 1*-bit inter-Agent messages* [8]. Surprisingly, any solution to the one-dimensional FSSP can be converted to a solution for any k-dimensional FSSP *that operates in exactly the same number of steps.* (**Note:** *This is actually a readily accessible exercise for even early students.*)

3.2 A Simplified Solution to the FSSP

We sketch a recursive algorithm for the FSSP that operates in roughly $3n$ steps instead of the optimal $2n-2$ steps. We then provide a "simplified" analysis that avoids floors and ceilings. This algorithm and analysis should be accessible to students at many levels. Beginning students should "get the basic idea"; students who have the basics of recursion and linear recurrences should understand the "simplified" details; really clever students should be able to build on this setting to obtain an improved solution.

A. A simple recursive solution. The solution has each Agent send messages of two types to its neighbors. These messages do not individually instigate actions; it is the *co-arrival* of messages of distinct types that triggers actions, as will become clear. Our verbal sketch ignores certain details that complicate the "end game" of the FSSP; these details do appear (beginning at step 15) in our illustration of the process in Fig. 3.

1. The initial stage. The leader Agent, ($\boxed{▭}$ in the figure) initiates the process by sending two messages, m_1 (• in the figure) and m_2 (∘ in the figure), to its eastward neighbor. Message m_1 is sent immediately; message m_2 is sent at step 3.

– Message m_1 travels at the rate of one Agent per step. It is relayed from each receiving Agent to its eastward neighbor until it reaches the end of the line of Agents, at which point it begins to travel westward at the same rate. (On this return trip, each receiving Agent relays m_1 to its westward neighbor.)
– Message m_2 travels at the rate of one Agent every third step. It also is relayed from each receiving Agent to its eastward neighbor.

At some point, messages m_1 and m_2 meet, i.e., arrive simultaneously, at some Agent $\mathcal{A}_i$. At this point, $\mathcal{A}_i$ becomes a *subleader* (▭ in the figure).

2. The inductive stage. Every newly anointed subleader recursively initiates the described process simultaneously and independently into the half-line of Agents to its left and into the half-line of Agents to its right. During these recursive invocations: (a) references to "left" and "right" are adjusted in the obvious way; (b) a (sub)leader encountered by a message in transit plays the same role as an end of the line.

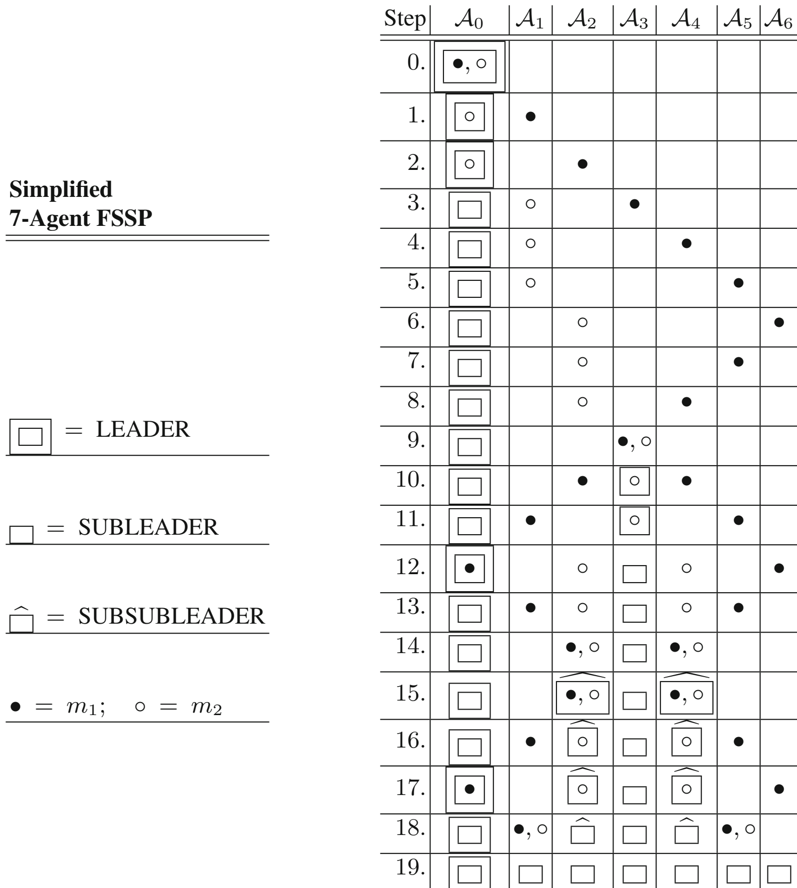


Fig. 3. The FSP synchronization protocol illustrated for seven Agents

3. Terminating the process. The process terminates when an Agent learns that both of its neighbors are subleaders—which will occur at the same step for all Agents. Figure 3 illustrates the sketched procedure for seven Agents. Note in Fig. 3 that the detailed algorithm suffers additional complication during the "end game" of a synchronization, to accommodate the (unknown) number n. To wit, from step 15 in the figure onward, we employ *sub-subleaders* ($\widehat{\square}$ in the figure) to prevent a subleader from activating too soon. Note also that the rightmost Agent ($\mathcal{A}_6$ in the figure) acts differently from other Agents. This does not mean that $\mathcal{A}_6$ differs structurally, only that the absence of a righthand neighbor modifies its behavior—specifically, with respect to termination.

B. Analyzing the recursion. We verify that the process terminates for all Agents at the same step by showing that messages m_1 and m_2 meet during the initial

stage at the midpoint of the line of Agents. (The analysis then recurses down to quarter-points, eighth-points, etc.) To see this: Say that m_1 and m_2 meet when t steps have passed since the initiation of the process. Ignoring floors and ceilings, during this time:

(a) message m_2 travels $t/3$ steps eastward;

(b) message m_1 travels n steps eastward then $x = t - n$ steps westward.

Clearly, m_2 has traveled from the leader to $\mathcal{A}_{t/3}$, while m_1 has traveled from the leader to $\mathcal{A}_{n-x}$, where $n - x = 2n - t$. But m_1 and m_2 *meet* at this time, so $t/3 = 2n - t$ or, equivalently, $t = \frac{3}{2}n$. This analysis verifies the algorithm's validity and also allows us to estimate the number of steps, $T(n)$ needed to synchronize n Agents:

$$T(n) \;=\; \frac{3}{2}n + T\left(\frac{1}{2}n\right) \;=\; \frac{3}{2}\left(1 + \frac{1}{2} + \frac{1}{4} + \cdots\right)n \;=\; 3n - \frac{3}{2}.$$

Our recursive procedure thus allows the n Agents to synchronize within $3n$ steps.

4 (Algorithmic) Scalability in CANTs

Our algorithm for the FSSP never refers to the number of Agents being synchronized; instead it uses the positions of the "leader" and the rightmost Agent as the delimiters of the messages that enable the synchronization. It is this feature—the fact that *a single algorithm works for* CANT*s of arbitrary sizes*—that we identify as *(algorithmic) scalability.* Of course, there are other valuable notions of scalability in PDC, but ours has advantages: (a) It requires no background beyond basic definitions. (b) It can be accessible to beginning students. (c) It can engage the students by requiring some thought to achieve. We present two computational problems that illustrate these advantages.

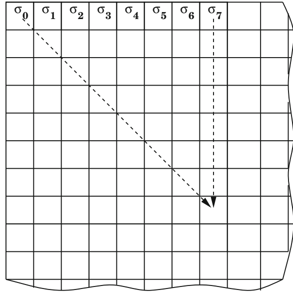
4.1 Example #1: Scalably Creating Square Meshes from Orthant Meshes

"Natural" computational problems for a CANT $\mathcal{C}$ usually operate within a *(finite) square* mesh, rather than the semi-infinite orthant mesh. For many such problems—specifically those that supply an input to $\mathcal{C}$ in the form of a length-n pattern $\sigma_0 \cdots \sigma_{n-1}$ left-justified along mesh-row 0—the following simple—and *scalable*—process converts the orthant mesh to a square mesh that is "natural" for the problem. See Fig. 4.

1. Simultaneously (via an FSP-synch, i.e., a synchronization using the FSSP):

   (a) $\mathcal{A}^{\langle 0,0\rangle}$ sends a southeasterly signal, which is propagated toward the southeast, i.e., toward cell $\langle n-1, n-1\rangle$;

   (b) $\mathcal{A}^{\langle 0,n-1\rangle}$—which knows its identity because its easterly neighbor contains no σ-symbol—sends a signal that is propagated southward (toward $\langle n-1, n-1\rangle$).



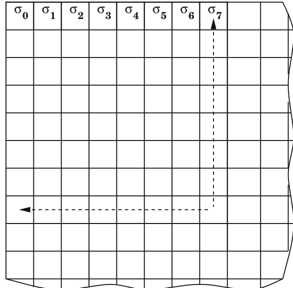


Fig. 4. Using an input pattern to delimit a square mesh from an orthant mesh: (left) measuring the square; (right) establishing the square mesh's eastern and southern boundaries.

Because CAnts operate synchronously (one of our "reasonable simplifications"), the signals from $\mathcal{A}^{\langle 0,0\rangle}$ and $\mathcal{A}^{\langle 0,n-1\rangle}$ arrive simultaneously at $\langle n-1, n-1\rangle$.

2. When $\mathcal{A}^{\langle n-1,n-1\rangle}$ receives the signals from $\mathcal{A}^{\langle 0,0\rangle}$ and $\mathcal{A}^{\langle 0,n-1\rangle}$, it simultaneously:
   (a) sends a message YOU ARE A BOTTOM CELL westward;
   (b) sends a message YOU ARE A RIGHT-EDGE CELL northward;
   (c) initiates an FSP-synch among all cells to its northwest.

After this $O(n)$-step process:

- the cells $\{\langle i,j\rangle \mid 0 \le i,j \le n-1\}$, the copies of $\mathcal{A}$ within these cells, and the Ants residing on these cells can function as an $n \times n$ CAnt $\mathcal{C}_n$;
- the cells $\{\langle i,j\rangle \mid [0 \le i \le n-1], [j = n-1]\}$ function as the "right edge" of $\mathcal{C}_n$;
- the cells $\{\langle i,j\rangle \mid [i = n-1], [0 \le j \le n-1]\}$ function as the "bottom row" of $\mathcal{C}_n$.

4.2 A Scalable Pattern-Reversing CAnt

The *Pattern-Reversal Problem* on an $n \times n$ mesh begins with an n-symbol input pattern $\Pi = \sigma_0 \cdots \sigma_{n-1}$ along row 0. The challenge is to design a CAnt $\mathcal{C}$ that copies Π along row $n-1$ *in reversed order*. Our CAnt $\mathcal{C}$ employs n identical *virtual* Ants, $\mathcal{A}_0, \ldots, \mathcal{A}_{n-1}$, with each $\mathcal{A}_k$ deployed initially on cell $\langle 0, k\rangle$. Figure 5 sketches a program that is shared by all Ants $\mathcal{A}$. The sketch is easily expanded to a formal program as in Fig. 2—*that nowhere mentions the length n of pattern Π.*

Figure 6 depicts the n-step (not counting the initiating FSP-synch) "multi-trajectory" for $\mathcal{C}$ mandated by the program of Fig. 5. To validate $\mathcal{C}$'s solution, focus on an Ant $\mathcal{A}_r$ that begins at a cell $\langle 0, r\rangle$. When $\mathcal{A}_r$ takes a *southwesterly* (resp., *southeasterly*) step, this adds $\langle +1, -1\rangle$ (resp., $\langle +1, +1\rangle$) to $\mathcal{A}_r$'s current cell's coordinates. It follows that, under the dogleg patterns of Fig. 6, $\mathcal{A}_r$'s trajectory consists of:

At each step:	
Case	**Action**
$\mathcal{A}$ on $\mathcal{M}_n$'s left edge	- delays one step - then moves symbol one cell *southeastward*
$\mathcal{A}$ moving southeastward	continues toward $\mathcal{M}_n$'s bottom edge
$\mathcal{A}$ on $\mathcal{M}_n$'s bottom edge	deposits its symbol and **halt**s
Otherwise	$\mathcal{A}$ moves symbol one cell southwestward

Fig. 5. A sketch of a program for one of $\mathcal{C}$'s (identical) pattern-reversing Ants $\mathcal{A}$, as it: (1) *picks up* the symbol in its initial cell c; (2) *conveys* the symbol, via a SW-then-SE path, to c's "mirror" bottom-edge cell $\bar{c}$; (3) *deposits* the conveyed symbol in cell $\bar{c}$; (4) *halts*.

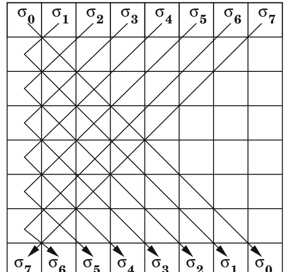


Fig. 6. CANT $\mathcal{C}$'s trajectory as it copies the pattern along row 0 *in reversed order* along row $n-1$.

1. an r-step *southwesterly* walk from cell $\langle 0, r\rangle$ to cell $\langle r, 0\rangle$;
2. an $(n-r-1)$-step *southeasterly* walk from cell $\langle r, 0\rangle$ to cell $\langle n-1, n-r-1\rangle$.

The fact that cell $\langle n-1, n-r-1\rangle$ is the "mirror image" along row $n-1$ of cell $\langle 0, r\rangle$ completes the validation.

5 Leader Election/Symmetry Breaking

A central challenge in distributed computing is coordinating the actions of identical autonomous agents. An important approach to meeting this challenge is to "elect" one of the agents as a "leader," thereby "breaking" the "symmetry" caused by agents' being indistinguishable. Many leader-election protocols have been invented, all requiring algorithmic sophistication. When the distributed agents are Ants within a CANT $\mathcal{C}$, the underlying CA affords us a rather simple, efficient leader-election protocol. In particular, $\mathcal{C}$ selects as the "leader" the unique Ant (if any exist!) that is "closest" to the origin Agent, $\mathcal{A}^{\langle 0,0\rangle}$, in the following sense. For each Ant $\mathcal{A}$, we count the number of cells $\mathcal{A}$ needs to traverse in order to reach $\mathcal{A}^{\langle 0,0\rangle}$ via a path of northward moves (toward row 0) followed by a path of westward moves (toward column 0), *under a regimen that gives a westward moving Ant priority over northward moving one.* (The latter clause resolves ties when two Ants compete to enter the same row-0 cell.)

5.1 The Leader-Election Process

(a) $\mathcal{A}^{\langle 0,0\rangle}$ initiates the process by simultaneously sending two messages:

1. an FSP-synch to start the process for any Ants that exist within $\mathcal{M}_n$;
2. an eastward-bound message, NO ANT YET.
   - This message is relayed along row 0 up to $\mathcal{M}_n$'s eastern edge, whence it is bounced back toward $\mathcal{A}^{\langle 0,0\rangle}$.
   - If the message reaches an Agent $\mathcal{A}$ that knows of an Ant—from receiving Ant-related messages—then $\mathcal{A}$ "swallows" this message by not relaying it.

   Note that if $\mathcal{A}^{\langle 0,0\rangle}$ receives the bounced-back message, *and* it has not received an Ant-related message, then it knows that no Ant resides on $\mathcal{M}_n$.

(b) When "activated" (via the FSP-synch), each cell that contains an Ant sends the message I HAVE AN ANT northward, toward row 0.
(c) While a row-0 Agent $\mathcal{A}^{\langle 0,k\rangle}$ is *active:*

- The *first time* it receives the message I HAVE AN ANT from its southern neighbor, $\mathcal{A}^{\langle 1,k\rangle}$, it sends the message ANT IN MY COLUMN westward, toward cell $\langle 0,0\rangle$.
- If it receives the message ANT IN MY COLUMN from its eastern neighbor, $\mathcal{A}^{\langle 0,k+1\rangle}$, then it relays that message westward, toward cell $\langle 0,0\rangle$.

In both cases, $\mathcal{A}^{\langle 0,k\rangle}$ *then becomes inactive.*
(d) While a row-0 Agent $\mathcal{A}^{\langle 0,k\rangle}$ is *inactive,* it ignores all messages from its eastern and southern neighbors.
(e) $\mathcal{A}^{\langle 0,0\rangle}$ learns about the presence or absence of Ants in one of three ways.

- If $\mathcal{A}^{\langle 0,0\rangle}$ receives the message NO ANT YET from its eastern neighbor, $\mathcal{A}^{\langle 0,1\rangle}$, then it knows that no Ant resides on $\mathcal{M}_n$.
  In response, $\mathcal{A}^{\langle 0,0\rangle}$ broadcasts NO ANTS FOUND eastward and southward.
- • If the first message that $\mathcal{A}^{\langle 0,0\rangle}$ receives is I HAVE AN ANT from its southern neighbor, $\mathcal{A}^{\langle 1,0\rangle}$, then a leader-Ant has been discovered.
  • The first time $\mathcal{A}^{\langle 0,0\rangle}$ receives ANT IN MY COLUMN from its eastern neighbor, $\mathcal{A}^{\langle 0,1\rangle}$, it knows that a leader-Ant has been discovered.

When either occurs, $\mathcal{A}^{\langle 0,0\rangle}$ broadcasts LEADER ANT FOUND eastward and southward. It also transmits YOU ARE THE LEADER in the direction from which it received the Ant-related message. This "congratulatory message" is relayed back to the originating Ant by intermediate Agents.

In parallel with its broadcast, $\mathcal{A}^{\langle 0,0\rangle}$ initiates an FSP-synch to terminate the procedure.
(f) When row-0 Agent $\mathcal{A}^{\langle 0,k\rangle}$ receives LEADER ANT FOUND from its western neighbor, $\mathcal{A}^{\langle 0,k-1\rangle}$, it relays the message eastward to $\mathcal{A}^{\langle 0,k+1\rangle}$ and southward to $\mathcal{A}^{\langle 1,k\rangle}$.

5.2 Analyzing the Leader-Election Process

A. Validation. The correctness of the process follows from the observations that: (a) If there is an Ant upon $\mathcal{M}_n$, then $\mathcal{A}^{\langle 0,0\rangle}$ receives precisely one message I HAVE AN ANT—and that comes from an Ant that is closest to $\mathcal{A}^{\langle 0,0\rangle}$. Competing messages are swallowed by intervening Agents. The message NO ANT YET tells $\mathcal{A}^{\langle 0,0\rangle}$ there is no resident Ant. Thus, the leader-election process always halts, with a closest leader Ant if one exists.

B. Timing. The leader-election process completes within $4n$ steps on an $n \times n$ CANT:

- $\exists$ *Ant on* $\mathcal{M}_n$. Then $\mathcal{A}^{\langle 0,0\rangle}$ receives the message I HAVE AN ANT within $2n$ steps.
- $\nexists$ *Ant on* $\mathcal{M}_n$. Then $\mathcal{A}^{\langle 0,0\rangle}$ receives the message NO ANT YET within $2n$ steps.

Within an additional $2n$ steps, $\mathcal{A}^{\langle 0,0\rangle}$ initiates an FSP-synch that both terminates the process and announces either the election of a leader or the absence of an Ant. In parallel, $\mathcal{A}^{\langle 0,0\rangle}$ sends a "congratulatory message" to the new leader.

6 Conclusion

Many PDC-related concepts that are quite sophisticated in general settings have rather simple versions within the Cellular ANTomaton (CANT) model. An instructor can use CANTs to gently introduce such problems to students who have only basic knowledge about topics such as linear recurrences, asymptotics, and Agents. When a student encounters the sophisticated versions of the problems later, s/he has intuitions from the CANT-based simplifications. Additionally, these intuitions can be strengthened using the many convenient tools such as *NETLOGO* [14], *CARPET* [21] and *MATLAB*®. It would be exciting to try this approach with a range of classes, beginning even with CS0.

References

1. Avis, D., Bremmer, D., Deza, A. (eds.): Polyhedral computation. In: CRM Proceedings and Lecture Notes, vol. 48. American Mathematical Society (2009)
2. Chen, L., Xu, X., Chen, Y., He, P.: A novel ant clustering algorithm based on cellular automata. In: IEEE/WIC/ACM International Conference, Intelligent Agent Technology (2004)
3. Chowdhury, D., Guttal, V., Nishinari, K., Schadschneider, A.: A cellular-automata model of flow in ant trails: non-monotonic variation of speed with density. J. Phys. A: Math. Gen. **35**, L573–L577 (2002)
4. Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C.: Introduction to Algorithms, 2nd edn. MIT Press, Cambridge (1999)
5. Fisher, A.L., Kung, H.T.: Synchronizing large VLSI processor arrays. IEEE Trans. Comput. **C-34**, 734–740 (1985)

6. Goles, E., Martinez, S. (eds.): Cellular Automata and Complex Systems. Kluwer, Norwell (1999)
7. Greene, J.W., El Gamal, A.: Configuration of VLSI arrays in the presence of defects. J. ACM **31**, 694–717 (1984)
8. Gruska, J., La Torre, S., Parente, M.: Optimal time and communication solutions of firing squad synchronization problems on square arrays, toruses and rings. In: Calude, C.S., Calude, E., Dinneen, M.J. (eds.) DLT 2004. LNCS, vol. 3340, pp. 200–211. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-30550-7_17
9. Laurio, K., Linaker, F., Narayanan, A.: Regular biosequence pattern matching with cellular automata. Inf. Sci. **146**(1–4), 89–101 (2002)
10. Leighton, F.T., Leiserson, C.E.: Wafer-scale integration of systolic arrays. IEEE Trans. Comput. **C-34**, 448–461 (1985)
11. Leiserson, C.E.: Systolic and semisystolic design. In: IEEE International Conference on Computer Design, pp. 627–630 (1983)
12. Marchese, F.: Cellular automata in robot path planning. In: EUROBOT, pp. 116–125 (1996)
13. Moore, E.F: The firing squad synchronization problem. In: Moore, E.F. (ed.) Sequential Machines, Selected Papers, pp. 213–214. Addison-Wesley, Boston (1962)
14. https://ccl.northwestern.edu/netlogo/
15. Prasad, S.K., Gupta, A., Kant, K., Lumsdaine, A., Padua, D., Robert, Y., Rosenberg, A.L., Sussman, A., Weems, C.: Literacy for all in parallel and distributed computing: guidelines for an undergraduate core curriculum. CSI J. Comput. **1**(2), 10:81–10:95 (2012)
16. Quinton, P.: Automatic synthesis of systolic arrays from uniform recurrence equations. In: 11th IEEE International Symposium on Computer Architecture, pp. 208–214 (1984)
17. Rosenberg, A.L.: The Pillars of Computation Theory: State, Encoding Nondeterminism. Universitext Series. Springer, New York (2009). https://doi.org/10.1007/978-0-387-09639-1
18. Rosenberg, A.L.: Cellular ANTomata. Adv. Complex Syst. **15**(6) (2012)
19. Rosenberg, A.L.: Bio-inspired pattern processing by cellular ANTomata. J. Cell. Automata **13**(1–2), 53–80 (2018)
20. Sirakoulis, G.C., Adamatzky, A. (eds.): Robots and Lattice Automata. ECC, vol. 13. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-10924-4
21. Spezzano, G., Talia, D.: The CARPET programming environment for solving scientific problems on parallel computers. Parallel Distrib. Comput. Practices **1**, 49–61 (1998)
22. Williams, T.: Clock skew and other myths. In: IEEE International Symposium on Asynchronous Circuits and Systems (2003)
23. Wolfram, S. (ed.): Theory and Application of Cellular Automata. Addison-Wesley, Boston (1986)