Complex Systems **7** (1993) 297–307

Two-Dimensional FHP Lattice Gases Are Computation Universal

Richard K. Squier
Computer Science Department, Georgetown University, Washington, DC 20057, USA

Kenneth Steiglitz
Computer Science Department, Princeton University, Princeton, NJ 08544, USA

Abstract. We show that the FHP lattice gases are computation universal, implying that general questions about their behavior are undecidable. The proof embeds a universal one-dimensional cellular automaton in the two-dimensional FHP lattice gas. This provides evidence that general questions about fluid behavior are undecidable.

1. Introduction

Ever since people began asking quantitative questions about physical systems, they have also been asking whether there are shortcuts to finding the answers. When mathematics gives a closed-form solution for the state of the physical system that is easy to evaluate, the problem of getting the information is usually essentially solved. However, for some problems the best approach known involves numerical computation that requires astronomical numbers of operations, and the question naturally arises whether quantitative information can be found by some clever shortcut technique, or if the nature of the physical system makes this impossible [1, 2].

The FHP lattice gases [3] are extremely simple cellular automata (CAs) that can be used to solve the Navier-Stokes equations. Their simplicity might tempt us to hope that fluid behavior can be predicted without direct, step-by-step simulation. In this paper we show that the FHP lattice gases are computation universal, which implies that general questions about their behavior are undecidable. More precisely, the general question of whether a given lattice gas started from given initial conditions ever reaches a prescribed state is equivalent to the Halting Problem for Turing machines.

Our method of showing that FHP lattice gases are universal will be to embed a universal one-dimensional CA in a lattice gas initialized to a periodic quiescent background state and having moving particles that code the initial

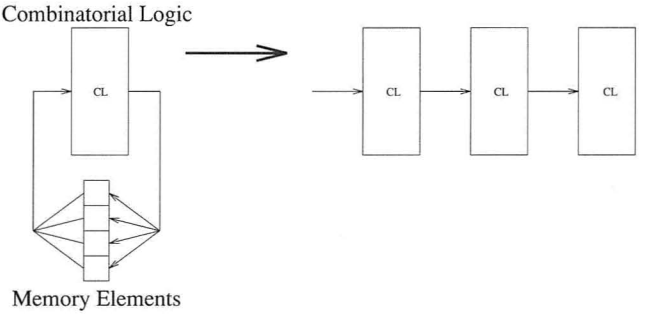

Figure 1: Unfolding a sequential logic machine to create an equivalent combinatorial one.

state of the one-dimensional CA. The periodic background state supports the evolution of the one-dimensional CA by propagating an image of the instantaneous state of the one-dimensional CA through the periodic background. The state of the one-dimensional CA at evolution time step t can be read off by observing the appropriate locations in the two-dimensional space of the lattice gas at a distance from the origin that is linearly related to t. The general principle of this embedding in the plane is simply that of converting a sequential logic device to a combinatorial one by "unfolding" (see Figure 1).

2. Lattice gas definitions

Our simulation of a universal one-dimensional CA will use only a subset of the rules of a lattice gas such as FHP-III [3] or LGM-1 [4], and therefore our result includes any two-dimensional lattice gas on the triangular lattice whose rule set includes the required subset of rules. The following definition characterizes these types of two-dimensional lattice gases.

Definition 1. *A lattice gas is FHP-like if its lattice graph is the two-dimensional triangular lattice with nearest-neighbor edges, and the rule set contains the reversible collision rules shown in Figure 2. (In Figure 2 a circle represents a rest particle, a square represents a barrier site, and an arrow represents a particle located at the arrow's tail moving with unit velocity in the direction of the arrow.)*

3. Universality

We will embed a general one-dimensional CA in an FHP-like lattice gas. Because there exist universal one-dimensional CAs, the FHP-like lattice gas is also universal. Each state variable of the one-dimensional CA is identified by a pair of lines parallel to the x-axis, the ith state variable s_i corresponding to the ith pair of lines counting from the origin. One line in each pair codes the value 0, the other the value 1, by the presence of a rightward-moving

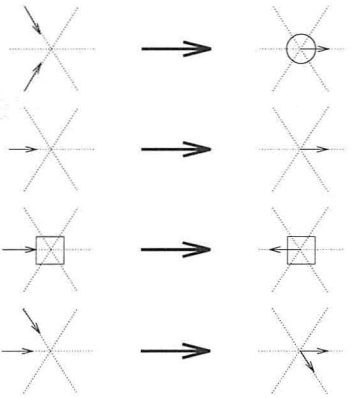

Figure 2: Required collision rules for the lattice gas. Also includes the reverse of the first rule.

particle at a specific x distance along that line from the origin, the distance corresponding to the CA time step t. For instance, $s_i^t = 0$ is coded by the presence of a rightward-moving particle on the 0-value line for s_i at x coordinate ct for a particular constant c. The status of these "state lines" at a given x value of ct can be thought of as representing the "current" one-dimensional CA at automaton time step t. The current machine sends the values of its state variables propagating through a boolean network that computes next-state values for each active cell in the next copy of the one-dimensional CA. This is a computation wavefront moving to the right, and the state of the one-dimensional machine is coded by the arrangement of particles at lattice sites on the wavefront. To the left of the wavefront the particles have no significance, and to the right of the wavefront there are no moving particles.

Theorem 1. *The two-dimensional FHP-like automata are universal.*

Proof. We begin by describing the embedding of a one-dimensional CA in the xy plane. Later we show the implementation of this embedding in the FHP lattice. There exists a universal 14-state one-dimensional CA with a neighborhood of $r = 1$ [5], and implementing this CA requires 4 bits of state information. We show the embedding of an arbitrary 2-bit one-dimensional CA with the same neighborhood; the extension to 4 bits will be obvious.

Suppose A is a one-dimensional CA with two 1-bit state variables per cell. Let the two state variables of cell i be a_i and b_i. Let $x(t)$, for $t \in \mathbf{Z}^+$ (the non-negative integers), be defined by

$$x(t) = \frac{\alpha}{\beta} t,$$

where α and β are positive integers we will choose later. The function $x(t)$ defines the distance from the origin in the positive x direction, where the

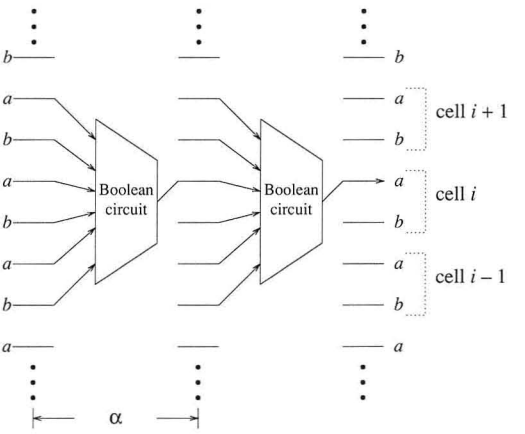

Figure 3: Three copies of the state wires for A and the boolean circuits for the next-state function for a_i.

values of the state variables of A can be read at time t. Let

$$T(t) = \lfloor t/\beta \rfloor$$

define the evolution time step for A. (Later, t will become the time step for the lattice gas.) Thus, the values of a_i^T and b_i^T can be read along the line parallel to the y axis at a distance $x(T) = \alpha T$ from the origin. That is, for every β time steps we look α units farther to the right to see the state of A.

Let the set of points $\{(x(T), i \pm \epsilon) \mid i \in \mathbf{Z}^+\}$ be called "state wires." Each state wire (x, y) represents a state variable of A at evolution step T: $(x(T), i + \epsilon)$ represents a_i^T, and $(x(T), i - \epsilon)$ represents b_i^T.

Between the set of state wires at $x(T)$ and the state wires at $x(T+1)$ we install a network of wires, fan-out devices, and logic gates that implement the update rules of A. That is, the input wire for a_i at $x(T+1)$ is the output of a boolean circuit that implements the update rule of A from the inputs $\{a_k^T, b_k^T \mid k = i, i \pm 1\}$, and similarly for b_i. The delay through this circuit is exactly α. Figure 3 shows three one-dimensional CAs laid out with boolean circuits for a_i between them. This completes the embedding of A in the xy plane, and we next take up the realization of this embedding in the lattice gas.

There are two main elements in the realization of these boolean circuits in the lattice-gas automaton: the implementation of wire fan-out and crossings, and the implementation of the logic gates and other devices. Here we show what is generally required of the wiring, and later show the specific implementation of a wiring scheme that satisfies these requirements. After that, we show how to implement the specific wiring and logic devices.

The general layout of the update circuit for a single state variable a_i between $x(T)$ and $x(T+1)$ is shown in Figure 4. The update circuit realizes the boolean update function f in its disjunctive normal form (that is, a

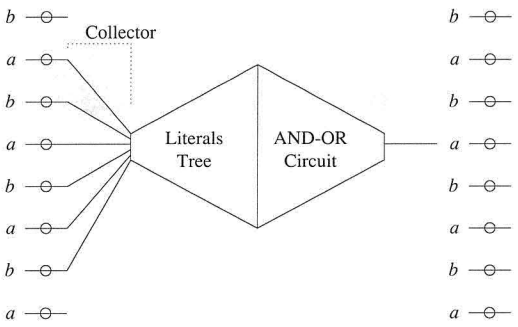

Figure 4: A collector, literals tree, and AND-OR circuit for a_i between $x(T)$ and $x(T+1)$.

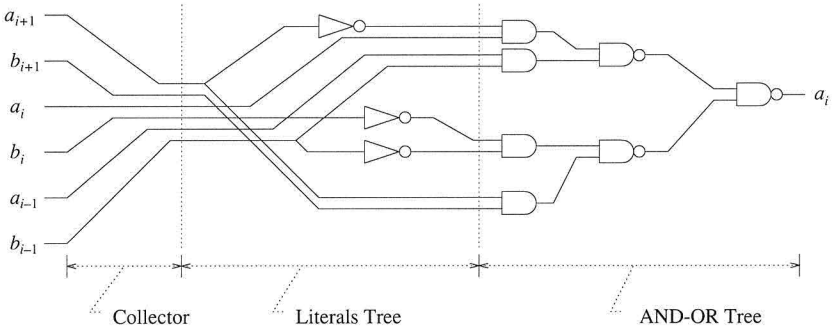

Figure 5: A collector, literals tree, and AND-OR circuit for some update function f.

logical OR of "minterms," each minterm being a logical AND of literals.) This circuit consists of three parts. First, wires from all state variables appearing in the expression for f are sent to a "collector." Next, a "literals tree" fans out the wires for the state variables in the collector to produce the literals for each minterm appearing in the expression for f. Finally, an AND-OR circuit for f uses these literals as input and sends its output to a_i^{T+1}.

Figure 5 shows a schematic example of an update circuit for $a_i^{T+1} = f(a_{i+j}^T, b_{i+j}^T)$, $j = 0, -1, +1$, using the logic and wiring devices we construct in the lattice gas. Generally the devices we have available are the following: two-input AND gates, NOT gates, horizontal wires, slope $\pm 1/\sqrt{3}$ wires, two-wire crossovers, and one-to-two fan-out devices. Each of these devices spans the same unit distance along the x axis, and each has the same signal delay. Since {NOT, AND} forms a complete logic family, we have sufficient logic resources to produce all literals and any AND-OR tree. All that remains is to show a sufficient wiring scheme for connecting these elements in the two-dimensional space of the lattice gas.

3.1 Low-level implementation of primitive devices

In our scheme a logical "wire" representing a single bit consists of two parallel paths in the lattice along which particles may travel. One path is considered the boolean "true" path, and the other is considered the "false" path. A particle will appear on one or the other of these paths at time T at x position $x(T)$, and the logic value of the wire is determined by which path the particle is on at that x position. Because of the conservation properties of the FHP lattice gases, our primitive gates and devices produce moving particles that are not part of the "valid" computational wavefront. These "garbage" particles could destroy the coding of the one-dimensional CA state bits, if they arrived at $x(T)$ simultaneously with or before the computational wavefront. We ensure that this interference does not occur by assuming that every lattice site not on a path contains a barrier (essentially we are insulating our wires). This ensures that garbage particles stay on the wire paths, and because they are initially deflected to the left in any wavefront interaction they are guaranteed to be at least one time step behind the wavefront. We now describe the primitive elements for constructing the basic wiring described above (fan-out, wires, and crossovers), and for constructing the complete logic family {NOT, and AND}.

3.1.1 Turns

The main device necessary for implementation of all our logic and wiring devices is the "turn." This device deflects a moving particle from its incident path by 60 degrees and was introduced in [6]. In this device a moving particle collides with a rest particle, resulting in two particles leaving the collision site—one at $+60$ and the other at -60 degrees from the direction of the incident particle. For clarity, the extra particle (a garbage particle) is "trapped" in the vicinity of the turn by a pair of barrier sites. This trapping is not essential to the construction, but makes the idea clearer. Figure 6 shows a turn device and the symbol used for it in subsequent figures.

3.1.2 Primitive devices

A horizontal path can be implemented as an unobstructed horizontal path in the lattice. The timing of particles arriving at the cell locations must be coordinated, however, so the horizontal path is formed from a delay line whose extent in the x direction and signal delay is the same as all other devices we will introduce. Figure 7 shows such a delay line. For the rest of the devices we will dispense with the detailed descriptions and show only the schematic representation.

The remainder of the required elementary devices are shown schematically in Figure 8. They are the fan-out, the crossover, the half-AND, and the half-XOR devices. After introducing these elementary devices we will use them to build AND and NOT gates.

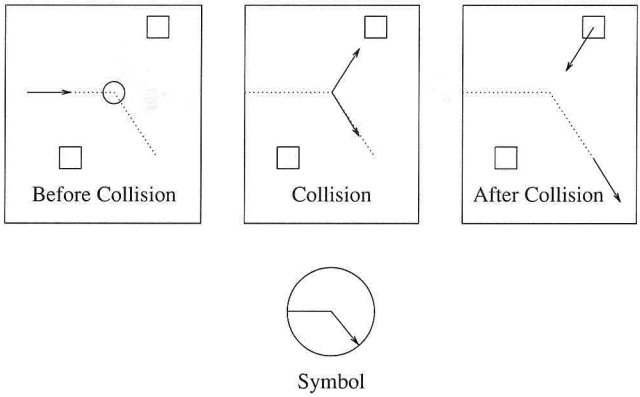

Figure 6: A "turn" device at three consecutive time steps. The small circle represents a rest particle and the small squares represent barrier sites.

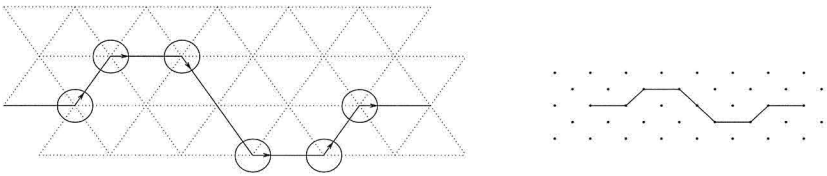

Figure 7: The delay line for a single path. The x extent is 6 lattice units, and the time delay is 8 lattice-gas evolution time steps. On the right the delay line is shown schematically.

3.1.3 The AND and NOT gates

The two primitive logic gates introduced in Figure 8, the half-AND and the half-XOR, give only partial results but can be combined to give a true AND function. Suppose the input paths to these devices are labeled "A = true" and "B = true." The half-AND gate gives as output a single particle moving right if its two inputs are asserted, and nothing otherwise; consequently its output would be labeled "AB = true," which is only one-half an AND gate since there is no "AB = false" output. A half-XOR having the same inputs would have an output labeled "$A \oplus B$ = true." Again, there is no false output. Figure 9 shows schematically the construction of the full AND gate with true/false inputs and outputs.

The last device we need is a NOT gate. This is implemented simply by switching the "true" path with the "false" path. (This is easily accomplished by shifting the true path to the false path using, for instance, the slope-line device.) The switch of the false path to the true path can be done at the same point in space since there will be only one particle on the wire.

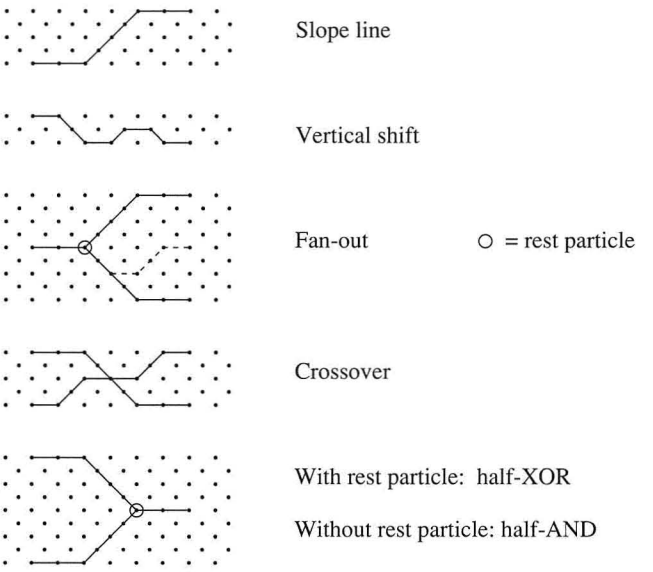

Figure 8: The primitive devices. Most are self-explanatory. The fan-out device shows an optional dashed path for one of the exit paths. This may be used if the fan-out is one-sided.

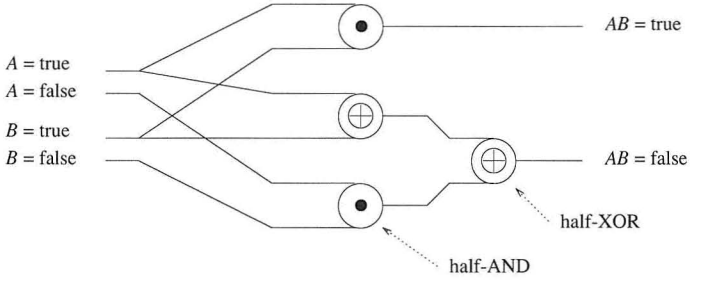

Figure 9: Schematic for the full AND gate.

3.2 Final layout

Figure 10 shows the method for fanning out a single bit. All bit wires not currently being fanned out are implemented as horizontal wires. The bit being fanned out is fanned out to the horizontal lines that lead directly to the destinations at the right end of the collector in Figure 4. Thus all wire crossings are two-wire crossings.

Using this general method for fanning out wires, we complete the layout of the collector/literals tree/AND-OR tree in such a way that we guarantee the connectivity and correctness of the entire layout as follows. The entire set of lines for every path of a single state variable is laid out on the triangular

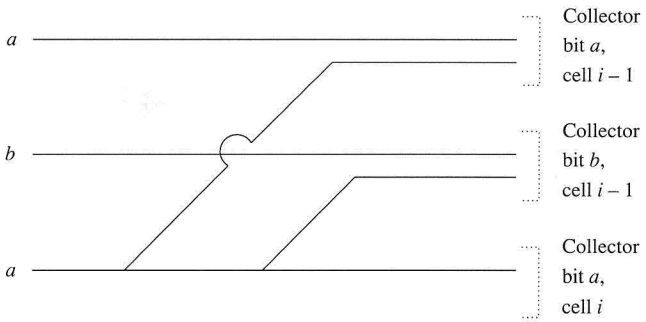

Figure 10: Fanning out a single bit.

lattice using lattice points as endpoints for every line segment. These lines are constrained to follow the connection and angular requirements of the primitive devices detailed above. This layout consists of (1) the collector lines for the neighborhood of the state variable, (2) the literals tree generating every possible minterm from these lines, and (3) the AND-OR tree using these minterms (the unused minterms are blocked by barrier sites). The entire structure, minus the collector, spans some vertical distance. The state wires are spaced vertically to keep structures for adjacent state wires from interfering with each other.

Next, we replace wire crossings, fan-out, and logic devices with the primitive devices. For convenience, we can restructure the devices to have common extents in basic lattice unit distances, say d_x in the x direction and d_y in the y direction. Placing the devices could result in two problems. First, distances along some lines might not be integer multiples of these basic device lengths, and second, devices may overlap. If the first problem occurs, we scale the entire structure by d_x in the x direction and d_y in the y direction so that every line can be covered by an integer number of devices. Next, if we still cannot place devices without overlap, we again rescale a second time, thereby guaranteeing that no devices overlap. This completes the layout and embedding in the lattice. The resulting layout is uniform and infinitely repeated vertically, forming a periodic structure.

4. Discussion

It is interesting to note a few features of the above construction. First, it depends on non-conservation of energy since fan-out is accomplished by accelerating a rest particle to unit velocity using a single particle traveling with unit velocity both before and after the collision. Second, it depends on non-conservation of momentum since the "turn" device uses collisions with a barrier of infinite mass. We have been unable to find an energy-conserving embedding of universal computation in the lattice gas similar to the billiard-ball machine of Margolus [7]. It remains an open question whether one exists.

What conclusions can we draw from the result of this paper about using the lattice gas to answer questions about fluid behavior? We cannot conclude that the initial conditions that result from embedding fluid-flow problems in the lattice gas result in unsolvable problems. However, it seems unlikely that all the decision problems about all the configurations that result from embedding fluid problems into the lattice gas are decidable. Thus, we should view the main result as providing evidence that predicting the future state of a fluid, at least one whose behavior is determined by the partial differential equations corresponding to the lattice gas, is undecidable. That is, general questions of the sort, Does a fluid with a given initial state ever exhibit some specific behavior? apparently cannot be answered by computation.

The same conclusion can be drawn from a completely different argument. Build a universal Turing machine using fluidic valves to implement the gates [8]. Then this "fluidic" computer can simulate any Turing machine, and therefore there are general questions about this computer's behavior that are undecidable. Since the fluidic computer's state is entirely determined by the fluid and its initial and boundary conditions, we reach the same conclusion as above.

We have just mentioned two distinct arguments, neither conclusive, that lead us to believe that, put informally, there are many questions about fluid behavior that are impossible to answer computationally. The argument based on lattice gases leaves open questions about the generality of the subset of initial conditions corresponding to fluid embeddings. It also assumes that lattice gases describe the behavior of fluids. The argument based on a fluidic computer assumes that such a machine can be built and its state measured.

5. Acknowledgments

We thank Neal Young for valuable discussions. This work was supported by NSF Grant MIP-9201484. Richard Squier was also supported by an Office of Naval Technology Postdoctoral Fellowship while at the Naval Research Laboratory, Washington, DC.

References

[1] S. Wolfram, "Undecidability and Intractability in Theoretical Physics," *Physical Review Letters*, **54** (1985) 735.

[2] R. P. Feynman, "Simulation of Physics with Computers," *International Journal of Theoretical Physics*, **21**(6/7) (1982) 467–488.

[3] U. Frisch, D. d'Humiéres, B. Hasslacher, P. Lallemand, Y. Pomeau, and J. P. Rivet, "Lattice Gas Hydrodynamics in Two and Three Dimensions," *Complex Systems*, **1** (1987) 649–707.

[4] S. D. Kugelmass and K. Steiglitz, "A Scalable Architecture for Lattice-Gas Simulations," *Journal of Computational Physics*, **84** (1989) 311–325.

[5] J. Albert and K. Culik II, "A Simple Universal Cellular Automation and its One-Way and Totalistic Version," *Complex Systems*, **1** (1987) 1–16.

[6] R. K. Squier and K. Steiglitz, "Testing Parallel Simulators for Two-Dimensional Lattice-Gas Automata," *Complex Systems*, **5** (1991) 63–68.

[7] N. Margolus, "Physics-like Models of Computations," *Physica D*, **10** (1984) 81–95.

[8] E. C. Fitch and F. B. Surjaatmadja, *Introduction to Fluid Logic* (Washington: Hemisphere, 1978).