Discrete Modelling and Discrete Algorithms in Continuum Mechanics
T. Sonar, I.Thomas (Eds.), Logos-Verlag, Berlin, 2001, 217--226

Simulating reaction-diffusion cellular automata with JCASim

Jörg R. Weimar
Institute of Scientific Computing,
Technical University Braunschweig,
D-38092 Braunschweig, Germany
J.Weimar@tu-bs.de

June 7, 2002

Abstract

The program system JCASim is a general-purpose system for simulating cellular automata in Java. It includes a stand-alone application and an applet for web presentations. The cellular automata can be specified in Java, in CDL, or using an interactive dialogue. The system supports many different lattice geometries (1-D, 2-D square, hexagonal, triangular, 3-D), neighborhoods, boundary conditions, and can display the cells using colors, text, or icons. We use three kinds of cellular automata for reaction-diffusion systems to demonstrate the wide applicability of the simulation system. These are microscopic block-CA, reactive lattice gas CA, and macroscopic CA related to finite difference methods.

1 Introduction

The software system JCASim is an environment for simulating cellular automata. It is written entirely in Java and is capable of simulating many different kinds of cellular automata. Here we present three different classes of cellular automata for the modeling of reaction-diffusion systems and highlight the features of JCASim used in these simulations. The applets demonstrating the different simulations can be found on the web at
http://www.jweimar.de/jcasim/casim_rd/.

2 JCASim

JCASim [3, 10] is a software environment for the simulation of cellular automata (CA). It consists of an application and an applet for the simulation of CA and some translation tools. The system has the following features relevant to the simulation of reaction-diffusion systems:

2.1 Portability through Java

The system is widely portable through the exclusive use of Java as a programming language. This makes it possible to deploy the simulation in a web-presentation without any change to the specifications used in the development of the CA. Despite the use of Java, high performance simulations can be achieved. It is important to follow a few design guidelines. For example, one should avoid object creation in the inner loop, i.e., in the cell state transition function. Even higher performance can be achieved through the use of a translation system which creates native C code (integrated through the Java Native Interface) or creates purely Java multispin code, which is even more efficient. These translations will be described elsewhere.

2.2 Flexible options for the CA

The simulation system can handle many different choices for the different aspects of a cellular automata. It provides lattices of one to three dimensions, can handle two dimensional square, hexagonal or triangular lattices, three different boundary conditions (periodic, reflective, constant), very flexible initial conditions, and cell states which may be compounds of different components with different types. The system does allow probabilistic state transition rules and has options for block-cellular automata.

2.3 Specification using CDL or Java

The state of a CA and the state transition function are properties that need a detailed specification, or better, programming. Only in the simplest cases graphical user-interfaces (which JCASim also provides) are expressive enough for this task. The specification of the composition and type of the cell state and the state transition function can be done in two different languages in JCASim.

The language **CDL** is a special purpose language for cellular data processing [4, 5]. It has a Pascal-like syntax and allows the compact description of cellular automata. For the JCASim system, a few extensions have been

```
cellular automaton Schloglf ; /* Schloegl Model */

type celltype = float;
initial 0.01*random(100) ~ true ;
color   [(*[0,0])*255, (*[0,0])*255, (*[0,0])*255]  ~ true;
var count : float;  n: celltype;
group neighbors = {*[0,0], *[0,1], *[0,-1], *[1,0], *[-1,0]};

rule begin
    /****** Diffusion  *******/
    count := sum(n in neighbors: n)/neighbors.length;

    /****** Reaction   *******/
    count := count - 3.0*(count-0.1)*(count-0.5)*(count-0.9);
    if (count > 1.0) then count := 1.0;
    if (count < 0.0) then count:= 0.0;
    *[0,0] := count;
end;
```

Figure 1: Simple CDL description for a reation-diffusion model (using a float-variable, thus not a strict cellular automaton).

introduced. In JCASim, a CDL-description of a CA is transformed automatically into a java class, which is then directly compiled and loaded into the system. An example CDL-file, which is complete, appears in Figure 1.

Alternatively, CA can be specified directly in **Java**. To facilitate this, the JCASim system provides a framework in which the user only needs to implement a subclass of one class (`State`) and only needs to implement four methods. The derived class contains the components of the cell state as member of variables, and the method `transition` implements the state transition. The helper method `copy` is used to copy the state from a backup cell (this is needed for synchronous updates and is unnecessary in the case of asynchronous updating or block cellular automata). The method `initialize` can be used to set the initial state of some or all cells in the lattice. The method `getColor` is used for the visualization of the cell state.

2.4 Visualization

The cell state, which can be a compound of many different components of different type, can be visualized in three different ways in JCASim. The first option is to use a textual representation, as returned by the `toString` method in Java. Another option is to use a color to paint the cell on the screen. The mapping between cell states and colors is done by the method

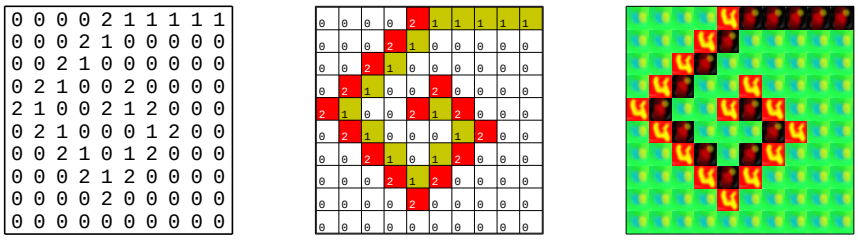

Figure 2: Different representations of a simple CA with three states.

`getColor`, and often it is not injective. A third option in JCASim is to select an image to be displayed at the cell's location, and have the image selected according to the cell's state (in the method `getIcon`). Figure 2 shows all three possibilities.

3 Simulating diffusion in cellular automata

As an example we present three different ways to simulate reaction-diffusion systems with cellular automata in JCASim. First, we treat the diffusion equation, then add reaction. Diffusion can be described microscopically as the effect of many particles executing independent random walks, or macroscopically as a concentration c evolving under the Laplace operator, i.e., described by the differential equation

$$\frac{\partial c}{\partial t} = D\nabla^2 c, \tag{1}$$

where $\nabla^2 = \frac{\partial^2}{\partial x^2}$ or $\nabla^2 = \frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2}$ etc., depending on the space-dimension, and D is the diffusion coefficient.

The description in terms of particles executing random walks can be directly translated into cellular automata models. It is important to obey the conservation law for particles. This is not straightforward in general CA, but there are two classes of CA for which it is easy to verify. These are block-CA and partitioned CA [8].

3.1 Block-CA

In block-CA, the lattice of cells is divided into blocks, and a transition function operates on all cells of one block, changing all of them, but accessing only these cells, not any other neighbors. The transitions of all blocks on the lattice can be executed in parallel or in any order, since they do not interfere. At each time step, a different tiling of the lattice into the blocks is

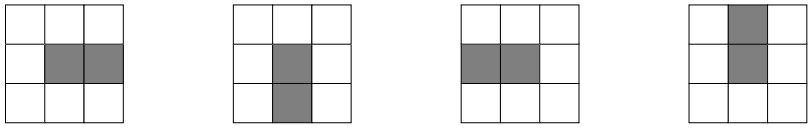

Figure 3: Different orientation of blocks of two cells on a square lattice.

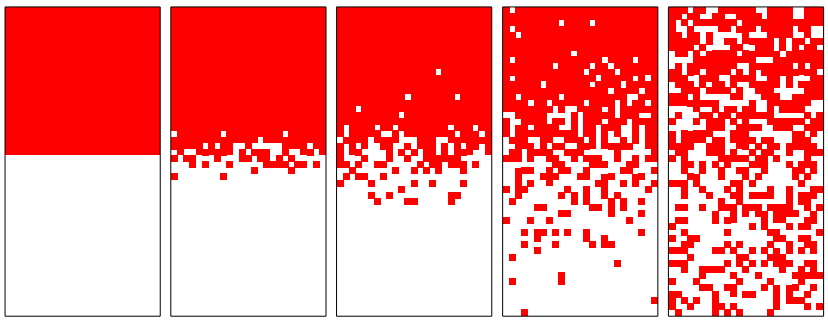

Figure 4: Diffusion of a step in the block-CA (of size 25×50) at times 0, 10, 50, 200, 1000.

used, in this way information can flow across all of the lattice. In block-CA it is easy to verify the conservation of particles, since only the state of one block of cells before and after the application of the transition rule needs to be compared.

The simplest block-CA showing diffusion uses blocks of two cells, and a boolean state indicating the presence or absence of a particle in the cell. The transition rule exchanges the occupancy of the two cells in the block with probability 0.5. In subsequent steps, all possible orientations of blocks of two cells must be used, in two dimensions there are four such orientations (see Figure 3). The result of this rule is that each particle executes a random walk with the mean square displacement $\left\langle |r(t) - r(0)|^2 \right\rangle = \frac{1}{4}t$, which corresponds to a diffusion with diffusion coefficient $D = \frac{1}{8}\frac{\Delta r^2}{\Delta t}$ (with Δt and Δr the time- and space-scales of the CA). Each cell can contain at most one particle. Nevertheless, for concentrations between zero and one, this model leads to correct diffusion. An example can be seen in Figure 4.

3.2 Partitioned CA

The second possibility of ensuring the conservation law is to partition each cell into different compartments. The transition rule then consists of two substeps: First, some compartments are transported to neighboring cells. Since all the cells are equal, the transport happens at the same time in all cells,

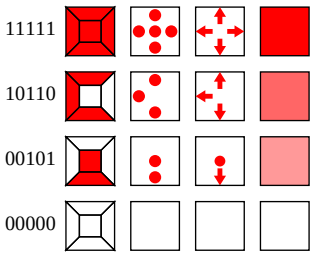

Figure 5: Different representations of a 5-particle lattice gas. Four different states are shown in four possible visualizations: text (binary), compartments, points, arrows, concentration of particles.

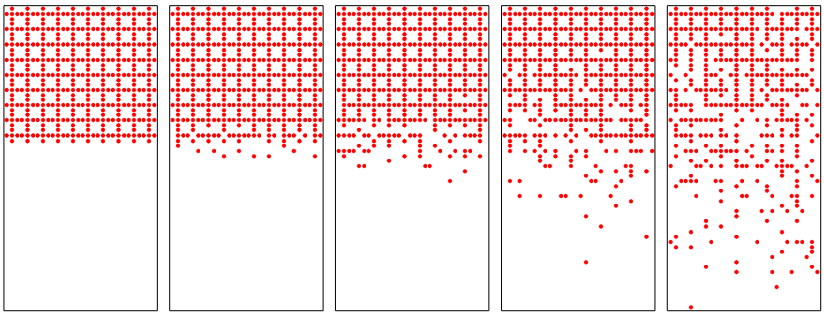

Figure 6: Diffusion of a step in the lattice gas CA (of size 10×20) at times 0, 1, 5, 20, 100.

and the compartments are transported into the corresponding compartments of the neighbors, this only amounts to a permutation of the contents of the compartments (if the boundaries are treated correctly). One can easily verify that such a permutation of cell content does not violate the conservation law. Such cells can be visualized using different representations, some of which are shown in Figure 5.

The second part of the transition rule is local to each cell. In this part, the different compartments interact, e.g., by exchanging particles. One can easily verify whether this part obeys the conservation law, since it operates only on one cell. Models of this type using compartments to hold one or more particles are called lattice gas automata [1, 2]. The simplest two-dimensional diffusive model of this kind uses five compartments, each of which can hold at most one particle. Four of these compartments are exchanged with the four nearest neighbors, the fifth stays in the current cell. Figure 6 shows an example.

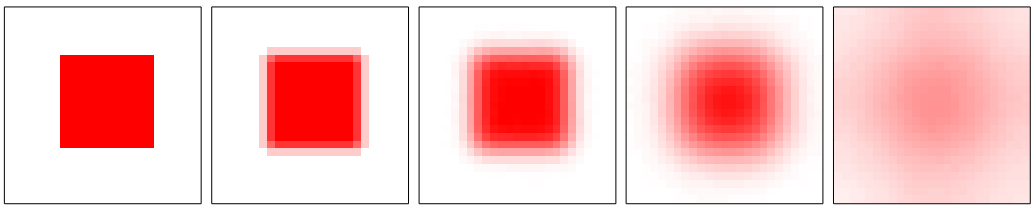

Figure 7: Diffusion of a step in the averaging CA (of size 25×25) at times 0, 1, 5, 20, 100.

3.3 Macroscopic CA

The diffusion equation (Eq. 1) can be solved using finite difference methods. In cellular automata, it is not possible to use implicit methods, so explicit methods remain. Many different kernels or stencils can be used (see [8]), all of which should approximate the Gaussian function

$$G(r) = \frac{1}{\sqrt{4\pi D \Delta t}} e^{-\frac{r^2}{4D\Delta t}}, \tag{2}$$

which is the fundamental solution to the heat equation. One possibility is to use local averaging

$$x(r, t+1) = \frac{1}{|N|} \sum_{\tilde{r} \in N(r)} x(\tilde{r}, t) \tag{3}$$

for any given neighborhood $N(r)$. The division by the size of the neighborhood introduces roundoff errors if an integer cell state is used. In this case, probabilistic rounding can be used to keep the errors small [7]. Figure 7 shows an example, where the cell state is an integer between 0 and 100, the neighborhood consists of the five nearest neighbors, and rounding is probabilistic.

4 Adding reaction to the diffusive automata

To add reactive events to the three kinds of cellular automata, we first need to include more than one species of particles. Here we model the catalytic oxidation of carbon monoxide on a platinum surface. We model the reaction as $A + \frac{1}{2}B_2 \to 0$ with the substeps of adsorption of A-molecoles on an empty site of the surface, adsorption and dissociation of B_2 molecules on two adjacent empty sites of the surface, reaction of adjacent A and B molecules on the surface to a product which desorbs immediately, and diffusion of A particles on the surface (more details in [9]).

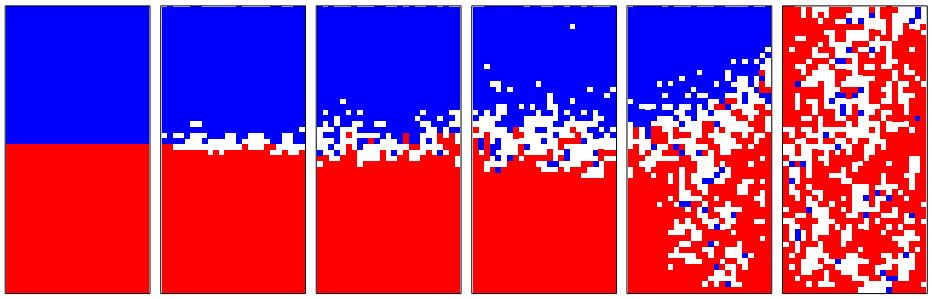

Figure 8: Reactive front in the block CA (of size 25×50) at times 0, 10, 50, 200, 1000, 5000. The dark region shows A-particles, the grey region B-particles, and the white regions are empty sites.

4.1 Block CA

In a block CA, the substeps mentioned above can be translated directly into changes of the configuration of a block of two cells. Each cell can now have one of three states: A, B, or 0 (empty). The transformations are:

$$\begin{aligned} [0,.] &\rightarrow [A,.] && \text{with probability } p_A \\ [0,0] &\rightarrow [B,B] && \text{with probability } p_B \\ [A,B],[B,A] &\rightarrow [0,0] && \text{with probability } p_R \\ [A,0] &\leftrightarrow [0,A] && \text{with probability } p_D \end{aligned}$$

Figure 8 shows such a simulation starting with two large areas filled with A and B particles. At the interface, reaction takes place, and the reactive front moves in both directions (but with different speeds).

4.2 Reactive lattice gas

In the case of reactive lattice gas automata, which are partitioned CA, we introduce reactions in the second substep, where the rule changes the state of each cell independently. The two types of particles are transported independently on two different channels, so that each cell can hold at most 5 A-particles *and* 5 B-particles. The reactive rule then must transform a configuration (α, β) of $0 \leq \alpha \leq 5$ A-particles and $0 \leq \beta \leq 5$ B-particles into a configuration (α', β') with a probability $P(\alpha, \beta)$. This probability can be calculated [8, 6, 1] from the desired polynomial reaction rate (with degree ≤ 5) and the probability to obtain a configuration with α particles given a concentration of A-particles of x:

$$P_x(\alpha) = \binom{5}{\alpha} x^\alpha (1-x)^{(5-\alpha)}. \tag{4}$$

By equating different polynomial equations, it is thus possible to derive the probabilities of configuration changes. Note that for the case where more particles are created than fit in one cell (in this case $\alpha', \beta' > 5$), a correction term must be applied [6].

4.3 Macroscopic CA

In a macroscopic CA with probabilistic rounding, it is very easy to add a reactive term as an Euler-integration step. Simply include the reactive step before the roundoff-operation. One option available in cellular automata, which is a big improvement over other (even high-order) integration methods for ODE's, is to precalulate the reactive step for all possible input values. We use the operator-splitting technique to split the reaction-diffusion equation into the diffusion and the reaction part. The diffusive part is then treated by approximating the Green's function, as describe above, and the reactive part is treated by integrating the resulting ODE excactly (or numerically with high precision using many steps) once in advance for each possible input value:

$$\frac{\partial x}{\partial t} = D\nabla^2 x + f(x)$$

Operator splitting:

$$\begin{array}{rcl} \frac{\partial x}{\partial t} & = & D\nabla^2 x \\ x(r, t+\Delta t) & = & \int_D G(\tilde{r}) x(\tilde{r}, t) d\tilde{r} \\ G(r) & = & \frac{1}{\sqrt{4\pi D \Delta t}} e^{-\frac{r^2}{4D\Delta t}} \end{array} \quad \left| \quad \begin{array}{rcl} \frac{\partial x}{\partial t} & = & f(x) \\ x(t+\Delta t) & = & x(t) + \int_{\tau=t}^{t+\Delta t} f(x(\tau)) d\tau \end{array} \right.$$

$$\begin{array}{rcl} x(r, t+\Delta t) & = & \text{Round}(\text{Diffuse}(x) + \text{React}(x)) \\ & \text{or} & \\ x(r, t+\Delta t) & = & \text{Round}(\text{React}(\text{Diffuse}(x))) \end{array}$$

This approach is unique to the method of cellular automata, since only here there are only a finite number of possible input values, thus it is possible to precalculate the reactive outcome for all of them. When floating-point variables are used, there are far too many possible input values, so that this approach cannot be used.

5 Conclusion and Availability

We have described a new system for simulating cellular automata in Java. The cellular automata can be specified in Java or *CDL* and the system supports many different options for the CA. We have shown three kinds of cellular automata for reaction-diffusion systems as examples, ranging from microscopic (block-CA) to macroscopic approaches with reactive lattice gas automata as an intermediate model. These examples demonstrate the wide applicability of the simulation system JCASim , which is available for download or testing at `http://www.jweimar.de/jcasim/`.

References

[1] David Dab, Anna Lawniczak, Jean-Pierre Boon, and Raymond Kapral. Cellular-automaton model for reactive systems. *Phys. Rev. Lett.*, 64(20):2462–2465, 1990.

[2] Gary Doolen, editor. *Lattice gas methods for partial differential equations*, Redwood City, CA, 1990. Addison-Wesley.

[3] Uwe Freiwald. Eine Java - Simulationsumgebung für Zellularautomaten. Diplomarbeit, Inst. of Scientific Computing, TU Braunschweig, 1999.

[4] C. Hochberger and R. Hoffmann. CDL - a language for cellular processing. In Giacomo R. Sechi, editor, *Proc. Second Int. Conf. on Massively Parallel Computing Systems*, pages 41–64. IEEE, 1996.

[5] Christian Hochberger. CDL – Eine Sprache fr die Zellularverarbeitung auf verschiedenen Zielplattformen . Dissertation, TU Darmstadt, 1998.

[6] Jörg R. Weimar. *Cellular Automata for Reactive Systems*. PhD thesis, Université Libre de Bruxelles, Belgium, 1995.

[7] Jörg R. Weimar. Cellular automata for reaction-diffusion systems. *Parallel Computing*, 23(11):1699–1715, 1997.

[8] Jörg R. Weimar. *Simulation with Cellular Automata*. Logos-Verlag, Berlin, 1998.

[9] Jörg R. Weimar. Coupling microscopic and macroscopic cellular automata. *Parallel Computing*, (to be published), 2001.

[10] Jörg R. Weimar and Uwe Freiwald. JCASim – a java system for simulating cellular automata. In S. Bandini and T. Worsch, editors, *Theoretical and Practical Issues on Cellular Automata (ACRI 2000)*, pages 47–54, London, 2000. Springer-Verlag.