Visual Steps towards Better Understanding of Cellular Automata Algorithms

Mahmoud A. Saber and Nikolay Mirenkov
Graduate Department of Information Systems
The University of Aizu,
Japan

Abstract: The cellular automata (CA) models and corresponding algorithms have a rich theoretical basis. They have also been used in a great variety of applications. A number of programming languages and systems have been developed to support the implementation of the CA models. However, these languages focus on computational and performance issues, and do not pay enough attention to programming productivity, usability, understandability, and other aspects of software engineering.

In this paper, we describe a new special-purpose programming environment developed for visual specification, presentation and explanation of CA systems, as well as, for programming them. This environment is based on using visual patterns, colors, and animation for representing the CA system structures and operations on these structures, and for performing editing and composing manipulations with corresponding software components. Examples of the CA algorithm representations and details of the environment implementation are presented.

Key-Words: cellular automata, multiple views, visual representation, self-explanatory components, Programming Environment

1 Introduction

It has been a long time since Von Neumann invented cellular automata [1]. Cellular automata (CA) are decentralized spatially extended systems consisting of large numbers of simple identical components with local connectivity. Such systems have the potential to perform complex computations with a high degree of efficiency and robustness, as well as to model the behavior of complex systems in nature. For these reasons, CA and related architectures have been studied extensively in the natural sciences, mathematics, and in computer science. They have been used as models of physical and biological phenomena, such as fluid flow, galaxy formation, earthquakes, and biological pattern formation. They have been considered as mathematical objects about which formal properties can be proved. They have been used as parallel computing devices, both for the high-speed simulation of scientific models and for computational tasks such as image processing. In addition, CA have been used as abstract models for studying emergent cooperative or collective behavior in complex systems [2]. A number of programming languages and systems have been developed to implement cellular automata (CA) based systems like CAMEL/CARPET, CLANG, WINALT, and CELLAB (see, for example [3].)

The work presented here is situated in the context of several research areas like pixel rewriters, pixel-level computation, and fine grain parallel computation. Some of them include software visualization techniques. Pixel rewriters are used to explore the variety of interesting computations on, and manipulation of, shape directly in the pixels [4]. Pixel-level computations are used extensively in the early stages of processing and parsing images in the area of computer vision. Filters highlight edges
by enhancing brightness gradients in the pixel array. In morphological analysis, noisy edges are healed by first dilation. Image processing is central to photo manipulation applications of Photoshop type, using pixel operations like blurring, sharpening, and color substitution. WinALT [5] is a simulating system of fine grain algorithms and structures where authors try to create a recognizable and intuitively clear interfaces which ease the learning of the system.

However, most of these systems usually focus on computational and performance issues, and do not pay enough attention to the human abilities, and reserve the graphics and colors to represent the intermediate and final results only, while the model structures and steps of computation are described in a pure textual form. They do not provide essential support to improving usability, understandability, and other aspects of software engineering.

Earlier stages of our research were dedicated to the design and implementation of a visual programming environment, which is used to manipulate with CA systems [6]-[7]. The environment is based on using visual patterns, graphics and colors in representing the CA system structures and computational steps, as well as manipulating with them in different phases of implementation. We also considered the use of this environment for programming image processing algorithms[8]. This consideration, as well as analysis of other applications, provided us with a good feedback for the improvement and further development of the environment.

The reminder of this paper is organized as follows. First, we provide a brief description of the algorithmic “film” format and corresponding notions in Section 2. Then we describe the features of the cellular systems in Section 3. Next, we present film frame examples in Section 5; finally, we report about the enhanced interfaces of the environment used in manipulating with cellular systems in Section 6.

2 Algorithm Filmification
Algorithmic “film” is used as a new type of abstraction to represent computational algorithms by combining mathematical and physical concepts [9]. Mathematical concepts are used to convey the arithmetic/logic aspects of algorithmic activities. Physical concepts are used mainly to convey the spatial and temporal aspects of computation. A film is a series of multimedia frames. One frame represents a view (aspect) of an algorithm; many frames represent many algorithmic aspects. Frame views are arranged into six special groups. The first three groups visualize 1) computational steps and data structures, 2) variables attached to the structures and formulas used in space-time points of algorithm activity, and 3) input/output operations. The fourth group consists of frames of an integrated view where important features from previous groups are presented altogether. Groups 5 and 6 are the auxiliary views related to the film title, authorship, registration date, and links to other algorithms, additional explanations, statistics of the usage, etc. [10]

For a great variety of computational algorithms, the visualization is based on rendering some parameterized structures and partial orders of node scanning in these structures. In other words, it is based on direct demonstrations of computational wave-fronts within the structures. However, there is another group of algorithms where indirect techniques to show algorithm activity should be used. These techniques are based on using visual elements to specify rules defining places and times of activities on different sets of nodes. These sets are defined in associative manner by patterns of neighborhood not by the order of scanning nodes as in the other group. The cellular algorithms are just from this group.

Visually, film consists of a set of scene(s), each scene is a set of frames, a frame is a visual picture consisting of visual symbols, and, visual symbols are atomic units of the filmification. Next, we describe these notions in a bottom-up approach.

• Visual “symbols” are figures, images, icons and text units. These symbols are used to represent contents of frames through rendering basic, foreground, and background items.
Frames are of two types. Basic frames represent some activity essential to computational scheme and the generation of executable code. Auxiliary frames are used to show additional information to further explain or clarify the activity.

Scene is a set of frames related to the same set of the structures and activities. Each frame of a scene has the same structure but show different states of activity. Frames of different scenes, in general, represent different structures or different activities.

In space-time domains, we use structures and a partial order of traversing the structures to visualize: 1) objects, 2) states of objects, 3) sequence of events, 4) transactions between objects. Structures represent the objects and the traversal order represents a scheme. A scheme is denoted by the flashing of cells (nodes) where flashing represents some activity. Any number of cells can be flashed in each frame, and multiple flashing cells indicate parallel activity. An activity can be any operation (a series of formulas) specified by icons or an enhanced text.

Algorithmic film concept is related to a “film machine” concept that is an environmental supporting watching the films and performing manipulations with them.

3 Cellular Algorithm Features

To visualize CA algorithms, we should know their features. So, in this section, we present our observation of cellular model features. They are derived from cellular automata paradigm and its extensions.

CA algorithms have one-dimensional or multi-dimensional grid of cells, each of which is connected to a finite neighborhood of cells that are nearby in the grid. Cells in the regular spatial grid can be in one of a finite number of possible states. This grid of cells is updated synchronously (or asynchronously) in discrete time steps, according to identical local rules (substitution rules) that determine the cell’s next state based upon the current states of the cell and its neighbors. The global behavior of the system is determined by the evolution of the states of all cells as a result of multiple interactions.

Each cell is characterized by two parameters: a state and a name. The state is an abstraction of a data item. It may consist of a set of attributes too, based on the system requirements. The name is a label, assigned to a cell. It may be its number, its code, or a set of coordinates in space.

Algorithms on cellular grids are defined as a set of iterations. Each iteration is a special-purpose repetitive process with a computational body (as a series of computational steps) performing operations on a cellular grid. A computational step is specified by a set of substitutions, all of which are applied in parallel (at the same time), and to each cell in the entire grid (everywhere). New states obtained in this way undergoes the same procedure within the next iteration until, in the long run, there is no single substitution is applicable to the grid, or an expected predicate is fired.

A substitution contains expressions called configurations in its left-hand (up hand) and right-hand (bottom hand) sides. A configuration associates a set of cells with each cell name of a cellular grid. If a cellular grid being processed contains a subset of cells associated with a left-hand side configuration, then the substitution is applicable to this grid. Its application means performing two actions: 1) removing the subset of cells identical to the basic cell-set from the grid and 2) adding the cell-set, associated with the same name by the right hand side configuration. This replacement is performed simultaneously for all subsets of the grid, which meet readiness conditions.

As an extension of the conventional model, we provide substitutions that can update the states of a group of neighbor cells instead of only one cell. For the class of algorithms using this type of substitutions, a case where several substitutions are applicable on the same cell within a grid at the same time is likely to happen. We provide priority conditions to handle this situation. In these conditions substitutions are ranked according to the order they placed in the visual representation (left-right or right-left), or a non-deterministic approach can be specified either, that is no any order for substitutions is defined.
In our model, a cellular grid is processed as a 2D structure of colored cells. Depending on operations on cells, CA systems can be classified into different categories. The first category uses unitary operations to change a state(color) of one cell. The partial order of the cell scanning is not defined by addresses (coordinates) of cells but by external configurations (stencils, masks) around cells or internal configurations (some attribute values) of cells. This means that all cells with the corresponding local configurations are selected for the specification of processing. The second category uses operations to change the states of a group of cells. A set of substitutions is used. Each substitution consists of two local configurations. The first defines the selection of cells and the second replaces the first one. In this way, we implicitly point out cells where processing should take place and a rule for changing “the output image” of the cellular grid. The third category is similar to the second one but uses global configurations (border-to-border set of cells like a strip of rows or columns) and substitutions to extract some features of the processed grid as a whole. One more category, an extended view for formula-based operations is used to change not only cells colors but also perform some arithmetic or logic operations.

4 Film Format Examples
In this section, we present visual representations of different cellular systems in film format. This format aims at providing clear representations of algorithmic steps that ease the understanding and manipulating of the tasks and enhances editing, modifying, upgrading or maintaining the created applications.

4.1 Thinning Algorithm
First example is thinning images system; it is related to image processing field. Thinning is used to remove selected foreground pixels from binary images (ones and zeros). Thinning is normally only applied to binary images, and produces another binary image as output. A simple algorithm for doing this is the following: Consider all pixels on the boundaries of foreground regions (i.e. foreground points that have at least one background neighbor). Delete any such point that has more than one foreground neighbor, as long as doing so does not locally disconnect (i.e. split into two) the region containing that pixel. Iterate until convergence. This procedure erodes away the boundaries of foreground objects as much as possible, but does not affect pixels at the ends of lines.

Thinning film can be used as a stand-alone application or inserted in a complex or composite film like skeletonization. In this mode, it is commonly used to tidy up the output of edge detectors by reducing all lines to single pixel thickness.

Fig.1 depicts the visual representation of the thinning algorithm in a film format, where, all the attributes like dimension, radius, state, neighbor, conditional statements, and substitution rules are implicitly described and physically touched in the form of shapes icons, and colors. The corresponding film consists of one frame that has three micro-icons for control schemes, sample grid, and eight pairs of micro-icons placed downright describing the substitution rules.

4.2 Routing Traffic Systems Algorithm
This system is related to the design of roads and control of the traffic on these roads. Routing is to determine a course by which one can go from an entrance to an exit location. This is closely related to the problem of finding a path through a maze [11]. The corresponding film can
have only one frame depicted by Fig. 2. This frame has two micro-icons for iteration scheme, and the irregularity condition, and a micro-icon for the priority condition. The priority of rules is increased if we consider them from left to right. The priority of cells, to be used for a same rule is decreased if we consider them from top-left corner to bottom-right corner. This view implicitly represents two nested loops (through iteration and substitution rules), ordering (priority) scheme for all substitution rules, and exception (irregularity) for the scheme termination. Fig. 2 also depicts an image of a 2-D grid representing an initial CA. In this image, black cell represents a wall, white cell represents a path, and gray one represents entrance and exit locations. To find the route, all the paths that culminate in a dead end are eliminated; the four pairs of micro-icons placed downright describe the corresponding substitution rules so that all that remain is the path.

4.3 Adding Binary Numbers Algorithm

![Fig.2 A film frame of the routing algorithm](image)

Third example is adding binary numbers. It is related to design and simulation of new types of parallel computer architecture [5]. Adding many non-negative binary integers based on cellular automaton approach is considered. The corresponding film has also only one frame shown in Fig. 3.

In this frame two substitution rules, three micro-icons for the iteration scheme, irregularity condition, priority condition, and a sample grid are presented. The binary integers are arranged in successive rows of the 2-D grid. A black cell represents a value of one, and white one represents zero. The grid converges in time toward a final configuration that is the sum of the numbers.

4 Implementation

In the case of a conventional language, a source program is input as a text with some support of a text editor. In our case of the film language, a source program can be input as collection of icons, colored shapes, in addition to text whenever it is more expressive than other media. The users can create their own films from

![Fig.4 Panels of multimedia interface subsystem for the CA domain](image)
scratch or get their needs from the saved films. After that, the user can watch a film or its parts and perform editing and composing manipulation. Editing mode allows the specification of cellular models. The power of a specification method is mainly related to its flexibility, generality, and capability to customize visualizations [12]. While watching a film is close to algorithm animation field, an algorithm animation visualizes the behavior of an algorithm by producing an abstraction of both data and the operations of the algorithm (see [13] for more details of this field).

The multimedia interface subsystem is developed in JAVA. Fig.4 shows some of its panels (editing, watching, and, formula attaching panels) developed and used for the CA domain. See [6]-[7] for more details about the design and implementation of this subsystem.

5 Conclusion
We have implemented a visual programming environment that is used to manipulate with CA algorithms in shorter time, less effort, and reduced cognitive load in comparison with traditional systems. In this paper, we presented our observation of CA systems, provided examples from different application areas, and reported about the enhanced interfaces from the visual programming environment used in manipulation with CA.

References