

C-cards: using paper and scissors to understand computer science

Andrea Valente
Aalborg University in Esbjerg
email: av@cs.aue.auc.dk
home page: <http://cs.aue.auc.dk/~av>

12 August 2003

Abstract

We define a simple card game, where cards are computational elements; computing machines can be defined, built and animated in a concrete way by disposing cards and moving pegs around them, following formal rules. We discuss how to use this card game as an educational tool, to introduce children 8 to 10 years old to the concept of computation, seen as manipulation of symbols. Students will not need any mathematical knowledge to explore information theoretic concepts by means of our tool; moreover the students can easily expand the tool with new components for exploring new concepts. Graph-based syntax and graph-rewriting semantics are given to formalize cards and their behaviour.

Keywords

Educational tool, teaching Computer Science, graph-rewriting system, visualization, concretization.

1. Intro

In this paper we aim at defining an educational tool, suitable to teach basic computer science concepts to classes of 8 to 10 years old. Such a tool has a number of desirable features:

- no mathematical knowledge needed to understand and use it. In CS courses, computing machines are classically introduced using mathematical definitions and exercises are usually based on algorithms taken from number theory (eg factorial, Fibonacci). This approach does not scale to children.
- the tool should offer a concrete, mechanical metaphor for reasoning about computation: it should enforce the idea that computation is a property shared by many different systems, some of which made of a few simple pieces. We think formal semantics should be hidden *inside* real world objects, that can be animated in game-like simulations.
- direct manipulation and learning-by-doing. The tool should enable students to define their own computational elements, extending the basic card

set; it should also encourage the exploration of problem-solving and design methodologies (top-down and bottom-up project development, for example).

- cheap and flexible support (paper), wide availability (web-based simulator) and individual learning. The tool should be both web and paper-based: in this way even when it is not possible to have a PC for each student, the tool could be used on individual basis. An optional advantage of a tool made of paper, is that in the era of Internet, distribution will be inexpensive.

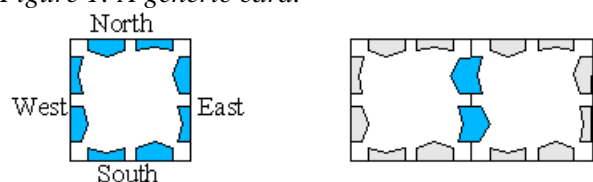
Taking all these requirements into account, we conceived a card game, with both a paper and a javascript implementation (both available at the author's home page). The game has a precise, formal set of rules and it is suitable to build and study various computational devices: we call this game *c-cards*, that stands for *computational cards*.

The cards are introduced informally in section 2, via some examples; in section 3 *c-cards* are provided with graph-rewriting semantics. Then we discuss how to turn *c-cards* into an educational tool, ie how to express exercises for young students; we give some insight about what teaching areas can be covered by our tool. We present the web implementation of our tool in section 5, and finally we discuss some related projects and some problems that need further investigation.

2. The deck

A *card* is a square piece of paper (see figure 1, on the left); directions, and card sides will be addressed as north, east, south and west.

Figure 1: A generic card.

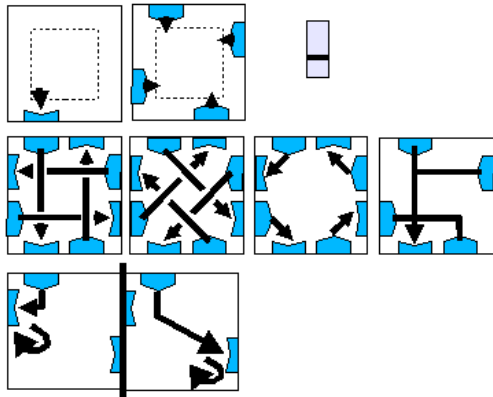


Sometimes cards will be labelled, ie named, to address them more easily in the discussion. Each card has two *ports* on each of its four sides: an input port and an output one, so that when two cards are laid aside, their ports connect in a complementary way to form a *complete port* (figure 1, on the right). The area inside a card can contain arrows, going from an input port to an output one, defining the behaviour of the card.

The basic *c-cards* set is called the *deck* (figure 2). It is composed of seven card types: the *peg-source* card and the *peg-pit* card (first row); the *cross* card,

the *east* and *west turn* cards and the *confluence* card (second row). All the cards in the second row of figure 2 are basically pipes, useful to build statical circuits on which pegs will move. The *switch* card (last row of figure 2) is the only one double-faced: after cutting it, we have to fold it along the thick vertical line. Together with these seven card types, there is a *peg* (the small rectangle at the far right on the first row).

Figure 2: A basic c-cards set: the deck.



To play a game at c-cards, we have to compose a *card circuit*, by connecting cards together; every circuit must have at least one source and one pit card. To execute a computation, a number of (identical) pegs will be placed on the source cards of a circuit; then we will move them, one at the time, to represent the flow of information traversing the circuit. Pegs will run through the cards following arrows, from port to port, until they reach a pit card, where they must stop.

By now we leave parallelism aside, for the sake of simplicity, and we assume that pegs are released sequentially by source cards, each after an arbitrarily long delay: in this way we can reason as if a single peg at the time is ever present in a circuit. Under these assumptions about execution, our card circuits will have a deterministic behaviour.

Source cards contain (possibly empty) queues of pegs, being scheduled out southwards; every source shall then be described by a string, or a regular expression, defining the (finite) sequence of pegs it will release. Eg a source defined by *(white.black)** produces a finite sequence of alternating white and black peg; to specify the precise number of pegs produces, a string can be used instead, eg *white.black.white*.

The last card type we have to detail is the *switch* card (third row of figure 2), the only one capable of changing its state dynamically. A switch card has two faces, each describing a specific state and

behaviour. When a card of this type is placed in a circuit it is in its *initial state*, ie the first peg arriving from north will exit westwards; when leaving the card the peg will force a *flip*: the switch card will be flipped in its other state (face). A second peg arriving from north will then exit eastwards. The state of the card will flip back to the initial one, and so forth.

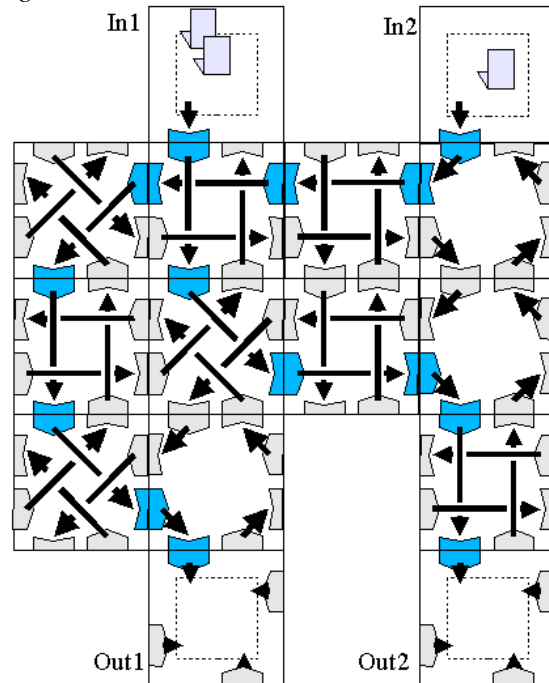
The switch card represents a simple form of memory (similar to a flip-flop boolean circuit) and is the only active computational element in the deck.

2.1. A card circuit

To demonstrate the behaviour of c-cards, we compose a circuit (figure 3) with two pegs placed on the *In1*-labeled source card, and one on the other, labeled *In2*. After a number of steps, the two pegs will arrive at the *Out2* pit, while the other one will stop at the *Out1* pit. In the figure all unreachable ports are light-gray, to simplify spotting the two main paths pegs can follow in the circuit.

The action performed by this circuit is to *swap* its inputs: in fact we shall say that pegs starting from the *In1* source card are in state TRUE, while pegs starting from *In2* are FALSE pegs. For the same reason, a peg stopped at *Out1* pit will be in TRUE state, and FALSE if it ends up on the other pit.

Figure 3: The not card circuit.



The whole circuit should then be seen as a boolean *not gate*, because it turns TRUE pegs into FALSE ones, and vice versa. This suggests that boolean operations are realizable with c-cards, although we

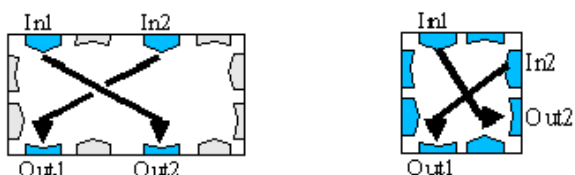
know that c-cards do not reach Turing-completeness. However we feel that boolean operators are still too *mathematically oriented* for young students, so we limit the discussion of c-cards in education to regular languages and simple pattern-recognition exercises. We also want to draw attention to an interesting concept related to our card circuits: *time-reversibility*, closely connected with the idea of representing information by means of physical objects. Standard boolean gates create and destroy boolean values as they compute; a gate in which no bits are to be lost or freshly created during computation, is called *time-reversible*, meaning that we could totally recover what the input was using the output. All the circuits that we can define with c-cards have this property (for a presentation of time-reversible gates and their expressive power, refer to Fredkin and Toffoli (1982)).

Finally we would like to remark that although all pegs are treated as identical by our cards, pegs can come in different colors, with labels on them and possibly in different size, but only for helping the user spotting them and following them in their way through the circuit. Only their physical location, or better their arrival direction, can be used as information by cards, and this is why a circuit changing peg positioning, is executing a computation. Pegs in c-cards have a status similar to the one of tokens in Petri Nets (see Ajmone et al. (1994)), but more *persistent*. Moreover these circuits bear similarities with flow graphs (see Milner (1989) and Gardner (1999)), because all these graphical notations express movements of information flowing into and out of ports, in a topology of communicating neighbours.

2.2. Shapes, macros and custom cards

The *shape of a circuit* is defined as the structure obtained by replacing every card, different from source and pit, with a blank square of paper. Arrows can be drawn freely on these blank parts to show the pegs' paths in the circuit.

Figure 4: A macro-card and a card.



The shape of a circuit is its functional specification, and reasoning about shapes (ie incomplete circuits) is useful to encourage top-down analysis and modelling in students.

A standard feature of programming languages is the possibility of defining macro-instruction, and use them to write more readable and compact programs; macros, automatically expanded by translators and compilers, encourage a simple form of reuse and abstraction.

The corresponding concept in c-cards is the one of *macro-card*. Once a circuit is defined and we like to use it repetitively to build more complex circuits, we can map the whole circuit it into a single *large* card, preserving the circuit's behaviour. The idea here is simple: source cards became input ports, while pits are turned into output ports; arrows (and some informal annotations about actions to be performed) will express the high-level behaviour of the large card. The macro card in figure 4 on the left has the same behaviour of the not-circuit: its ports are labelled after sources and pits of the original circuit, and two crossing arrows represent the swapping of incoming pegs.

It is also possible to define new *custom cards* behaving as a whole circuit, although in these cases the geometry of the original circuit is hard to preserve. Since cards are realized in the real world as paper squares, students can easily define their own cards. An example of custom card is visible in figure 4 on the right: the new card will behave roughly like the not-circuit.

3. Graph-rewriting semantics

Cards and the way they can be connected to build circuits form the syntax of the c-cards game. Here we give a graph-rewriting semantics to our tool.

As a card is mainly a set of ports (of two types) plus some arrows connecting them, we can consider a card circuit as a graph of ports, each one possibly containing pegs.

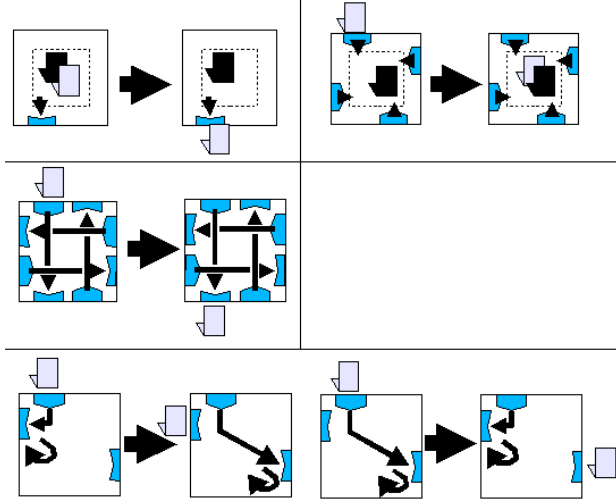
In figure 5 we show the rewriting rules for source and pit cards (first row), cross and switch cards (second and third row respectively). All rules are intended to work up-to card rotations, this explains why there is only one rule for pits, while otherwise four should be needed. Card rules are deterministic, triggered by the presence of pegs, and depend upon the current state of cards (in fact the switch card has two rules). We omit rules for turn and confluence cards because they behave essentially as the cross card: pegs movements are directed by arrows.

As c-cards have a sequential execution model, no more than one peg can be present on a single port at any given instant; to simplify design we assumed from the beginning that pegs flow in the circuit one at the time.

We think that graph-rewriting rules are rather simple

to explain to young students and fit naturally with our visual and physical idea of computation.

Figure 5: Rewriting rules for c-cards.



Moreover we believe it is evident that children can easily manage games with formal rules (eg the goose game or RPG games) therefore hide formal aspects behind a game-like look seems a good, general trick for designing educational tools.

4. Educational applications

Wanting to organize a basic course in CS, and adopting c-cards as our main tool, we can follow ideas from the pedagogical pattern project (Sharp et al.): at the beginning of the course some simple static circuits (such as the not-circuit) can be shown and animated with pegs, so to explain the behaviour of all seven types of cards through direct manipulation. This strategy conforms to the *spiral* and *early bird* pedagogical patterns.

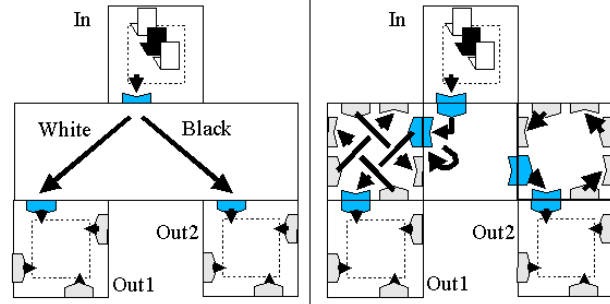
After this first, hands-on introduction to the c-cards tool, some top-down modelling examples should be shown: a shape, together with the specification of its desired behaviour, can be proposed to the class. The solution of the exercise is an implementation of a card-circuit performing the required task.

As an example of such an exercise, let us consider the shape in figure 6 (on the left), where the *In* source card is defined by $(white.black)^*$, and the desired circuit has to separate whites from blacks (we recall that colors are meaningful only for users, not for the cards). If we count pegs as they exit from their source card, here white pegs will be odds, and the black ones even; a circuit implementing these specifications correctly must make white (odd) pegs stop at the *Out1* pit, and black (even) ones at the *Out2* pit card. Students are asked to give an implementation of the shape-specification; a possible

answer is depicted in figure 6 (on the right).

A mathematical description of this exercise could be to implement a circuit *counting modulo 2*, but we don't need to use such terminology with children, since speaking of patterns of pegs is expressive enough to specify the correct behaviour of the circuit.

Figure 6: A shape with specification, and a possible implementation.



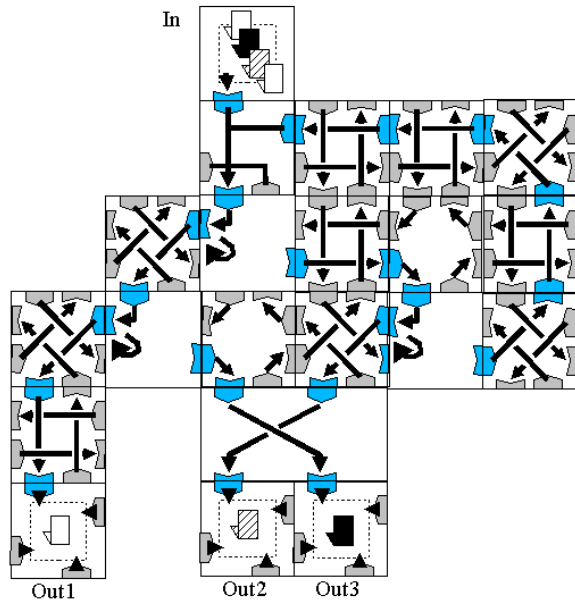
This is one of the main reasons to propose c-cards for teaching to young students: no mathematical knowledge is needed for reasoning about card behaviours, nor to build them. Patterns and physical manipulation of symbols, ie pegs, seem to be general enough concepts to introduce the study, construction and use of computational machines.

Suppose now we want to make a more complex exercise, working with a source defined by $(white.slashed.black)^*$. The resulting circuit will have three pits, in which pegs of the same *color* will accumulate: the first (western) pit will receive all white pegs, the second all the slashed ones and the black pegs will stop at the third. The circuit in figure 7 implements a solution for this exercise.

There is a general technique (that can be discussed with students), for building a *modulo k* peg splitter, for any given *k*. When *k* is a power of 2, we simply have to put a cascade of switch cards, forming a tree with *k* levels and placing the pits in the correct positions; note that by using some copies of the not-circuit, we can always re-arrange the final (southern) part of a circuit so that pits are in any desired east-to-west order. With this method will have a pit card collecting each class of integers *modulo k*; in fact for all other values of *k*, we can start building a circuit for the first power of 2 larger than *k*, then short-cutting all pits in excess.

As these *modulo k* circuits should be widely used to implement bigger circuits easily and in a modular way, it seems reasonable to convert them into macro-cards, and consider them as off-the-shelf components.

Figure 7: A circuit splitting pegs modulo 3.



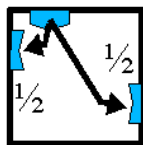
We can decide to name k -color splitters these macros, in this way the card in figure 6 will be called 2-color splitter.

The possibility to use macro-cards and forgetting about their real implementation (ie. the circuit they are based on), together with the idea that different circuits can solve the same problem, enables the teacher to discuss optimization. A circuit implemented using macro-cards, or better its expansion (ie the whole circuit obtained replacing every macro with the actual circuit it represents), can in the general case be simplified, without changing its functional behaviour. Investigating in this direction, students can be guided to understand, in a practical way, the meaning of complexity and performance in computing machines.

4.1. Frequencies, probability and transmission

In order to design exercises about probability, information and transmission theory, we propose a simple extension to the basic deck: a random card, that is the non-deterministic version of the switch card. The random card is shown in figure 8.

Figure 8: The random card type.



A peg entering a card of this type will be sent out towards east or west, with $\{1/2,1/2\}$ probability distribution. Again the students do not need to know

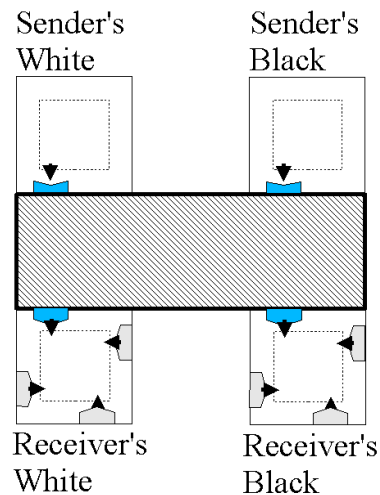
probability, they can simulate the random choice needed by this type of cards, by tossing a coin.

After a small number of trials, they should be convinced that half of the pegs will leave the random card from the east out port and half from west: in this way we can use frequency instead of probability for discussing about these cards and the circuits that students will built with them. Using two or more of these cards in cascade, we can implement almost every probability distribution needed; macros and custom cards can be devised to create specific probability distributions.

A possible exercise for the class about the use of random cards can ask to build a circuit for dividing white pegs from black ones, generated by the source (white.black)*, with 50% probability of making an error (note that this is the random version of the modulo 2 counter discussed in the previous section).

When the students have realized that a single random card is sufficient to solve this exercise, we can propose them the exercise in figure 9, called the pegs' telegraph.

Figure 9: The pegs' telegraph example.

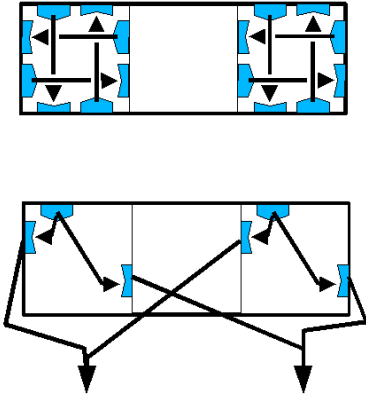


We suppose two friends want to communicate sending pegs to each other. Pegs come in two colors: white and black. The circuit in figure 8 is incomplete and we want the students to design the grayed part so that there is exactly 50% of errors in sending the pegs. The solution is shown informally in figure 10, at the bottom (the top part shows the design to be used to have a perfect transmission). Here an error is defined as sending a white peg and receiving a black one and vice versa, ie a peg starting from the source card labelled *Sender's White* arriving at the pit card labelled *Receiver's Black* and vice versa.

This last exercise demonstrates how, with the simple extension of the random card, we can discuss of data

transmission and errors within c-cards. The concepts of *redundancy*, *message coding* and *error detection* can also be introduced using similar exercises. We can propose the class to modify the circuit in figure 7; the modified circuit should work sending a couple of pegs, like *blue.blue*, instead of the *Sender's White*, and *red.red* instead of the *Sender's Black*. The receiver will then get blue and red pegs, and will have to consider couples to understand the original message sent from the sender.

Figure 10: Solution of the pegs' telegraph example.



In this modified circuit there are *illegal couples*: couples of pegs that can be received but that could never be sent, like *red.blue*, which is obviously caused by a transmission error. We can show the students that illegal couples are the result of our decision of using a code with two pegs for each one of the original message, and they give us the ability to *detect errors*.

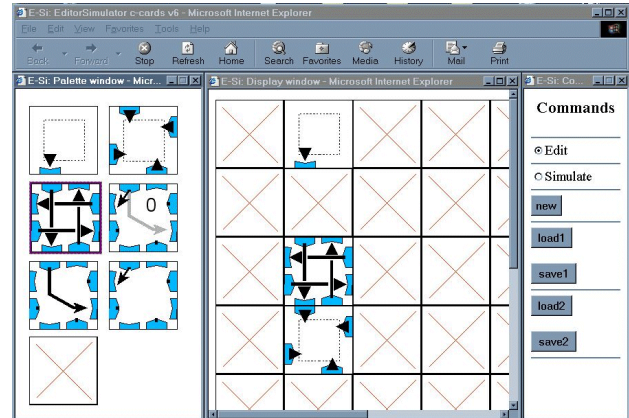
We are currently investigating possible ways to give a visual meaning (using c-cards circuits) to Shannon's information measure: we would then be able to reason about most of Shannon's results within the c-cards tool, and most important without mathematical notations (eg Shannon's entropy for a source of symbols is computed using logarithms and probability distributions, which are very complex for children to manipulate and understand).

5. Web implementation

As we write in the introduction, we believe that an educational tool for teaching young students should both be physical and computer/internet based. This is why we developed c-cards to have both a paper and an html-javascript implementation.

Figure 11 shows E-Si (pronounced *easy*), the c-cards' editor and simulator. The program (available at the author's home page) runs entirely on the client machine, so it can be used on-line or downloaded and deployed in a school's lab.

Figure 11: E-Si's interface.



The current version of E-Si is a simple experimental realization and implements only the basic card set; it enables the user to design, save/reload and test circuits.

In the next version we plan to add the definition of custom cards and the possibility of executing circuits with multiple pegs.

6. Related work

It is kind of difficult to locate the subject of this study: toy-making, educational and computer science issues have been closely related in developing c-cards. However there are other works that go along the same interdisciplinary path. The requirement for our educational tool to be a physical toy, can also be found in the concept of *tangible programming* (see McNerney(2000)), defined as *the activity of arranging the blocks to build (as opposed to "write") computer programs*. As we are discussing about toys as didactic tools, we want to cite Papert's constructivism, that in his words, *attaches special importance to the role of constructions in the world as a support for those in the head*; he also maintains that children should learn better by *self-directed activities*, such as *bricolage* and *thinking*, both closely related to c-cards.

For the methodology to adopt when using toys in education, we found that our ideas match with a pedagogical pattern called *toy box*, presented in Sharp et al. (1996) by the *pedagogical pattern project*: it aims at giving *students broad historical and technological knowledge of the field by letting them "play" with illustrative pedagogical tools*.

7. Conclusions and future works

In this article we presented c-cards, a card game with formal rules, and discussed its application as educational tool suitable for teaching computer science concepts to young students.

With c-cards students can create and manipulate their own computing machines, reason about design and implementation of card circuits. Also concepts like probability and information (intended as Shannon's entropy) can be discussed on a concrete basis using a simple extension of the standard card deck; likewise we showed how to set up experiments about redundancy and transmission errors.

However there are still many problems (and spin-offs) that we would like to investigate:

- Measure/prove the real computational power of c-cards.
- Give a parallel semantics to our cards; will they become similar to cellular automata? Shall we then be able to model and study classical concurrent systems (such as the dining philosophers) with our tool? In fact parallelism and concurrency are considered a fairly complex subject for bachelor students, so scaling them to children is an intriguing goal.
- An implementation the cards in a mainstream educational language such as Logo/StarLogo to see whether it is possible for the students to pass smoothly from cards to a real, universal programming language.
- Design a three-dimensional, real-time implementation of c-cards, based on connectable cubes, that could compute *automatically*, by gravity; pegs should be replaced by balls running through cubes. Can such a tool be made of paper? Could we use plastic parts to create the equivalent of the switch card?
- Enrich the number of exercises and examples for c-cards, to build a *teaching methodology* on top of it. We wish to be able to draw inspirations from automata and language theories (deterministic finite automata for example), Petri nets and queue networks.

References

Ajmone Marsan, M. and Balbo, G. and Conte, G. and Donatelli, S. and Franceschinis, G. (1994). *Modelling with Generalized Stochastic Petri Nets*. Wiley Series in Parallel Computing, John Wiley and Sons.

Fredkin, E. and Toffoli, T (1982). *Conservative logic*. *International Journal of Theoretical Physics*, 21(3/4): 219–53.

Gardner, P. (1999). *Graphical presentations of interactive systems*. Submitted for publication. Available at: <http://www.cl.cam.ac.uk/~pag20/>

Helen Sharp, Mary Lynn Manns, Phil McLaughlin, Maximo Prieto and Mahesh Dodani (1996). *Pedagogical patterns – successes in teaching object technology*. *ACM SIGPLAN Notices*, Vol. 31(12), December 1996, pp. 18-21.

McNerney, T. (2000). *Tangible Programming Bricks: An Approach to Making Programming Accessible to Everyone*. Ph.D. Thesis.

Milner, R. (1989). *Concurrency and Communication*. Prentice Hall.