Concretization and animation of Finite Automata with c-cards

Andrea Valente
Aalborg University in Esbjerg
Denmark

av@aue.auc.dk

Abstract

The paper presents a new way of introducing *finite automata* to classes of 10 to 12 years old, and in general to students with a limited mathematical background. Our approach builds on *computational cards* (or c-cards), a project aiming at scaling-down the learning complexity of computer science core contents, by presenting symbol manipulation via a tangible, physical metaphor. C-cards are still at a very early stage of development as an educational tool, yet we believe they represent a nice, intuitive mind-tool, that can be consistently applied to a variety of (theoretical) computer science concepts. Here an algorithm for mapping finite automata on c-cards is given and its educational implication discussed: the mapping provides a mean to concretize and animate automata. The algorithm works with both deterministic and non-deterministic finite automata.

1 Introduction: motivation and background

In this paper we present a new way of introducing *finite automata* to classes of 10 to 12 years old, or older students with a limited mathematical background. Finite state machines are central in (theoretical) computer science and represent a powerful mind tool for understanding a great variety of computing machines that we (and our children) face in everyday life: mobile phones, video-recorders, vending machines and even more traditional devices like cars and lifters can be easily described and discussed by formalizing them as finite automata (FA for short). However these concepts are seldom learned outside university courses and even in that environment they are usually considered tough by students (Hämäläinen, 2003); in this context a state-of-the-art tool is JFLAP (Cavalcante et al., 2004), an instructional software covering very many topics in the theory formal languages. However fast and robust, visual and interactive in nature, this tool (like others of its kind) does not make any attempt to unify the various notations used in the field, resulting in a collage of highly integrated ad-hoc tools, each adopting a different, specific formalism; the suggested target-group of JFLAP is university-level students.

Our approach builds on *computational cards* (or c-cards), a project aiming at scaling-down the learning complexity of computer science core contents, by presenting symbol manipulation via a tangible metaphor: physical objects represent computational elements and objectified symbols (Valente, 2003). Although c-cards are still at a very early stage of development as an educational tool, we already showed that small-sized card systems (called *card circuits*) can implement interesting and quite complex behaviors. Here we propose to concretize and animate FA with c-cards. In this way students can play and learn about automata theory *before* actually being exposed to its formal notation, so to ground the meaning of abstract symbols in concrete examples (a discussion on *symbol grounding* can be found in Harnad (1990)); our approach is strongly related to *learning-by-doing*, advocated by Papert and constructivism in general (Papert, 1993).

In the rest of the paper we will recall the definition of deterministic FA (or DFA) and the basic characteristics of c-cards as an educational tool. After that an automaton will be defined and implemented by a card circuit. The discussion of this example will guide us to define a general mapping between DFA and card circuits. Some considerations about the way of introducing DFA to the students will conclude the paper.

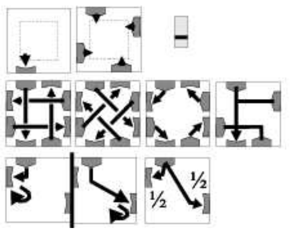

Figure 1: A deck of c-cards.

2 Finite Automata

A DFA is defined as a 5-tuple: $< S, \Sigma, T, s, F >$ where S is the set of all states of the automaton, Σ is its alphabeth, T is the transition function, defined as a mapping from $S \times \Sigma$ onto S; there are also a single state $s \in S$, that is the initial state for the automaton, and the set of all final states (also called acceptance states), F.

An automaton can be used in two ways: as a language (or string) recognizer and as a generator. In the first case the user provides a string (in the alphabeth of the automaton); the automaton will read the input string one symbol at the time, and change state accordingly, until the string is completely consumed. At this time the user can check if the DFA is in one of its final states, and if so, the string is considered as *accepted* by the machine. The set of all the strings accepted by a DFA is the *language* recognized by the automaton. The other use of a DFA is to generate strings of its language. In this case the user starts with the automaton in the initial state and an empty output string. At each step the user picks (randomly) a transition of the form $T(currentState, symbol_i) = newState$, where $currentState$ is the actual state of the state machine, and applies it, changing state to $newState$ and appending the symbol $symbol_i$ to the output string. The *game* can be stopped when the DFA is in a final state, so that the output string is part of the language of the automaton.

Another kind of finite automata is the non-deterministic finite automata, or NFA, where the transition function is defined from $S \times \Sigma$ onto S^* (sets of states). After reading a symbol from the input string the state of an NFA can change in a non-deterministic way, into one of a number of new states.

3 C-cards

C-cards are presented in (Valente, 2003), where we give the 8 basic types of cards (together they form a deck, figure 1); some of these types represent pipes, connecting elements, while others are the active computational elements. The standard set of c-cards is called a deck.

To play a game at c-cards, we need to print some copies of the deck, then we cut free our cards and some pegs (small rectangles of paper, representing symbols). At this point we are ready to compose a card circuit, by connecting cards together; in every circuit there is at least one source (a starting point for pegs) and one pit card (where pegs will stop and accumulate). 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 are *objectified symbols*, that run through the cards, following a precise semantics for each card type, until they reach a pit card, where they must stop.

C-cards semantics is defined by a graph-rewriting system (Valente, 2003), where rewriting card circuits corresponds to manually animating them by moving pegs around. This semantics fits well with the concrete and interactive nature of c-cards; even very young students can easily manage games with formal rules (see Kharma et al. (2001), McNerney (2000) and Papert (1993)).

Sometimes, in problem solving, top-down reasoning is crucial, and in these cases it is

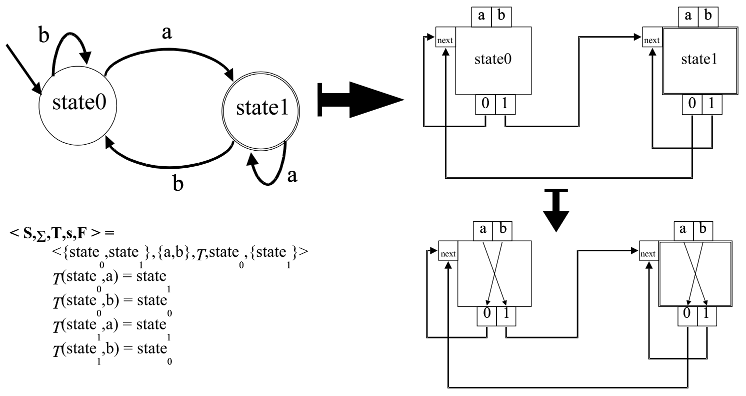

Figure 2: A DFA and its shape in c-cards

possible to sketch a card-circuit functional specification, called *shape*. In a shape, source and pit-cards are visible, but the rest of the circuit is blank; arrows can be drawn freely on blank parts to show pegs paths. Shapes are used later, to implement state machines with our cards.

Many abstract concepts, like deterministic state machines, probability and information transmission can be discussed on a concrete basis with c-cards, that, for what we know, is the only tool where both computing elements and information itself are reified. A distinctive feature of c-cards is that they show the rather abstract concept of *state* of a computing system in a very explicit way: the state of a card circuits coincides with the visual layout of its cards. To change the state, you have to change the layout (for example, by flipping one card on its the other face).

We believe c-cards have many of the features needed to encourage explorations of (theoretical) computer science concepts; moreover it is a very cheap and extendible tool. In fact we designed c-cards to have both a paper (tangible) and an html-javascript implementation; using the E-Si (pronounced easy) editor and simulator, teachers can design exercises for their courses based on traditional lectures (to visualize and animate), lab-based activities, and also for individual learning.

4 Implementing finite automata in c-cards

As an example consider the deterministic finite automaton in figure 2 on the left; its recognized language is $b^*aa^*(bb^*aa^*)^*$, and an example of accepted string is *bbaaa*. This automaton can be represented as graph with two states, connected by arrows, labelled with symbols from the alphabeth.

To implement the DFA in c-cards we can start by designing some shapes, corresponding to the automaton's states, that we later will refine into circuits. Our idea is based on the fact that a state of a DFA can be seen as a black box, waiting for an input (the next symbol provided by the user) to compute the automaton next state; such computation is directed by the DFA's transition function. Then the next state is activated and it will wait for its input.

Figure 2 in the upper-right corner, shows the card circuit that results from this idea of *state-as-black-boxes*; there are infact two shapes, labelled $state_0$ and $state_1$, each with two source cards on the top, representing the possible inputs that the state can receive (a or b). Each shape also has two possible outputs, at the bottom, labelled 0 and 1, representing the next state of the automaton; finally there is a pit-card labelled *next* at the side of each shape, that is used during the animation of the circuit as the unique entry point in the shape, something like a handle to access that state. Special colors can be used to indicate which circuit-shapes correspond to the intial state (and final states) of the automaton. The final things we need to do after defining these two shapes, is to connect them in the right way and

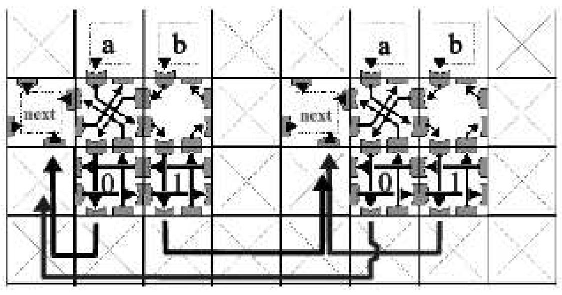

Figure 3: A possible implementation of the DFA-shape

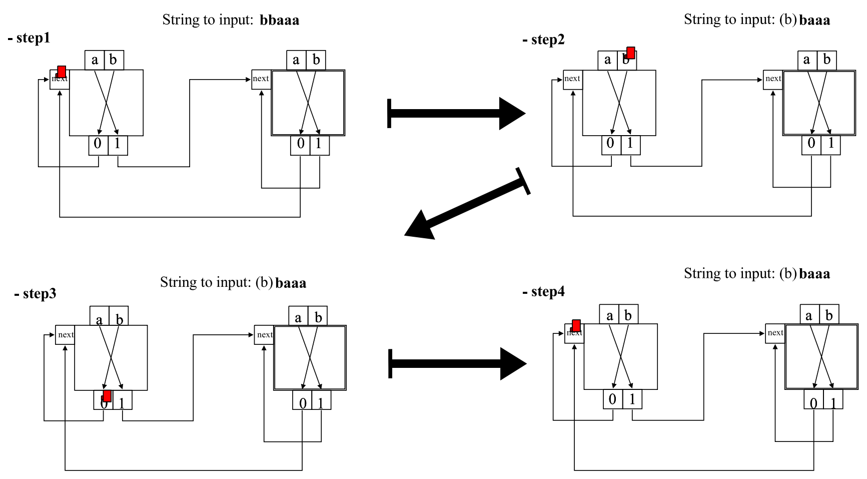

Figure 4: Initial steps of the animation of a DFA-shape

refine their internal structure, so to match the transition function of the original automaton.

A partial refinement of the states is visible in figure 2, lower-right corner, while the (almost) complete implementation of our circuit is depicted in figure 3 (some connections are still sketched, to simplify the readability of the circuit).

The implemented card circuit can then be used recognize the input string *bbaaa*; figure 4 shows the first steps of this recognition: at the beginning a peg (the only one needed to animate this circuit) is set into the *next* pit-card at the side of $state_0$, because it is the initial state for this automaton. Then the user has to manually set the peg in the source-card labelled *b* of the $state_0$-shape of the circuit. The shape will then answer by sending the peg out through the channel labelled 0, from where it will reach again the *next* pit-card at the side of $state_0$. At this point the user have to set the peg in the *b* source-card again, to simulate the reading of the second symbol of the input string. This process will go on until the string is consumed, and if the current state will then be $state_1$, then the string belongs to the language of the automaton.

The general algorithm for implementing a DFA $< S, \Sigma, T, s, F >$ in c-cards is then:

- for each $state_i$ in S, design a shape with:
  - a pit-card at the side, labelled *next*
  - a source-card on the top for each symbol of Σ
  - a cross-card at the bottom for each state in S
  - an arrow connecting the source-card $symbol_j$ to the cross-card $state_k$, where $T(state_i, symbol_j) = state_k$

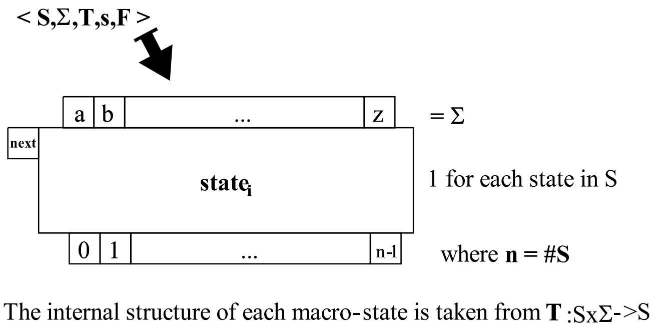

Figure 5: The general recipe for mapping DFA into c-cards circuits

- for each shape $state_i$
  - for each cross-card $state_l$ at its bottom,
    - draw an arrow connecting it to the *next* pit-card at the side of the $state_l$ shape

This mapping (visually summarized in figure 5) is easily extended to NFA. Suppose that for the state $state_i$ and a symbol $symbol_j$, the transition function of the NFA is defined like: $T(state_i, symbol_j) = \{state_a, state_b\}$. Then we just need to use a random-card to implement the non-deterministic choice between states $state_a$ and $state_b$.

5 How to present the subject to the students

A possible way of introducing students to FA (with c-cards) is then to start with remarking the importance of top-down analysis in problem solving. Then we could give the students some shapes already defined, that implement a particular automaton, maybe taken from some known contexts. For example a DFA with three states, called *home*, *school* and *holiday*, and a transition function representing the life of a student commuting among the three places. Animations could then be shown, to demonstrate how to use automata and how to *read* their results. After two or three examples, generalization can take place, and we can discuss what is needed to describe a generic FA. In this way we can lead students towards the standard definition of this class of state machines: the 5-tuple.

As for the exercises, there are a number of possibilities for presenting automata theory in an *enjoyable* way (Hämäläinen, 2003). For example students can be asked to design their own *bike lock*, with code: the lock will be modeled as an automaton and the accepted language will be the code (or a set of possible codes). In another kind of exercises we can exploit a finite automaton as string generator: the string produced can then be mapped on *colored scarfs*. Another source of inspiration could be some simplified version of ethograms (see *www.ethograms.org*), where an ethogram is a (probabilistic) automaton, used in ethology to provide an inventory of the behaviors of a species; this should strengthen the idea that automata exist *in the wild*, in the real world. Non-determinstic automata could be introduced by discussing errors and random behaviors (see the way information theory and communication are expressed in c-cards, in Valente (2003)).

6 Conclusion

In this paper we show that FA can be implemented and discussed within c-cards, thus enlarging the area of applicability of our educational tool: not only it can be used to introduce young students to the idea of computational devices, and to the tasks involved in designing, implementing and testing symbol-manipulating machines; it is also possible to build on top of that and eventually structure a course of automata theory, suitable for slightly older children.

C-cards offer in fact a plain and tangible metaphor about computation, and this results in a scalable mind-tool, that can be consistently applied from simple computer science concepts to more formal and complex issues (such as finite automata). This september, in a workshop (ICALT 2004 at Joensuu, Finland) c-cards will be tested with a large class of children (6 to 12 years old); we anticipate that feedback and data collected in that occasion will help to improve both the design and effectivness of our tool. Our long-term goal is to base a teaching methodology on c-cards, and for this reason we are currently exploring an extension to concurrency and more complex state machines, such as (approximations of) Turing machines; the latter is quite challenging, given that standard c-cards do not have enough expressive power to implement that class of language-recognizers.

References

R. Cavalcante, T. Finley, and S. H. Rodger. A visual and interactive automata theory course with jflap 4.0 . *Proc. of SGICSE'04, March 3-7, 2004*, 2004.

W. Hämäläinen. Problem-based learning of theoretical computer science. *Proc. of the Kolin Kolistelut - Koli Calling 2003, Koli, Finland*, 2003.

S. Harnad. The symbol grounding problem. *Physica*, D 42:335–346, 1990.

N. Kharma, L. Caro, and V. Venkatesh. Magicblocks: a construction kit for learning digital logic. *Computer Applications in Engineering Education*, august, 2001.

T. McNerney. *Tangible Programming Bricks: An Approach to Making Programming Accessible to Everyone.* M.S. thesis, MIT Media Laboratory, Cambridge, MA (2000), 2000.

S. Papert. *The children's machine. Rethinking School in the Age of the Computer.* Basic Books, New York, 1993.

A. Valente. C-cards: using paper and scissors to understand computer science. *Proc. of the Kolin Kolistelut - Koli Calling 2003, Koli, Finland*, 2003.