TAGS: A Software Tool for Simulating Transducer Automata

Agustín Esmoris
Department of Computer Science and Engineering - Universidad Nacional del Sur
Av. Alem 1253 - B8000CPB Bahía Blanca, ARGENTINA
Tel. +54 - 291-459-5135 - Email: agustinesmoris@softhome.net

Carlos Iván Chesñevar[1] **María Paula González**
Department of Computer Science and Industrial Engineering - Universitat de Lleida
C/Jaume II, 69 - E-25001 Lleida, SPAIN
Tel. +34 - 973.702764 - Fax +34 - 973.702702 – Email: cic@eup.udl.es

Keywords: Digital Design Systems, Theoretical Computer Simulators, Automata Theory, Transducer automata

Abstract

This paper introduces **TAGS** (Transducer Automata Graphical Simulator), a software tool for teaching different aspects of transducer automata theory, a theoretical topic which underlies many aspects of the design of sequential digital circuits. **TAGS** allows to simulate both *Moore* and *Mealy* transducer automata, integrating different theoretical concepts associated with them. The student can define an arbitrary transducer automaton in an interactive way, being able to simulate and trace its behavior by means of different *views*.

1 Introduction and Motivations

Automata theory [10,4,16] is an important subject in the Electronic Engineering (EE) and Computer Science and Engineering (CS&E) curricula. *Transducer automata* are a special kind of finite state automata which are particularly important in the context of EE, as they provide a theoretical basis for the design of sequential systems [5,24]. From a pedagogic point of view, making some theoretical concepts of automata theory attractive for EE students is not an easy task. Many times students find considerably difficult to acquire the main underlying ideas as they are overwhelmed by the abstract formal notation being used. This situation has motivated the development of a number of *theoretical computer simulators* [8] as educational tools which allow the student to implement and `bring to life' many topics of automata theory which traditionally are studied and analyzed mathematically (as mathematical abstractions) rather than algorithmically (as properties or characteristics of computing

[1] Corresponding author.

devices). This applies particularly to transducer automata, which effectively allow to characterize a basic model of *interactive computation* [14,15] and provide the basis to understand the design of sequential digital circuits. Several theoretical computer simulators have been developed for modeling automata in the CS&E Curricula (such as Finite State Automata, Turing Machines, etc.) [8]. However, the design of similar tools for transducer automata theory has been particularly neglected, maybe because this particular topic is more related to Electronic Engineering rather than Computer Science.

In this paper we introduce **TAGS** (Transducer Automata Graphical Simulator), a software tool developed for helping students to design, analyze and execute transducer automata. **TAGS** includes a number of features which make it attractive as a pedagogical tool in a traditional Automata Theory course or an introductory course in Digital Systems. Some of the most relevant features of **TAGS** include:

- the design and execution of an arbitrary Mealy or Moore transducer automaton;
- the automatic conversion from a Mealy automaton into an equivalent Moore automaton (and vice versa) by applying equivalence theorems between these two kinds of automata;
- the minimization of both Moore and Mealy transducer automata by means of an algorithm that includes animation effects along with a step-by-step explanation.

This paper is structured as follows: Section 2 introduces some fundamentals on transducer automata theory as well as some relevant properties and theorems which are modeled by **TAGS**. Section 3 discusses the main features of **TAGS** and their application in an Automata Theory course. Section 4 outlines some salient features of **TAGS** and related teaching strategies oriented to promote *significant learning* of transducer automata theory. Finally, in Section 5 we detail the main conclusions that have been obtained.

2 Transducer Automata: Fundamentals

Next we will recall some basic concepts of transducer automata theory as well as some special properties and equivalence results. A (*deterministic) finite-state transducer automaton (FSTA)* is a device much like a deterministic FSA, except that its purpose is not to accept strings, but rather to *transform* input strings into output strings. Informally, a FSTA T starts in a designated initial state s_0 and moves from state to state, depending on the input, just as a deterministic finite automaton does. On each step, T emits (or writes onto an output tape) a string of zero or more symbols, depending on the current state s_i and the current input symbol. The state diagram for a deterministic finite-state transducer looks like that for a deterministic finite automaton, except that either **states** or **arcs** are labeled with an output symbol *s*, producing *s* as an output every time the state is reached (or the arc is traversed). In the first case, the automaton is called a **Moore automaton** [20]; in the second case, the automaton is called a **Mealy automaton** [19]. Formally:

Definition 1 [Finite State Transducer Automata]: A deterministic Finite State Transducer Automaton (FSTA) is a 6-tuple $T = (S, \Sigma, O, \delta, s_0, f_0)$ where S is a finite set of states, $S \neq \emptyset$, Σ is the input alphabet, O is the output alphabet $\delta : S \times \Sigma \rightarrow S$ is the next state function or transition function, s_0 is the initial state, $s_0 \in S$, and f_0 is an output function defined as $f_0: S \rightarrow O$ (if T is a Moore automaton) or $f_0 : S \times \Sigma \rightarrow O$ (if T is a Mealy Automaton).

Fig. 1 illustrates these two kinds of automata. Fig. 1 (left) shows a Moore automaton T that given the input sequence “aabba” outputs the string “11010”. Fig. 1 (right) shows a Mealy automaton T’ that behaves the same way. Note that in the first case states are labeled with output symbols, whereas in the second case the output symbols are associated with the arcs connecting nodes in the automaton.

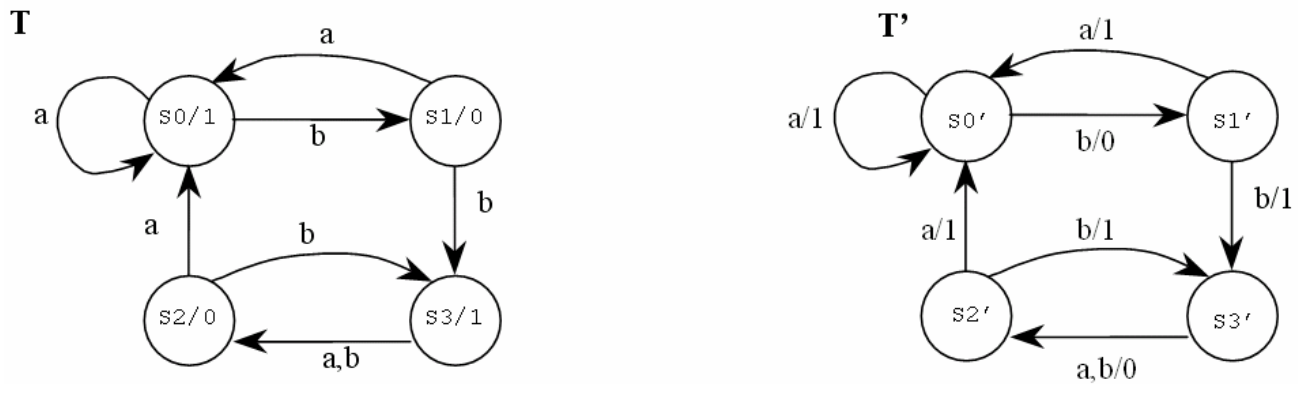

Figure 1: Example of equivalent Moore and Mealy transducer automata

States in any transducer automaton T can be *classified* according to how they behave with respect to different input strings. Thus, for example, a state s_k is called a *transitory state* if no input sequence α exists such that $\delta(s_k, \alpha) = s_k$. Similarly, a state s_j is called an *unreachable state* if there not exists one input sequence α such that $\delta(s_0, \alpha) = s_{j.}$ Consider the automata T and T' as defined in Fig. 1. Then no state is unreachable, and states s_0 and s_0' are not transitory states.

Definition 2 [Translation of α via T]: Given an arbitrary transducer automaton T = (S, Σ, O, δ, s_0, f_0) and an input string $\alpha = c_1c_2c_3...c_j$, $1\leq k\leq j$, $c_k\in \Sigma$, the translation of α via T (denoted T(α)) is defined as the output produced by T for that input α. Formally it results $T(\alpha) = T(c_1c_2c_3...c_j) = a_1a_2a_3...a_j$, $1\leq i \leq j$, $a_i \in (O \cup \{\lambda\})$, where λ represents the empty string. For the Moore automaton in Fig. 1 (left), it is the case that T(aabba) = 11010.

Minimizing Transducer Automata

Given a finite state automaton F with S states, a well-known result in automata theory refers to computing the *minimized* version of F [10,16,22]. That is to say, computing an automaton F' with S' states such that F and F' behave the same way and F' is the automaton which has a *minimal* number of states. A similar result applies for transducer automata:

Definition 3 [Output language]: Given a transducer automaton T = (S, Σ, O, δ, s_0, f_0), we define the *output language* for T as $L(T) = \{ w \mid T(\alpha) = w, \forall\ \alpha \in \Sigma^* \}$ (i.e., the set of all output strings that can be generated by T for every possible input string α).

Definition 4 [Minimized version of T]: A transducer automaton T' = (S', Σ, O, δ', s_0', f_0') is the *minimized version* of T if L(T') = L(T), $T(\alpha) = T'(\alpha)\ \forall \alpha \in \Sigma^*$ and besides T' has a minimal number of states.

The minimization algorithm for computing T' from T is based on computing equivalence classes from the original set S of states in T. See [10,16,22] for more details.

Equivalence Theorems:

Let α be an arbitrary input string, and let $T(\alpha)$ and $T´(\alpha)$ be the output strings obtained after processing α by means of a Mealy automaton and a Moore automaton, respectively. It should be noted that such strings can never be equal in length (no matter how T and T´ are defined), as the length of $T(\alpha)$ will always be less than the length of $T´(\alpha)$, for every possible α. However, the initial output provided by a Moore automaton can be discarded, defining T and T´ as equivalent if for any possible input α, $xT(\alpha) = T'(\alpha)$, where x is the output of T' for its initial state. According to this convention we can state the following theorems:

Theorem 1: If $T_1 = (S, \Sigma, O, \delta, s_0, f_0)$ is a Moore automaton, then there is a Mealy automaton T_2 equivalent to T_1 .

Theorem 2: If $T_1 = (S, \Sigma, O, \delta, s_0, f_0)$ is a Mealy automaton, then there is a Moore automaton T_2 equivalent to T_1 .

3 The TAGS Software: Overview

In this section we will describe the main characteristics of the **TAGS** *(Transducer Automata Graphical Simulator)* software tool. First we will give an overview of how **TAGS** can be used to define and execute an arbitrary Mealy or Moore automaton. We will then describe the different *views* of a given automaton which can be visualized using **TAGS**. Finally we will detail the main facilities provided by **TAGS** which solve important aspects of the curricula in transducer automata theory.

3.1 Designing and executing transducer automata with TAGS

TAGS is a standalone program that can be easily installed and run on any typical Windows-based PC. It is available as freeware from http://cs.uns.edu.ar/~cic/simuladores.htm. The **TAGS** interface looks similar to well-known educational tools for automata theory (e.g. JFLAP [3]), but specifically oriented for dealing with typical topics in transducer automata theory. As **TAGS** starts, an empty canvas is available for the user to draw the graph corresponding to either a Moore or a Mealy transducer automaton. At the right of this canvas a *design toolbar* appears, offering a number of icons for accessing different options: adding a new state to the current automaton, defining the input/output alphabet, save/load automaton to/from disk, etc.

The user can add a transition arc between two states by double-clicking on the origin state and then clicking at the destination state. It must also be noted that arcs may be dynamically reshaped using the mouse. States and arcs labels can also be dragged around in the same way. Moreover, the user may change the appearance of the automaton (e.g. colour of states and arcs, width of arcs, size of the circles corresponding to states, etc.). The user can also delete an arc or state by simply selecting it with the mouse and then pressing *Delete* on the keyboard. Additionally, **TAGS** also provides a set of keyboard shortcuts to facilitate the design process of a transducer automaton.

Once the transducer automaton has been completely designed, the user can start a simulation process of the execution of the automaton for an arbitrary input string (according to the input alphabet originally defined). **TAGS** provides the user with an *execution toolbar* located below the design toolbar. The execution toolbar provides four traditional buttons: Play, Stop, One Step Backward, and One Step Forward. By pressing "Play" **TAGS** starts executing the current automaton for the specified input string and continues until all input symbols have been processed. Once the "Play" Button has been pressed, **TAGS** enters in *simulation mode*, where the automaton cannot be edited until the input string has been completely processed. The "Stop" button allows the user to stop the animation process at any time, making **TAGS** return to the original *design mode.* The third and fourth buttons (backward and forward animation) allow animating the execution in a step-by-step fashion. The user can either rewind or move forward the current execution as many steps as he wants. The user can also set up the desired animation speed by means of a slide bar provided with the execution toolbar. Fig. 2 shows a snapshot of the **TAGS** interface when drawing a Moore automaton.

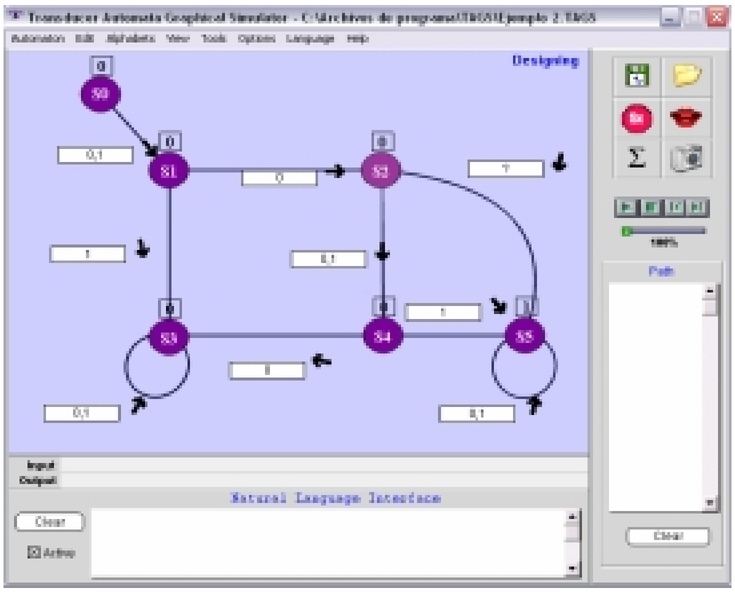
Transducer Automata Graphical Simulator
Designing
S0
S1
S2
S3
S4
S5
Input
Output
Natural Language Interface
Clear
Active
Path
Clear

Figure 2: The design of a Moore automaton using TAGS

3.2 Allowing Different Views for the Same Automaton

Using simulators as teaching aids to complement the different topics presented in an automata theory course encourages different *views* for simulating a given automaton. A view is a particular representation of an automaton which emphasizes some of its features. In [18] four different views associated with pushdown automata (PDA) are proposed: (1) ***tape view*** (showing the current input tape); (2) ***stack view*** (showing the current stack); (3) ***path view*** (verbose mode); and (4) ***automata image view*** (PDA visualization as a graph). We adapted and extended these views for transducer automata, incorporating them as a facility provided by the **TAGS** software. Next we detail each of such views:

1) **Transition diagram view:** When working in design mode, **TAGS** provides a graph-like representation of the automaton. In this view the student can design his own automaton as naturally as if he were drawing it on a blackboard. When simulating an automaton, the current state as well as the corresponding arc traversed when processing a symbol is always highlighted.

2) **Input/Output view:** this view provides an overview of the string being processed as well as the current output string produced by the automaton. In this view, the input string is shown within a text-field where the last symbol read is clearly highlighted. This view is practical for the user to know which part of the input string has already been processed.

3) **Path view:** this scrolling window reports a summary of every action performed during the simulation process. This summary involves a description of the current state, the symbol just read, the output obtained and the new state reached after processing the current symbol.

4) **Natural Language view**: this view provides a full explanation of every action performed by the automaton when processing an input string. The explanation is presented to the student in natural language (as if he were being told about it by a human tutor). This view is complemented by one particularly motivating feature: the ability to make use of the speech engine software provided by the Microsoft Windows-XP operating system. By means of such speech engine, **TAGS** provides the student with an explanation of how the automaton proceeds as a given input string is processed, or how to carry out the different steps in the minimization algorithm by computing classes of equivalent states.

5) **Theoretical view:** this view involves a matrix-like layout in which the formal definition of the current automaton is shown. In contrast to the three preceding views, this one is static, complementing the transition diagram view by means of a formal definition of the automaton that has been defined graphically.

3.3 Using TAGS to solve different aspects of Transducer Automata Theory

As presented in Section 2, there are several practical problems and theoretical issues that can be identified in the context of transducer automata. **TAGS** helps the student to cope with these situations, making easier and more motivating to solve typical exercises of transducer automata theory. Some special options included in **TAGS** aiming at this purpose are the following:

- **Automata Minimization: TAGS** can simulate the well-known minimization algorithm [10,16] on an arbitrary transducer automaton. This can be applied to either Moore or Mealy transducer automata. The execution of the minimization algorithm is performed stepwise, taking into account the main difficulties the students will find when applying the algorithms by themselves. First, **TAGS** identifies all unreachable states, deleting them from the graph. Afterwards the corresponding k-equivalent classes are found (i.e., all states that behave in equivalent way for strings whose length is less than or equal to *k*). Whenever a class was found, **TAGS** highlights all of the states belonging to such class with a colour that has not been used to represent another class before. This helps the student to easily recognize the different classes inside the automaton along the minimization process. **TAGS** will give a full-spoken explanation of the minimization process in case of having a suitable speech engine available on the operating system. Fig. 3 illustrates the screens associated with a Mealy Transducer Automaton and the application of the minimization algorithm.

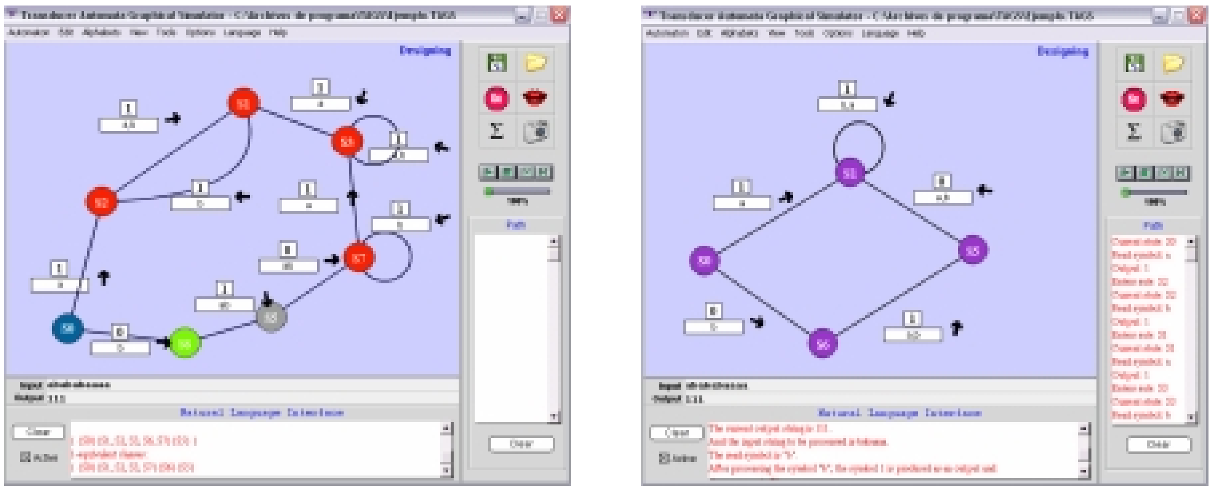

Figure 3: Mealy Transducer Automaton before and after applying the minimization algorithm

- **Converting Transducer Automata:** given a Moore transducer automaton, **TAGS** allows the student to transform it into an equivalent Mealy transducer automaton by applying the corresponding algorithm [4]. Conversely, **TAGS** is able to transform any Mealy transducer automaton T into an equivalent Moore transducer automaton T'. For this last case, **TAGS** shows a matrix-like layout with the complete definition for the automaton T' instead of transforming its graph. That is due to the possible high number of states and transitions that may be generated after the transformation from Mealy to Moore.

- **Analyzing States Properties:** when working on the drawing canvas, after holding the mouse pointer on a state for 2 seconds a pop-up window appears. This window reports different facts about the state under the mouse pointer (such as the condition of reachable/unreachable state, the list of states reachable from that state and many other special properties of these machines).
- **HTML Generator:** this option creates an HTML document containing the formal definition of the current automaton on the drawing canvas as well as its full picture. This aims at facilitating students to get a print out of the automata they define, or even paste it into other electronic documents (e.g. a webpage).
- **Updater and multilanguage support: TAGS** has the ability to perform an automatic update from its current web location (http://cs.uns.edu.ar/~cic/simuladores.htm), as well as a multilanguage support. The last version of **TAGS** can be run either in English or Spanish languages.

4 Promoting Significant Learning with TAGS: a constructivist viewpoint

We contend that **TAGS** can be used as a complementary tool to promote significant learning in the theory of transducer automata within a *constructivist approach*. Constructivism is a theory of learning which claims that knowledge is actively constructed by the student, and not passively absorbed from lectures and textbooks [1,2,6,7]. According to this theory the construction of new knowledge is performed recursively, based on previous knowledge. The constructivist approach postulates that effective learning relies on an explicit process in which viable mental models are constructed. This process of construction and reconstruction of ideas is actively guided by the teacher. The ultimate goal of constructivism is to achieve *significant learning*, i.e. adequate mental models that will be available for use in different contexts. According to Bruner [6,7] some of the major principles that lead to significant learning are the following:

1. New knowledge acquisition must encourage *discovery and active learning*. As the process of learning is a reconstruction of the student's mental models carried out by the student

himself/herself, it demands an active attitude. By getting involved in his/her own learning process the student can perform meta cognition activities as the result of thinking about the learning process itself. Discovery learning activities and the stimulation of meta cognition enable students to continue learning by themselves in the future with greater effectiveness.

2. The use of *previous knowledge* plays a major role in learning. Because the learning process is based on the construction and reconstruction of ideas, the learning process of highly abstract concepts is not possible without having some other structure developed from previous knowledge to build on. Therefore any effort to teach must be related to the mental models of the learner, providing a path into the taught material based on the learner's previous knowledge.
3. New knowledge must be related with experience and social context significant to the learner. Our learning is intimately associated with our interaction with other human beings (teachers, peers, society, etc.). The social dimension of learning involves discovering the implications and *human significance* of what is being learnt.

Next we outline some salient features of **TAGS** which can be analyzed under a constructive approach, in the context of the principles described above. As discussed in the introduction, theoretical computer simulators have proven to be a very motivating link between theory and practice, encouraging active and discovery learning of specialized knowledge through abstraction, interaction and visualization [8,9,23]. As a specialized theoretical computer simulator, **TAGS** encourages discovery and active learning in an automata theory course. In our teaching experience, learning how to make use of the different facilities provided by **TAGS** (e.g. using different views for the same problem) has also led to questions and problematic situations which incentivated self-directed learning. It must be remarked that the existence of multiple views for the same automaton has helped students to find a proper abstraction level in different problem solving situations. A proper handling of such abstraction levels is particularly important in the context of analyzing students' mental processes. In this respect, a simulator like **TAGS** helps students to "zoom in and out" when working on practical exercises.

When using **TAGS**, students apply previous knowledge in different ways. On the one hand, they refine and extend some concepts of automata theory they already know, as transducer automata theory naturally “extends” many notions related to finite state automata (e.g. by preserving the notions of state, transition, etc. used in finite state automata, and incorporating the new concept an “output alphabet”). The different views that **TAGS** makes available for any transducer automata defined by the student are also based on concepts already known from previous undergraduate courses (e.g. using a matrix or a graph as equivalent representations of a given transducer automaton).

The use of **TAGS** as a tool for solving practical problems of transducer automata theory was complemented by including details of the historical context in which transducer automata emerged. Although an Automata Theory course for Electronic Engineering is clearly not centered on such a topic, it can be considerably enriched by introducing related articles and discussions. Such contextualized analysis allows our students to become aware that transducers were the standard finite state machines in the 1950s and 1960s, when computer science was emerging as a new discipline and shared many common topics with electronic engineering. Edward Moore’s original work [20] was thus concerned in discussing the relationship of an abstract machine’s external input/output behaviour to its internal state-transition behaviour, allowing thus to model *interactive computation* performed by hardware devices or any other type of system where the amount of available memory is finite. As computer science turned away from interaction and toward a notion of computation restricted to algorithms, the study of transducers was no longer an important topic in the computer science curricula, and was kept as a theoretical topic for the design of sequential circuits. Knowing such facts make students aware of the role of foundational knowledge in relationship with the later development of real-world applications (e.g. the design of actual sequential circuits).

5 Conclusions

Simulation environments for automata theory reinforce the significance of theoretical issues when solving practical exercises, providing an interesting link between theory and practice. Although most important topics in an automata theory course can be illustrated using such simulators, an in-depth treatment of transducer automata has been neglected so far in existing simulator programs. **TAGS** makes it possible to “bring to life” many typical theoretical problems of transducer automata theory, such as converting a Moore automaton into a Mealy automaton (and vice versa), or obtaining a minimal transducer automaton. In this context, the possibility of having different views for modeling the formalization and execution of an arbitrary transducer automaton are particularly attractive.

In our opinion, the theory and application of transducer automata are very interesting topics for many students. We think that the availability of a software tool like **TAGS** makes such topics even more motivating and attractive, complementing the later usage of other software tools (such as OrCAD [17]) for the design and simulation of digital sequential circuits based on the use of logical gates.

References

[1] H. Aebli. Denken: Das Ordnen des Tuns - Kognitive Aspekte der Handlungstheorie. Klett-Cota Ed., 1980.

[2] M. Ben-Ari. "Constructivism in Computer Science Education". In *Journal of Computers in Mathematics and Science Teaching,* 20(1):45-73, 2001.

[3] A. Bilska, K. Leider, M. Procopiuc, O. Procopiuc, S. Rodger, J. Salemme., and E. Tsang. "A collection of tools for making automata theory and formal languages come alive". In *Proc. 28th SIGCSE Technical Symposium on Computer Science Education*, 1997, pp. 15-19.

[4] T. L. Booth. "*Sequential Machines and Automata Theory"*. John Wiley & Sons, Inc., New York, 1967.

[5] S. Brown and Z. Vranesic. "*Fundamentals of Digital Logic with Verilog Design*". McGraw-Hill Higher Education, 2003, ISBN 0-07-283878-7.

[6] J. Bruner. Towards a Theory of Instruction. Harvard Univ. Press, 1966.

[7] J. Bruner. Actual Minds, Possible Worlds. Harvard Univ. Press, 1986.

[8] C. Chesñevar, M. Cobo, and W. Yurcik. "Using Theoretical Computer Simulators for Formal Languages and Automata Theory". In *ACM SIGCSE Bulletin*, Vol. 35, Nr. 2 (2003), pp. 33-37.

[9] C. Chesñevar, A. Maguitman, M. P. González, L. Cobo. "Teaching Fundamentals of Computing Theory: A Constructivist Approach". *Journal of Computer Science and Technology*, special issue on Computer Science Education (submitted).

[10] D. Cohen. A. *Introduction to Computer Theory*. John Wiley & Sons, Inc., New York, 1997.

[11] A. Esmoris and C. Chesñevar. "Una Herramienta para la Simulación de Autómatas Traductores en la Enseñanza de Teoría de la Computación". In *Proc. of the IX Argentinean Conference in Computer Science*, 2003, pp. 287-295.

[12] L. D. Fink. "Creating Significant Learning Experiences: An Integrated Approach to Designing College Courses". Jossey-Bass Ed., 2003.

[13] C. Ghezzi, M. Jazayeri and D. Mandrioli. *"Fundamentals of Software Engineering"*. Prentice Hall, Englewood Cliffs (N.J.), 1991.

[14] D. Goldin. "Persistent Turing Machines as a Model of Interactive Computation ". In *Foundations of Information and Knowledge Systems*, 2000, pp. 116-135.

[15] J. Hartmanis. "Observations about the development of theoretical computer science". In *Annals in the History of Computing*, 3(1), 1981, pp.42-51.

[16] J. Hopcroft and J. Ullman. *"Introduction to Automata Theory, Languages and Computation"*, Addison-Wesley, 1979.

[17] A. Hossain. "*Computer Aided Electronic Circuit Board Design and Fabrication: Using OrCAD/SDT and OrCAD/PCB Software Tools"*. Prentice Hall, 1995.

[18] J. McDonald. "Interactive Pushdown Automata Animation". *ACM SIGCSE Bulletin 34*, 1 (2002), pp. 376-380.

[19] G. H. Mealy. "A method for synthesizing sequential circuits". *Bell Systems Journal 34* (5), pp. 1045–1079, 1955.

[20] E. F. Moore. "Gedanken experiments on sequential machines". In *C. E. Shannon and J. McCarthy, eds., Automata Studies (Princeton University Press)*, 1956, pp. 129–153.

[21] R. Rasala. "Toolkits in first year computer science: A pedagogical imperative". *ACM SIGCSE Bulletin* 32(1), (2000), pp. 185-191

[22] B. W. Watson. "Taxonomy of finite automata minimization algorithms". Computing Science Note 93/44, Eindhoven University of Technology, The Netherlands, 1993.

[23] C. Yehezkel, W. Yurcik, and M. Pearson. "Teaching Computer Architecture with a Computer-Aided Learning Environment: State-of-the-Art Simulators". In *Proc. Int. Conf. on Simulation in Engineering Education*, 2001.

[24] J. F. Wakerly, “*Digital Design. Principles and Practices*”. (2nd. Ed) Prentice Hall, 1995.