Introducing First-year Students to Theoretical Computer Science

Favre, Liliana[1] Mauco, Virginia[2] Barbuzza, Rosana[3]

Departamento de Computación y Sistemas
Universidad Nacional del Centro
de la Pcia. de Buenos Aires
7000 Tandil
Buenos Aires. Argentina

Abstract

There is a need to educate students about advances in Computing Theory that are effective for new technologies. This work describes an introductory course implemented in the Undergraduate Degree Program in Systems Engineering at U.N.C.P.B.A. ("Universidad Nacional del Centro de la Provincia de Buenos Aires") in Argentina. This course provides an introduction to the theory of computing, starting from the study of a hierarchy of formal languages and automata, and basic concepts of computability and complexity by Turing machines. It has been organized in a way that is accessible to first-year students.

Keywords: Information Systems Curricula, Theoretical Computer Science, Computability, Formal Language, Automata

1. INTRODUCTION

The main topics of the theoretical Computer Science have been included in most Computer Science and Information Systems (CS/IS) curricula. A traditional course covers the theory of formal languages and automata and notions of computational complexity and computability (ACM 1991; IS'97 1997; MSIS' 2000).

The scope of Computing Theory has changed over the last years and it will be continuously influenced by new applied problems. The emergence of new tools, techniques and paradigms forces a continuous reevaluation of the topics covered and pedagogical approaches used. We believe that topics in Computing Theory need to be integrated with practical ones in the curriculum at all levels, beginning with the first courses.

In general, students take a Computing Theory course when they have already carried out about 50% of the study program. Then, the contents cannot be presented as the cornerstone of CS/IS curricula. It also prevents students from relating early theoretical aspects with practical ones, developing an engineer attitude. Systems Engineering students must be able to translate problems into abstractions using formal models, manipulating them and reasoning about their properties in a rigorous way. It is intended that the student does not study these topics in an isolated course, mainly theoretical, where formalizing is an end in itself.

The Undergraduate Degree Program in Systems Engineering in our study program has incorporated "Computer Science I" as an introductory course in the theory of languages and automata. The main purpose of this course is to present an introduction to the study of computational processes and to explore their scope in the context of an automata hierarchy, with a suitable approach for first-year students. This course does not intend to substitute others with more theoretical focus,

[1] CIC (Comisión de Investigaciones Científicas de la Pcia. de Buenos Aires) lfavre@exa.unicen.edu.ar
[2] vmauco@exa.unicen.edu.ar
[3]PLADEMA –ISISTAN rbarbu@exa.unicen.edu.ar

but rather to give students basic concepts in Computer Science in early stages of their education. The idea is to motivate the study of the nature and limits of computation by means of concrete applications along the curriculum.

This paper has the following structure. Section 2 outlines the proposal. Section 3 describes the course contents. Section 4 presents the course context. Section 5 describes the methodology of the course and Section 6 an evaluation of the experience. Finally, conclusions are made.

2. MOTIVATION

Computing Theory should be an essential component of educational curricula in CS/IS. We believe students need to understand the laws of computation to make the best use of computing technology.

With the emergence of new computing paradigms (Artificial Intelligence, Decision Support Systems, Communications Networks, Security, DNA Computing, Molecular Computing, etc) new curricula should be developed that stress the interactions between Computing Theory and other disciplines.

Recent research in security has used important algorithmic questions arisen from both, the design of networks and their effective use. Modern AI students should understand the pragmatical implications of impossibility results and intractability results. With the growth in volume of online data, for example in databases and on the Internet, the focus of research in information retrieval has shifted to new applications in information management and decision support that demands asymptotically efficient algorithms. Activities such as Data Mining, Latent Semantic Indexing, On Line Processing, open directions for research in combinatorial algorithms, models, complexity and computability (Loui 1996).

It is our point of view that CS/IS students should analyze the nature and limits of computation earlier, because this analysis exposes students not only to the rich foundations of Computer Science but also it gives them a solid basis to address computational problems related to new technologies.

Based on the previous considerations, it was thought of implementing a first-year introductory course that included the bases of the theory of formal languages and automata, studying in depth and interrelating them in following courses. The intention of the course is to provide insights on essential questions about the nature of computation: What is an algorithm? What can be computed? When is a given algorithm intractable?.

Students learn how to create abstract models, to reason about their properties and to be able to apply them to practical problems. Models of languages and computers are developed to discover fundamental laws of computation and to provide abstractions for the design and implementation of algorithms. In summary, the course gathers and teaches recurrent concepts, which are fundamental to Computer Science in such a way that these concepts could be reused along students' formation.

Although the contents are the standard of any Computer Science course, this proposal is original in the level in which they are focused. The goal is to transmit these concepts to students whose mathematical background is still elementary, so that they can understand them at the same time they develop their mathematical skills.

Theoretical Computer Science is an interesting discipline with a wide range of applications. Usually, this is often hidden among definitions, theorems and demonstrations. This course intends to give students early basic concepts in Computer Science in a rigorous way without tedious demonstrations.

3. THE COURSE CONTENTS

The focus of the course is mainly practical. A hierarchy of abstract machines (finite automata, pushdown automata and Turing machines) and their computational power as language recognizers are analyzed. Grammars are also introduced as language generators.

The first unit introduces the mathematics of strings of symbols and languages. The second unit studies the regular languages. Formalisms associated to them are defined: finite automata, finite-state transducers, regular grammars and regular expressions. The notion of nondeterminism is also introduced together with the conversions of languages from one formalism to another. The conversions are nondeterministic finite automata (NFA) to deterministic finite automata (DFA), DFA to minimum state DFA, DFA to regular expression, regular expression to NFA, NFA to regular grammar and regular grammar to NFA. Besides, some properties of regular languages are studied. In addition, some problems outside programming languages processing scope are presented to enable students to widen the range of application of these automata and understand the utility of formal models as a tool for the design of solutions to practical problems.

The third unit considers the context-free languages and their relation with the pushdown automata (PDA) and the context-free grammars. PDA are defined, the design of deterministic and nondeterministic PDA is proposed, and the fact that both models are not equivalent is shown. The context-free grammars are introduced as a mechanism to generate the same languages that PDA can recognize. The notation BNF (Backus-Naur-Form) is also presented in order to familiarize students with it, since it will be possibly used in other courses to introduce the syntax of different programming languages, such as Pascal in the first year of the study program. Closure properties of context-free languages are investigated. In the forth unit the main topics are the Turing machines and the languages that they can recognize. This unit contains basic definitions and different versions of the deterministic Turing machines to translate languages or calculate functions (one-tape machine, multitape machine, and linear bounded automaton). The context-sensitive languages are presented as the ones recognized by the linear bounded automata and generated by the context-sensitive grammars.

In order to integrate the studied classes of languages, unit five introduces the hierarchy of languages and its correspondence with the hierarchy of the automata analyzed. It is intended that a student, given an arbitrary language, should design the most restrictive automaton and its corresponding grammar.

The last unit deals with computability in an introductory way. It is devoted to the inherent limitations of effectively executable algorithms, and hence of the computers that implement them. Basic concepts of temporal and spatial complexity are introduced with empirical evaluation of some theoretical results. Only basic definitions and examples are given.

4. THE CONTEX

The Undergraduate Degree Program in Systems Engineering lasts five years. The first three years correspond to the kernel of the basic formation and the remaining ones to the specialization period. The latter is organized in subject areas that represent specific domains of knowledge (Software Engineering, Signal Processing, Advanced Computer Architectures, etc).

Figure 1 shows the undergraduate study program. The course "Computer Science I" is dictated in the second semester of the first year. The necessary mathematical knowledge is provided in the compulsory Admission Course established in the curriculum. This course is aimed at providing students with basic mathematical background.

A course with these features should obviously be integrated with advanced courses that motivate a deeper analysis in a determined domain. The contents of "Computer Science I" are integrated with other courses. For example:

- During the second year, in "Analysis and Design of Algorithms I" and "Analysis and Design of Algorithms II" efficient algorithms related to languages processing are implemented. For example, algorithms that allow to obtain nondeterministic finite automata from regular expressions, that transform nondeterministic finite automata into deterministic ones, simple lexical analyzers, etc.
- In "Computer Science II" aspects of computability are again tackled.

	First Semester	Second Semester
First year	Admission Course	Computer Science I
	Introduction to Programming I	Introduction to Programming II
	Mathematical Analysis I	Geometry and Linear Algebra
	Algebra I	Physics
Second year	Computer Science II	Design and Analysis of Algorithms II
	Design and Analysis of Algorithms I	Data Communication I
	Introduction to Computer Architecture	Probability and Statistics
	Electricity and Magnetism	Digital Electronics
	Mathematical Analysis II	
Third year	Functional and Logic Programming Paradigms	Object Oriented Programming
	Data Structures	Databases
	Software Design Methodologies	Programming Languages
	Computer Architecture I	Operating Systems
		Operation Research
Forth year	Computer Architecture II	Software System Design
	Information Theory	Compilers
	Data Communication II	Specialization Period (12 credits)
	Advanced Mathematical Analysis	
	Specialization Period (4 credits)	
Fifth year	Software Engineering	Final Project
	Specialization Period (16 credits)	

Figure 1. Undergraduate Degree Curriculum in Systems Engineering.

- "Analysis and Design of Algorithms II" introduces the basis of the theory of computational complexity by means of the analysis of some problems related to the P and NP classes.
- In other courses of the kernel of the basic formation (for example, in "Software Design Methodologies", "Computer Architecture I", etc) the formal models analyzed in "Computer Science I" are used.
- The curriculum includes a compulsory course and an optional one in compiler design.
- Students can take up specific courses in computability and complexity through optional courses in the area of "Theoretical Computer Science".
- The Specialization Period includes, for example, Artificial Intelligence, Communication Networks, Information Retrieval and Visualization, and Computational Geometry courses.

5. THE METODOLOGY

The course is one-semester in length (70 hours). It consists of 2 hours of lectures and 3 hours of practical classes per week.

In general, the methodology followed to teach each of the contents mentioned in Section 3 begins by introducing the fundamental ideas informally by means of many examples, in order to motivate their use for practical modeling purposes. Then, the corresponding formal definitions are given.

To illustrate our approach we describe the way we teach, for example, Finite Automata. At this point, students have already learnt the concepts of alphabets, strings and languages, and they have faced the problem of determining if a string belongs to a given language. In this context, we present a language L over an alphabet A and ask the students if a string s belongs to this set. For example, $A = \{a, b\}$, $L = \{a^n b^{2k} / n, k > 0\}$ and $s = a^3b^2$. To answer this question we not only need to analyze the symbols which appear in s but also their relative positions. We construct a diagram which helps us to determine the different members of the language. This diagram has an input tape where the symbols of the string s are placed, some states, a reading head, a finite control, an initial state and a set of final states. To find out if s is in L, we simulate the evolution of the diagram from an initial state reading each successive symbol of s. If after consuming all the symbols a final state is reached, we say the string s belongs to language L. Otherwise, s does not belong to L. We repeat this process for many different test data. Thus, we present finite automata as a model of a machine, which accepts a particular set of words over some alphabet A. After this informal introduction, we formalize deterministic finite automata definition and we present deterministic finite automata as language recognizers and language transducers, emphasizing their important role in compiler construction. Besides, we give some examples of finite automata modeling "real-world" applications, like the vending machine, an elevator, traffic signals emulation, etc. Students are given enough time to solve many other exercises of increasing difficulty. When they are familiarized with these automata, we introduce nondeterministic finite automata, and we explain the algorithm to convert a nondeterministic finite automaton into the equivalent deterministic one. More exercises are provided to students in order to clarify this new concept. Approximately, 10 hours are used to teach the concepts mentioned above.

In each practical class, a brief theoretical-practical introduction is developed to help the students solve the corresponding practical work. In addition, sample solutions for selected exercises are shown. The students also have a specially prepared student's handbook, which contains notes for each one of the topics included in the practical works, exercises with step by step solutions, and detailed information of the bibliography to be used for each topic (Mauco 1998). This student's handbook was written to make the topics accessible to first-year students since, in general, traditional books (Aho 1995; Atallah 1999; Kelley 1995; Lewis 1998; Mandrioli 1987) are aimed at students of advanced courses, and thus they do not fit the proposed approach.

Students can work in an interactive way using a computer. Currently, JFLAP (Java Formal Languages and Automata Package) is being used in practical classes to provide visualization and interaction (Gramond 1999; Hung 2000). This tool is used to design and simulate deterministic and non-deterministic versions of finite automata, pushdown automata and Turing machines. Besides, it complements the concepts learned in class by giving students an alternative view, in addition to the theoretical representation, usually followed in textbooks. Moreover, interaction allows students to play an active role in the learning process, experimenting with the concepts to receive feedback.

Each student has to pass two examinations, a mid-term test and a final one. The mid-term test includes practical exercises similar to the practical works ones. The final exam evaluates the complete contents of the course. Figure 2 contains an example of a mid-term test and Figure 3 shows an example of a final test.

6. EXPERIENCE EVALUATION

Until 1996 the theory of automata and formal languages, computability and complexity were included in a one-semester third-year course named Theory of Computation. At that moment, students had already completed courses of programming, algorithms and data structures, comparative analysis of programming languages, analysis and design methodologies and computer architecture. These courses obviously made use of formal models, such as automata and grammars, without having analyzed them at a specific course that

COMPUTER SCIENCE I MID-TERM TEST

1) For each of the following languages over the alphabet {a, b, c, d}: a) Determine its type. b) Give the corresponding automaton and grammar. c) Give two examples of strings belonging to the language and show how they can be derived from the start symbol of the grammar.

$L_1 = \{ c^p a^j d^k b^{k+1} a^{2n} c^q / n > j \text{ and } k, p \geq 0 \text{ and } q, j > 0\}$

$L_2 = \{ x / x \in \{a, b, c\}^* \text{ and x has an even number of occurrences of the substring abc}\}$

2) Consider the following BNF for simplified regular expressions:

```
<reg_expr>   ::= <reg_expr> + <reg_term> | <reg_term>
<reg_term>   ::= <reg_term> . <reg_factor> | <reg_factor>
<reg_factor> ::= (<reg_expr>) | <reg_factor>* | a | b | ε
```

a) For the strings i) $(a + b)^* . a . a^*$ ii) $\varepsilon + a^{**} . b$ iii) $(( ) + b) + a$ construct complete derivation trees when possible. b) Select in a) one string described by the BNF and construct the minimum-state deterministic finite automaton, by using the algorithms studied in the course.

Figure 2. Mid-term test example

COMPUTER SCIENCE I FINAL TEST

1) Given the following languages over the alphabet {a, b, c, d}: a) Determine each language type. b) Choose a context-sensitive language and give its corresponding automaton and grammar. c) Choose a context-free language and give its corresponding grammar.

i) $L_1 = \{x / x \in \{a, b, c\}^* \text{ and x contains the substring } w. w^R \text{ where } w = abc \}$
ii) $L_2 = L_1^*$
iii) $L_3 = \{a^{n+1} b^{2i} c^{2n+i} d^n \ / \ i, n \geq 0\}$
iv) $L_4 = \{a^{n+1} b^k d^{i+2n} / i, n \geq 0 \text{ and } (k = 2i \text{ or } k = 2i+1)\}$

2) a) Define string, alphabet and language. b) By using sets, represent the relationships among all the types of languages studied. c) For each language type, define formally the corresponding automaton and grammar. d) ¿Which type of grammar is more restrictive: the sensitive-context grammar or the context-free one? Justify your answer.

3) a) Define i) recursive and recursively enumerable language; ii) decidable and undecidable problem. b) Define the Halting problem. Demonstrate which kind of problem it is.

4) a) List the closure properties of: i) regular languages; ii) context-free languages. b) Demonstrate the result of the following operation $L_1^R \cup L_2^*$, where L_1 is a context-free language and L_2 is a regular language.

Figure 3. Final test example

provided a general view. This caused content repetition. Moreover, although a great part of the students' basic formation had been already covered, the limitations of computational processes were still not perceived.

Students did not see the contents as a basis for solving problems, but as abstractions which they could have done without until that point. The lack of motivation was very high. During the course it was necessary to invest considerable time in exercises which students underestimated, as for example finite automata, pushdown automata and Turing machines design.

The experience in teaching these contents in a first-year course has shown that students' interest is higher. We consider that this is an important factor in any learning process. Students were comfortable with the topics and the methodology. Figure 4 shows the percentage of promoted students in the first-year courses for the last four years. It can be easily observed that the percentage of promoted students in "Computer Science I" is high and similar to the remaining courses.

In general, students find easy to identify a language type and to solve problems applying the taught models. However, they have difficulties in demonstrating properties about them.

Nowadays, the course is still being tested by following the students and their performance in advanced courses. It was perceived that students could solve different problems abstracting similar solutions based on the studied models. In addition, a survey was carried out in order to gather the students' comments in connection to the proposal. The task involved 120 students who had gone through "Computer Science I" in the last years. 90% of the students considered that the course could be taught in the first year. The survey also revealed that the course has the same difficulty degree as other first-year courses.

		First semester			Second semester			
	Number of Students	**Introduction to Programming I**	**Mathematical Analysis I**	**Algebra I**	**Physics**	**Geometry and Linear Algebra**	**Introduction to Programming II**	**Computer Science I**
	Enrolled	233	243	225	110	182	143	258
1996	Promoted	165	101	131	54	114	73	163
	% promotion	**70.82**	**41.56**	**58.22**	**49.09**	**62.64**	**51.05**	**63.18**
	Enrolled	228	237	240	168	200	175	227
1997	Promoted	139	106	147	122	148	89	148
	% promotion	**60.96**	**44.73**	**61.25**	**72.62**	**74.00**	**50.86**	**65.20**
	Enrolled	222	261	232	224	169	151	205
1998	Promoted	115	169	141	204	110	117	157
	% promotion	**51.80**	**64.75**	**60.77**	**91.07**	**65.09**	**77.48**	**76.59**
	Enrolled	259	234	241	123	210	186	218
1999	Promoted	182	103	190	99	162	122	163
	% promotion	**70.27**	**44.02**	**78.84**	**80.49**	**77.14**	**65.59**	**74.77**

Figure 4. Comparison of % of promotion in first-year courses

7. CONCLUSIONS

An introductory course in Computer Science intended for first-year students was described. It introduces the basic contents of the theory of computation starting from the study of a hierarchy of abstract machines. The difference with the traditional courses is not in the contents, but in the stage of study when it is taught and in the methodology followed to teach the topics.

The course proposed is a good starting point to make students aware of the basic notions of language processing and computational process limitations in the early stages of their formation.

Our intention has been to bring soon a solid foundation in computing theory for further studies along the study program.

The experience has been satisfactory. We believe that it could be reproduced in any CS/IS study program. The condition would be that students attended a basic course in discrete mathematics. Another alternative would be to incorporate these contents at the beginning of the course.

8. REFERENCES

ACM Curricula Recommendation, 1991, "Computing Curricula 1991", ACM/IEEE-CS. Available in www.acm.org/education/curricula.html

Aho, A. and J. Ullman, 1995, Foundations of Computer Science, Computer Science Press.

Atallah, M., 1999, Algorithms and Theory of Computation HandBook, CRC Pr.

Gramond, E. and S. Rodger, 1999, "Using JFLAP to Interact With Theorems in Automata Theory", Thirtieth SIGCSE Technical Symposium on Computer Science Education, pp. 336-340.

IS' 97, 1997, Model Curriculum and Guidelines for Undergraduate Degree Programs in Information Systems, Association for Computing Machinery (ACM), Association for Information Systems (AIS) and Association of Information Technology Professionals (AITP). Available in www.acm.org/education/curricula.html

Hung, T. and S. Rodger, 2000, "Increasing Visualization and Interaction in the Automata Theory Course", Available in www.cs.duke.edu/~rodger/papers

Kelley, D., 1995, Teoría de Autómatas y Lenguajes Formales, Prentice Hall.

Lewis, H. and C. Papadimitriou, 1998, Elements of the Theory of Computation, Second Edition, Prentice Hall.

Loui, M. et al., 1996, Strategic Directions in Research in Theory Computing, ACM Computing Surveys, Vol 28, 4, pp. 575-590.

Mandrioli, D. and C. Ghezzi, 1987, Theoretical Foundations of Computer Science, John Wiley and Sons.

Mauco, V. and R. Barbuzza, 1998, Notes for Computer Science I (in Spanish) Facultad de Cs. Exactas, Universidad Nacional del Centro de la Pcia. de Buenos Aires.

MSIS' 2000, 2000, Model Curriculum and Guidelines for Graduate Degree Programs in Information Systems, Association for Computing Machinery (ACM), Association for Information Systems (AIS). Available in www.acm.org/education