Proceedings of the Workshop on Effective Tools and Methodologies for Teaching Natural Language Processing and Computational Linguistics, Philadelphia, July 2002, pp. 71-76. Association for Computational Linguistics.

Formal Language Theory for Natural Language Processing

Shuly Wintner
Computer Science Department
University of Haifa
Haifa 31905, Israel
shuly@cs.haifa.ac.il

Abstract

This paper reports on a course whose aim is to introduce Formal Language Theory to students with little formal background (mostly linguistics students). The course was first taught at the European Summer School for Logic, Language and Information to a mixed audience of students, undergraduate, graduate and postgraduate, with various backgrounds. The challenges of teaching such a course include preparation of highly formal, mathematical material for students with relatively little formal background; attracting the attention of students of different backgrounds; preparing examples that will emphasize the practical importance of material which is basically theoretical; and evaluation of students' achievements.

1 Overview

Computational linguistics students typically come from two different disciplines: Linguistics or Computer Science. As these are very different paradigms, it is usually necessary to set up a common background for the two groups of students. One way to achieve this goal is by introducing the core topics of one paradigm to students whose background is in the other. This paper reports on such an experiment: teaching Formal Language Theory, a core computer science subject, to students with no background in computer science or mathematics. The course was first taught at the 13th European Summer School in Logic, Language and Information (Helsinki, Finland) in the summer of 2001.

While formal language theory is not a core computational linguistics topic, it is an essential prerequisite for a variety of courses. For example, regular expressions and finite-state technology are instrumental for many NLP applications, including morphological analyzers and generators, part-of-speech taggers, shallow parsers, intelligent search engines etc. The mathematical foundations of context-free grammars are necessary for a thorough understanding of natural language grammars, and a discussion of the Chomsky hierarchy is mandatory for students who want to investigate more expressive linguistic formalisms such as unification grammars.

The motivation for teaching such a course to students with no background in formal methods, especially linguists, stems from the observation that many students with background in linguistics are interested in computational linguistics but are overwhelmed by the requirements of computational linguistics courses that are designed mainly for computer science graduates. Furthermore, in order to establish a reasonable level of instruction even in introductory computational linguistics courses, I found it essential to assume a firm knowledge of basic formal language theory. This assumption does not hold for many non-CS gradutes, and the course described here is aimed at such students exactly.

The challenges of teaching such a course are many. Teaching at the European Summer School is always a challenge, as this institution attracts students from a variety of disciplines, and one never knows what background students in one's class will have. In this particular case, the course was advertised as a *foundational* computation course. Foundational courses presuppose absolutely no background knowledge, and should especially be accessible to people from other disciplines. The material had to be prepared in a way that would make it accessible to students of linguistics, for example, who might possess no knowledge of mathematics beyond high-school level.

Another characteristic of the European Summer Schools is that the students' education levels vary greatly. It is not uncommon to have, in one class, undergraduate, graduate and post-graduate students. This implies that the level of addressing the class has to be very delicately determined: it is very easy to bore most students or to speak over their heads. An additional difficulty stems from the fact that while the language of instruction at the Summer School is English, most participants (students and lecturers alike) are not native speakers of English.

Undoubtedly the greatest challenge was to prepare the course in a way that will attract the attention of the class. Formal language theory is a highly theoretical, mostly mathematical subject. Standard textbooks (Hopcroft and Ullman, 1979; Harrison, 1978) present the material in a way that will appeal to mathematicians: very formal, with one subject built on top of its predecessor, and with very formal (if detailed) examples. Even textbooks that aim at introducing it to non-mathematicians (Partee et al., 1990) use mostly examples of formal (as opposed to natural) languages. In order to motivate the students, I decided to teach the course in a way that emphasizes natural language processing applications, and in particular, to use only examples of natural languages.

While this paper focuses on a particular course, taught at a particular environment, I believe that the lessons learned while developing and teaching it are more generally applicable. A very similar course can be taught as an introduction to NLP classes in institutions whose majority of students come from computer science, but who would like to attract linguistics (and other non-CS) graduates and provide them with the necessary background. I hope that the examples given in the paper will prove useful for developers of such courses. More generally, the paper demonstrates a gentle approach to formal, mathematical material that builds on terminology familiar to its audience, rather than use the standard mathematical paradigm in teaching. I believe that this approach can be useful for other courses as well.

2 Structure of the course

Courses at the Summer School are taught in sessions of 90 minutes, on a daily basis, either five or ten days. This course was taught for five days, totaling 450 minutes (the equivalent of ten academic hours, approximately one third of the duration of a standard course). However, the daily meetings eliminate the need to recapitulate material, and the pace of instruction can be enhanced.

I decided to cover a substantial subset of a standard Formal Language Theory course, starting with the very basics (e.g., set theory, strings, relations etc.), focusing on regular languages and their computational counterpart, namely finite-state automata, and culminating in context-free grammars (without their computational device, push-down automata). I sketch the structure of the course below.

The course starts with a brief overview of essential set theory: the basic notions, such as sets, relations, strings and languages, are defined. All examples are drawn from natural languages. For example, *sets* are demonstrated using the vowels of the English alphabet, or the articles in German. Set operations such as *union* or *intersection*, and set relations such as *inclusion*, are demonstrated again using subsets of the English alphabet (such as vowels and consonants). *Cartesian product* is demonstrated in a similar way (example 1) whereas relations, too, are exemplified in an intuitive manner (example 2). Of course, it is fairly easy to define *strings*, *languages* and operations on strings and languages – such as *concatenation, reversal, exponentiation, Kleene-closure* etc. – using natural language examples.

The second (and major) part of the course discusses *regular languages*. The definitions of *regular expressions* and their denotations are accompanied by the standard kind of examples (example 3). After a brief discussion of the mathematical properties of regular languages (in particular, some *closure properties*), *finite-state automata* are gently introduced. Following the practice of the entire course, no mathematical definitions are given, but a rigorous textual description of the concept which is accompanied by several examples serves as a substitute to a standard definition. Very simple automata, especially extreme cases (such as the automata accept-

Example 1 Cartesian product

Let A be the set of all the vowels in some language and B the set of all consonants. For the sake of simplicity, take A to be {*a, e, i, o, u*} and B to be {*b, d, f, k, l, m, n, p, s, t*}. The Cartesian product $B \times A$ is the set of all possible consonant–vowel pairs: $\{\langle b,a\rangle, \langle d,a\rangle, \langle d,i\rangle, \langle k,o\rangle, \langle p,o\rangle, \langle t,e\rangle, \langle t,u\rangle, \ldots\}$, etc. Notice that the Cartesian product $A \times B$ is different: it is the set of all vowel–consonant pairs, which is a completely different entity (albeit with the same number of elements). The Cartesian product $B \times B$ is the set of all possible consonant–consonant pairs, whereas $A \times A$ is the set of all possible diphthongs.

Example 2 Relation

Let A be the set of all articles in German and B the set of all German nouns. The Cartesian product $A \times B$ is the set of all article–noun pairs. Any subset of this set of pairs is a relation from A to B. In particular, the set $R = \{\langle x, y\rangle \mid x \in A$ *and* $y \in B$ *and* x *and* y *agree on number, gender and case*$\}$ is a relation. Informally, R holds for all pairs of article–noun which form a grammatical noun phrase in German: such a pair is in the relation if and only if the article and the noun agree.

ing the empty language, or Σ^*), are explicitly depicted. *Epsilon-moves* are introduced, followed by a brief discussion of *minimization* and *determinization*, which is culminated with examples such as 4.

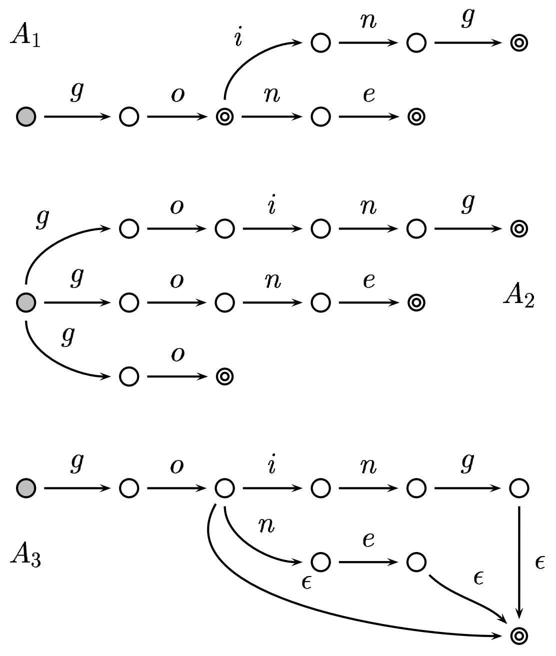
Example 3 Regular expressions

Given the alphabet of all English letters, $\Sigma = \{a, b, c, \ldots, y, z\}$, the language Σ^* is denoted by the regular expression Σ^* (recall our convention of using Σ as a shorthand notation). The set of all strings which contain a vowel is denoted by $\Sigma^* \cdot (a+e+i+o+u) \cdot \Sigma^*$. The set of all strings that begin in "*un*" is denoted by $(un)\Sigma^*$. The set of strings that end in either "*tion*" or "*sion*" is denoted by $\Sigma^* \cdot (s+t) \cdot (ion)$. Note that all these languages are infinite.

To demonstrate the usefulness of finite-state automata in natural language applications, some operations on automata are directly defined, including *concatenation* and *union*. Finally, automata are shown to be a natural representation for *dictionaries* and *lexicons* (example 5).

Example 4 Equivalent automata

The following three finite-state automata are equivalent: they all accept the set {*go, gone, going*}.



Note that A_1 is deterministic: for any state and alphabet symbol there is at most one possible transition. A_2 is not deterministic: the initial state has three outgoing arcs all labeled by g. The third automaton, A_3, has ϵ-arcs and hence is not deterministic. While A_2 might be the most readable, A_1 is the most compact as it has the fewest nodes.

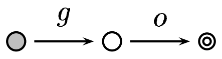
This part of the course ends with a presentation of *regular relations* and *finite-state transducers*. The former are shown to be extremely common in natural language processing (example 6). The latter are introduced as a simple extension of finite-state automata. Operations on regular relations, and in particular *composition*, conclude this part (example 7).

The third part of the course deals with *context-free grammars*, which are motivated by the inability of regular expressions to account for (and assign structure to) several phenomena in natural languages. Example 8 is the running example used throughout this part.

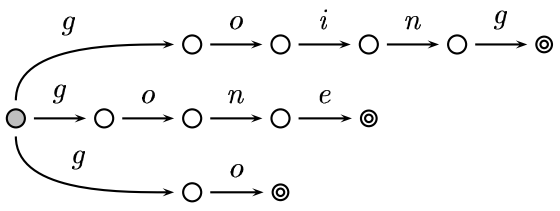
Basic notions, such as *derivation* and *derivation*

Example 5 Dictionaries as finite-state automata

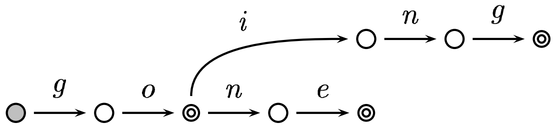
Many NLP applications require the use of lexicons or dictionaries, sometimes storing hundreds of thousands of entries. Finite-state automata provide an efficient means for storing dictionaries, accessing them and modifying their contents. To understand the basic organization of a dictionary as a finite-state machine, assume that an alphabet is fixed (we will use $\Sigma = \{a, b, \ldots, z\}$ in the following discussion) and consider how a single word, say *go*, can be represented. As we have seen above, a naïve representation would be to construct an automaton with a single path whose arcs are labeled by the letters of the word *go*:



To represent more than one word, we can simply add paths to our "lexicon", one path for each additional word. Thus, after adding the words *gone* and *going*, we might have:



This automaton can then be determinized and minimized:



With such a representation, a lexical lookup operation amounts to checking whether a word w is a member in the language generated by the automaton, which can be done by "walking" the automaton along the path indicated by w. This is an extremely efficient operation: it takes exactly one "step" for each letter of w. We say that the time required for this operation is *linear in the length of* w.

Example 6 Relations over languages

Consider a simple part-of-speech tagger: an application which associates with every word in some natural language a tag, drawn from a finite set of tags. In terms of formal languages, such an application implements a relation over two languages. For simplicity, assume that the natural language is defined over $\Sigma_1 = \{a, b, \ldots, z\}$ and that the set of tags is $\Sigma_2 = \{PRON, V, DET, ADJ, N, P\}$. Then the part-of-speech relation might contain the following pairs, depicted here vertically (that is, a string over Σ_1 is depicted over an element of Σ_2):

I	know	some	new	tricks	
PRON	V	DET	ADJ	N	
said	the	Cat	in	the	Hat
V	DET	N	P	DET	N

As another example, assume that Σ_1 is as above, and Σ_2 is a set of part-of-speech and morphological tags, including $\{$*-PRON, -V, -DET, -ADJ, -N, -P, -1, -2, -3, -sg, -pl, -pres, -past, -def, -indef*$\}$. A morphological analyzer is basically an application defining a relation between a language over Σ_1 and a language over Σ_2. Some of the pairs in such a relation are (vertically):

I	know	
I-PRON-1-sg	know-V-pres	
some	new	tricks
some-DET-indef	new-ADJ	trick-N-pl
said	the	Cat
say-V-past	the-DET-def	cat-N-sg

Finally, consider the relation that maps every English noun in singular to its plural form. While the relation is highly regular (namely, adding "*s*" to the singular form), some nouns are irregular. Some instances of this relation are:

cat	hat	ox	child	mouse	sheep
cats	hats	oxen	children	mice	sheep

trees are presented gently, with plenty of examples. To motivate the discussion, questions of *ambiguity* are raised. Context-free grammars are shown to be sufficient for assigning structure to several natural language phenomena, including subject-verb *agreement*, verb *subcategorization*, etc. Finally, some mathematical properties of context-free languages are discussed.

The last part of the course deals with questions of expressivity, and in particular strong and weak

Example 7 Composition of finite-state transducers

Let R_1 be the following relation, mapping some English words to their German counterparts:

$R_1 = \{$*tomato:Tomate, cucumber:Gurke, grapefruit:Grapefruit, grapefruit:pampelmuse, pineapple:Ananas, coconut:Koko, coconut:Kokusnuß*$\}$

Let R_2 be a similar relation, mapping French words to their English translations:

$R_2 = \{$*tomate:tomato, ananas:pineapple, pampelmousse:grapefruit, concombre:cucumber, cornichon:cucumber, noix-de-coco:coconut*$\}$

Then $R_2 \circ R_1$ is a relation mapping French words to their German translations (the English translations are used to compute the mapping, but are not part of the final relation):

$R_2 \circ R_1 = \{$*tomate:Tomate, ananas:Ananas, pampelmousse:Grapefruit, pampelmousse:Pampelmuse, concombre:Gurke, cornichon:Gurke, noix-de-coco:Koko, noix-de-coco:Kokusnuße*$\}$

Example 8 Rules

Assume that the set of terminals is $\{$*the, cat, in, hat*$\}$ and the set of non-terminals is $\{$*D, N, P, NP, PP*$\}$. Then possible rules over these two sets include:

$D \rightarrow$ *the*	$NP \rightarrow D\ N$
$N \rightarrow$ *cat*	$PP \rightarrow P\ NP$
$N \rightarrow$ *hat*	$NP \rightarrow NP\ PP$
$P \rightarrow$ *in*	

Note that the terminal symbols correspond to words of English, and not to letters as was the case in the previous chapter.

generative capacity of linguistic formalism. The *Chomsky hierarchy of languages* is defined and explained, and substantial focus is placed on determining the location of natural languages in the hierarchy. By this time, students will have obtained a sense of the expressiveness of each of the formalisms discussed in class, so they are more likely to understand many of the issues discussed in Pullum and Gazdar (1982), on which this part of the course is based. The course ends with hints to more expressive formalisms, in particular Tree-Adjoining Grammars and various unification-based formalisms.

3 Enrollment data

While the Summer School does not conduct teaching evaluations, I felt that it would be useful to receive feedback from participants of the course. To this end, I designed a standard teaching evaluation form and asked students to fill it in on the last class. The data in this section are drawn from the students' responses.

The number of students who submitted the questionnaire was 52. Nationality was varied, with the majority from Finland, Poland, Italy, Germany, the United Kingdom and the United States, but also from Canada, the Netherlands, Spain, Greece, Romania, France, Estonia, Korea, Iran, the Ukraine, Belgium, Japan, Sweden, Russia and Denmark. Thirty six defined themselves as graduate students, thirteen as undergraduates and three as post-PhD.

The most interesting item was background. Participants had to describe their backgrounds by choosing from Linguistics, Mathematics, Computer Science, Logic or Other. Only 32% described their background as Linguistics; 29% chose Computer Science; 21% chose Mathematics; and 15% — Logic. Other backgrounds included mostly Philosophy but also Biology and Physics. Why students of Computer Science, and in particular graduate students, should take Formal Language Theory in such an interdisciplinary Summer School is unclear to me.

Students were asked to grade their impression of the course, on a scale of 1–5, along the following dimensions:

- The course is interesting
- The course covers important and useful material
- The course progresses at the right pace
- The course is fun

The average grade was 4.53 for the interest question; 4.47 for the usefulness question; 3.67 for the pace question; and 4.13 for fun. These results show that participants felt that the course was interesting and useful, and even fun. However, many of them felt that it did not progress in the right pace. This might be partially attributed to the high rate of computer science and mathematics students in the audience: many of them must have seen the material earlier, and felt that progress was too slow for them.

4 Conclusions

This paper demonstrates that it is possible to teach formal, mathematical material to students with little or no formal background by introducing the material gently, albeit rigorously. By the end of the course, students with background in linguistics or philosophy are able to understand the computer science theoretical foundations underlying many aspects of natural language processing, in particular finite-state technology and formal grammars. This sets up a common background for more advanced classes in computational linguistics.

The course was taught once at an international, interdisciplinary summer school. I intend to teach it again this summer in a similar, albeit smaller event; I also intend to teach it to graduate Humanities students who express interest in computational linguistics, in order to introduce them to some foundational theoretical aspects of computer science essential for working on natural language processing applications. The positive reaction of most students to the course is an encouraging incentive to develop more courses along the same lines.

Acknowledgments

I wish to extend my gratitude to my students at ESSLLI-2001, who made teaching this course such an enjoyable experience for me. I am grateful to the reviewers for their useful comments. This work was supported by the Israeli Science Foundation (grant no. 136/1).

References

Michael A. Harrison. 1978. *Introduction to formal language theory*. Addison-Wesley, Reading, MA.

John E. Hopcroft and Jeffrey D. Ullman. 1979. *Introduction to automata theory, languages and computation*. Addison-Wesley Series in Computer Science. Addison-Wesley Publishing Company, Reading, Mass.

Brabara H. Partee, Alice ter Meulen, and Robert E. Wall. 1990. *Mathematical Methods in Linguistics*, volume 30 of *Studies in Linguistics and Philosophy*. Kluwer Academic Publishers, Dordrecht.

Geoffrey K. Pullum and Gerald Gazdar. 1982. Natural languages and context-free languages. *Linguistics and Philosophy*, 4:471–504.