An Educational Ontology for Formal Languages and Compilers

Mihaela Oprea[1]

(1) Petroleum-Gas University of Ploiesti, Department of Automatic Control, Computers and Electronics
Bdul Bucuresti, No 39, Ploiesti, 100680, ROMANIA
E-mail: mihaela[at]upg-ploiesti.ro

Abstract

Two fundamental courses for Computer Science and Informatics specializations are Formal languages and Compilers. Other courses are based on them, such as, computer programming, programming languages, artificial intelligence, natural languages processing, intelligent interfaces etc. Identifying and defining the basic and advanced concepts from the domains of Formal languages and Compilers as well as the relationships between the concepts under the form of an educational ontology can provide useful knowledge to be shared by undergraduate students as well as by computer science and informatics specialists. The paper presents an educational ontology for the domains of Formal Languages and Compilers that was developed with Protégé, in the OWL format, and some potential applications.

Keywords: Educational ontology, Formal languages, Translators, Compilers

1 Introduction

Formal languages and Compilers are two important disciplines of study for the Computer Science and Informatics student specializations. Some more advanced courses taught at undergraduate or master level are based on them, such as Artificial intelligence, Natural language processing, Human-computer interaction, Intelligent interfaces, Pattern recognition. Examples of specific topics are: compiler-based analysis for software engineering, software security and speech recognition. Providing computer-based educational resources that contain knowledge from the formal languages and compilers fields, to be shared by students it is imperative for the development of efficient e-learning platforms and intelligent tutoring systems in the Computer Science domain. Educational ontologies offer one of the best solutions for knowledge representation. In this paper it is presented a prototype educational ontology, Onto-FormalLanguages-Compilers-1, that we have developed in Protégé 4.3, for the course of Formal languages and Compilers that is taught to undergraduate Computer Science students at Petroleum-Gas University of Ploiesti.

The paper is organized as follows. Section 2 presents briefly some basic issues from the formal languages and compilers domains. The methodology that was followed for the ontology development as well as the ontology itself are described in section 3. The final section concludes the paper and highlights some future work.

2 Formal Languages and Compilers

Languages (natural or artificial) facilitate communication. Natural languages are used for the communication between humans, while programming languages (which are artificial languages) are used for the communication between humans and computers, or between computers (e.g.

connected in a computer network). Formal languages theory provides various methods for representing the structure of a language (i.e. its syntactic form). The main formalisms that can specify a language are: formal grammars, automata and regular expressions.

Basically, a language is a set of words composed with symbols from an alphabet, according to some rules that define well-formed words. A grammar is composed of an alphabet (defined as a set of terminal and non-terminal symbols), a start symbol and a set of production rules that are used for the derivation of well-formed words. There are various types of grammars. According to Chomsky, formal grammars are classified in four main classes: 0-type grammars (general grammars), 1-type grammar (context-sensitive grammars), 2-type grammar (context-free grammars) and 3-type grammars (regular grammars). Correspondingly, formal languages are classified in general languages, context-sensitive languages, context-free languages and regular languages.

The automata theory is the basis for the formal language theory and it can be applied, for example, to compilers design and text processing. There are different types of automata, as for example, finite automata, push-down automata, and translation automata.

Compilers are fundamental software tools for programming languages, in particular. In general, a compiler is composed of a lexical analyzer (scanner), a syntactic analyzer (parser), a semantic analyzer, an intermediate code generator, a module for code optimization, and an object code generator. A programming language compiler takes as input a program written in the specific programming language, performs an extended analysis (lexical, syntactical and semantically) and in case no errors occur it continue with a synthesis phase at the end of which it is generated the translation of the program in the object code. If some errors occur (e.g. syntactic or semantic errors) they must be corrected in order to continue with the synthesis phase.

The main domains that are fundamental for the Formal languages and Compilers fields are represented in Figure 1.

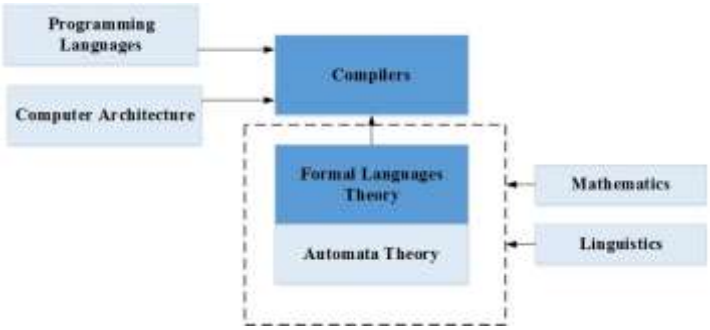

Figure 1. Fundamental domains for the Formal languages and Compilers fields

These domains are Automata theory, Mathematics, Linguistic, Programming languages and Computer architecture. Automata theory and Formal languages theory are applied in the Compiler field (see e.g. a practical example in (de Graaf, 2017))

The importance of Compilers course for undergraduate students is highlighted in (Hall et al., 2009). The authors argue that compiler algorithms are educational value for students and various applications can illustrate their use, as for example, virus detection based on compiler technology.

Formal languages and compilers have several real world applications, as for example: compilers designed for industrial network processors (see e.g. (Wagner and Leupers, 2001), computer and network security (Sassaman et al., 2013), pattern recognition (see e.g. the applications described in (Martín-Vide et al., 2004)). Some educational software tools were

developed for teaching formal languages and automata (see. e.g. the FLUTE system (Devedzic et al., 2000), an intelligent tutoring system, and FLApp (Pereira and Terra, 2018), a mobile application.

3 The OWL Onto-FormalLanguages-Compilers-1 Ontology

3.1 The methodology

The educational ontology, Onto-FormalLanguages-Compilers-1 was developed by following a methodology that integrates some basic guidelines reported in the literature as e.g. the specific frameworks for ontology engineering described in (Mizoguchi, 2004) and the guidelines for collaborative ontology development for higher education presented in (Oprea, 2016).

The main steps of the applied methodology are given below.

Methodology

Input: Formal languages and Compilers course specification and course resources
Output: Onto-FormalLanguages-Compilers-1 educational ontology

Step 1. Identify the main domains related to the course of *Formal languages and Compilers*;
Step 2. Identify, define and characterize the basic and advanced concepts of each domain identified in *Step 1*, used in the course of *Formal languages and Compilers*;
Step 3. Identify, define and characterize the relations between the concepts that were identified in *Step 2*;
Step 4. Develop the taxonomy and the derived ontology by using the concepts and relations identified in *Step 2* and *Step 3*;
Step 5. Implement and test the ontology with an ontology editor or ontology development software tool that was chosen.

The course specification includes course title and level, pre-requisite courses, year of study, number of hours/week for course teaching and laboratory work. The main course resources are textbooks, books, research papers, lecture notes (as e.g PowerPoint slides, PDF files) and software tools. The output of the methodology is the educational ontology for the Formal Languages and Compilers course, Onto-FormalLanguages-Compilers-1, that is composed of two sub-ontologies, Onto-FormalLanguages and Onto-Compilers, and a related sub-ontology, Onto-ProgrammingLanguages.

The course specification and the main resources of the *Formal languages and Compilers* course are:

Course specification:

- Course title: *Formal languages and Compilers*;
- Course level: undergraduate;
- Year of study: fourth year, first semester;
- Prerequisite courses: *Programming languages*; *Computer architecture*; *Data structures and algorithms*;
- Number of hours/week for course teaching and laboratory work: 2 hours/week - course teaching and 2 hours/week - laboratory work.

Course main resources:

Textbooks:

- (Aho et al., 2007) for basic and advanced concepts of compilers;
- (Șerbănați, 1987) for basic knowledge of programming languages and compilers design;
- (Grune et al., 2012) for basic knowledge on modern compilers design;

- (Martín-Vide et al., 2004) for basic knowledge of formal languages and applications;

Course lecture notes:

- PowerPoint slides for 2019-2020 academic year (first semester);

Other educational resources (research papers, technical reports etc)

- (Taylor and Moore, 2006) - an adaptive programming languages tutor;
- (Wagner and Leupers, 2001) - application example (C compiler design);
- (de Graaf, 2017) - practical use of automata and formal languages in compilers design;
- (Oprea, 2020) - compiler implementation in Prolog and syntactic analyzer implemented in Haskell;

Software tools (programming languages):

- C++/Java and optional Prolog/Haskell.

Examples of identified concepts and relations will be given in the next section.

3.2 The prototype ontology

We have designed a prototype educational ontology for the *Formal languages and Compilers* course taught at Petroleum-Gas University of Ploiesti by following the steps of the methodology described in the previous section, and we have implemented the resulted ontology in Protégé as an OWL ontology.

The ontology implementation was performed in Protégé 4.3 under the OWL format. Each identified concept and relation of the ontology was defined as a class in Protégé. Figure 2 shows a screenshot with some concepts that were included in the prototype ontology.

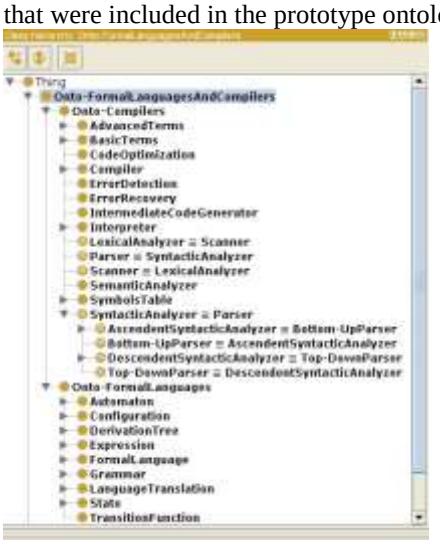

Figure 2. Screenshot with some concepts of the Onto-FormalLanguages-Compilers-1 prototype ontology (Protégé 4.3)

Examples of concepts that were defined are:

- in the Onto-FormalLanguages sub-ontology: FormalLanguage, Language, Alphabet, Grammar, Automaton, DerivationTree etc;
- in the Onto-Compilers sub-ontology: LexicalAnalyzer (synonym with Scanner), SyntacticAnalyzer (synonym with Parser), SemanticAnalyzer, CodeOptimization etc;

- in the Onto-ProgrammingLanguages ontology: Variable, Identifier, Statement, Declaration, DataType, Scope, ParameterTransfer, Procedure, Function etc.

We have defined some data properties (e.g. *grammarType*, *languageType*, *automatonType*, *arrayDimension*, *functionArity*, *variableName*, *stringLength*) and the relations between concepts as object properties. Our ontology uses the implicit relations between classes (i.e. the taxonomic relations *is_a* and *has*) that are provided by Protégé and some new relations as for example, *specifiedBy*, *hasArity*, *hasGrammar*, *part-1-of-a-compiler*, *hasIdentifier*.

Figure 3 presents examples of concepts and relations between them from the Onto-Compilers sub-ontology.

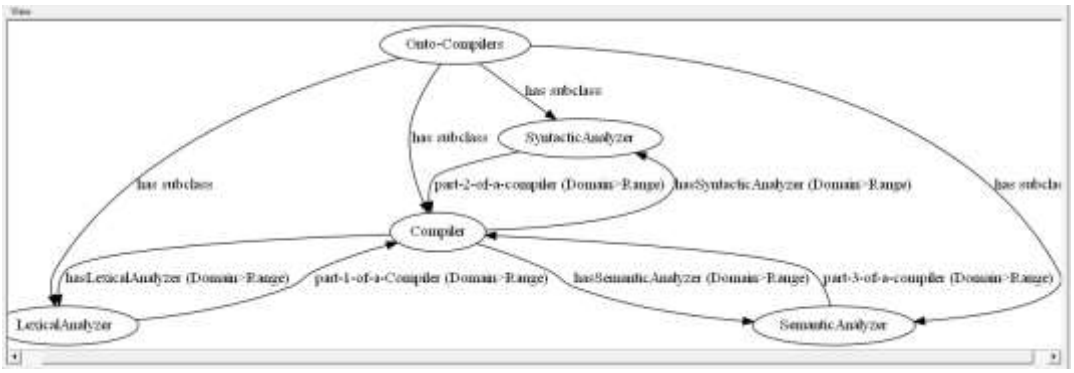

Figure 3. Graph with some concepts and relations from the Onto-Compilers sub-ontology

Some examples of concepts and relations between them from the Onto-FormalLanguages sub-ontology are given in Figure 4.

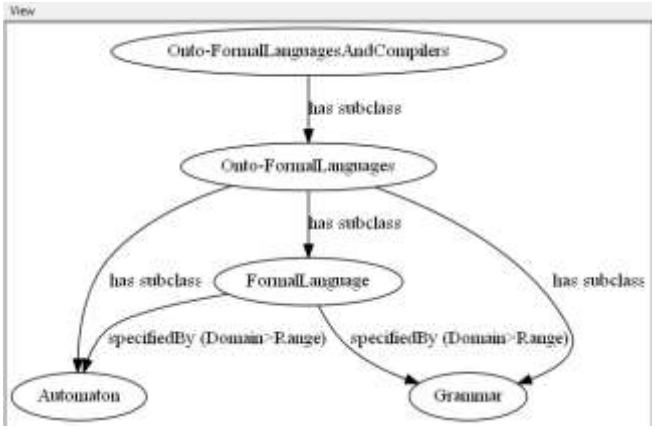

Figure 4. Graph with some concepts and relations from the Onto-FormalLanguages sub-ontology

A taxonomic tree (in OWL Viz, Protégé 4.3) for Onto-Compilers sub-ontology is shown in Figure 5.

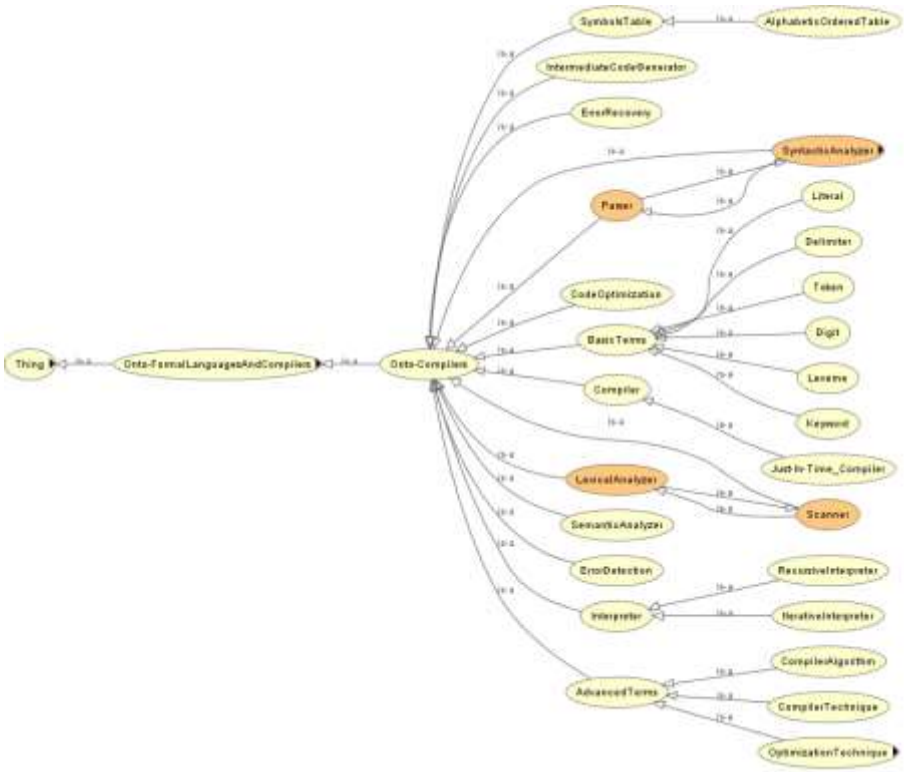

Figure 5. Taxonomic tree for the Onto-Compilers sub-ontology (in OWL Viz)

Another example of taxonomic tree from the Onto-FormalLanguages sub-ontology is presented in Figure 6.

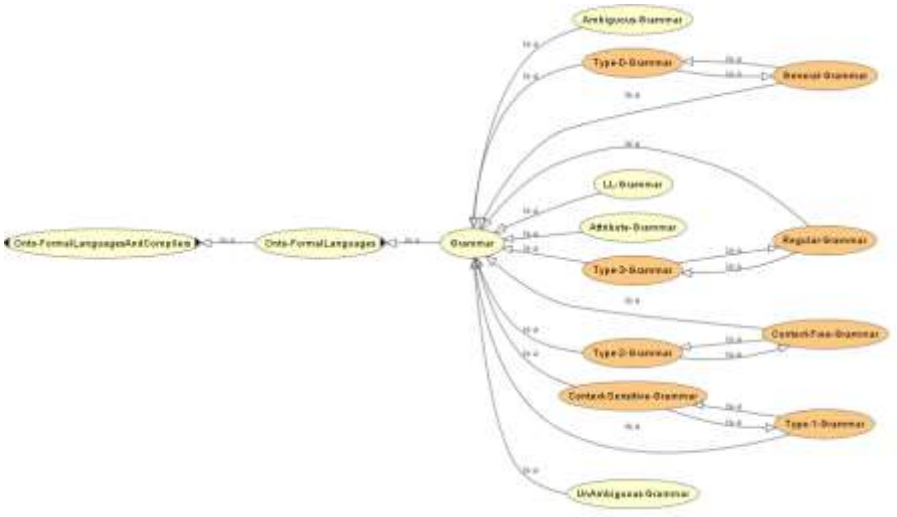

Figure 6. Taxonomic tree for the Onto-FormalLanguages sub-ontology (in OWL Viz)

The educational ontology was developed to be used in a first stage by students as a vocabulary of terms for the course of Formal languages and Compilers. Other examples of applications for which the ontology can be used are:

- compilers testing (see a recent reported example in (Li and Wotawa, 2020));
- program analysis (see an example in (Zhao et al., 2016));
- educational annotations (see an example in (Gayoso-Cabada et al., 2019));
- the examples discussed at the end of section 2.

Conclusion and Future Work

The paper presented an OWL prototype educational ontology for the course of Formal languages and Compilers, Onto-FormalLanguages-Compilers-1, that can be used by undergraduate students of the Computer Science specialization. The ontology was developed by following the guidelines of a methodology that was described in section 3 and was implemented in Protégé 4.3.

As a future work we intend to extend the ontology with more concepts related to the applications that were highlighted at the end of the third section of the paper.

References

Aho A., Lam M.S., Sethi R., Ullman J.D. (2007): *Compilers: Principles, Techniques and Tools*. 2nd edition, Pearson.

Grune D., van Reeuwijk K., Bal H.E., Jacobs C.J.H., Langendoen K. (2012): *Modern compiler design*, 2nd edition, Springer.

Martín-Vide C., Mitrana V., Păun Gh. (2004): *Formal languages and applications*, Springer.

Oprea M. (2020): *Logic programming and functional programming – Theory and applications*. In Romanian. MatrixRom Publishing House, Bucharest.

Șerbănați, L.-D. (1987): *Programming languages and compilers*. In Romanian. Romanian Academy Publishing House, Bucharest.

Mizoguchi, R. (2004): Ontology engineering environments. In S. Staab and R. Studer (Eds): *Handbook on Ontologies*. Springer.

Devedzic V., Debenham J., Popovic D. (2000): Teaching formal languages by an intelligent tutoring system. *Educational Technology & Society 3*, 2.

Gayoso-Cabado J., Goicoechea-de-Jorge M., Gómez-Albarrán M., Sanz-Cabrerizo A., Sarasa-Cabezuelo A., Sierra J. -L. (2019): Ontology-enhanced educational annotation activities. *Sustainability 11*, 4455.

Hall M., Padua D., Pingali K. (2009): Compiler research: the next 50 years. *Communications of the ACM 52*, 2, 60-67.

Pereira C.H. and Terra R. (2018): A mobile application for teaching formal languages and automata. *Computer Applications in Engineering Education 26*, 5, 1742-1752.

Oprea, M. (2016): A case study of collaborative ontology development for higher education. *International Journal of Artificial Intelligence 14*, 2, 70-97.

Sassaman L., Patterson M.L., Bratus S., Locasto M.E. (2013): Security applications of formal language theory. *IEEE Systems Journal 7*, 3, 489-500.

Conference Proceedings:

Li Y. and Wotawa F. (2020): On using ontologies for testing compilers. *Proc. of the IEEE Int. Conf. on Software Testing, Verification and Validation Workshops (ICSTW)*.

Taylor K. and Moore S. (2006): My compiler really understand me: an adaptive programming language tutor. In V. Wade, H. Ashman, B. Smith (Eds): *AH 2006*, LNCS4018, 389-392.

Zhao Y., Chen G., Liao C., Shen X. (2016): Towards ontology-based program analysis. *CVIT 2016*, 26:1-26:25.

Wagner J. and Leupers R. (2001): C Compiler design for an industrial network processor. *Proc. of the ACM SIGPLAN Workshop on Languages, compilers and tools for embedded systems*, 155-164.

Technical Reports:

de Graaf D. (2017) : Practical use of automata and formal languages in the compiler field. *BSc Thesis*, University of Amsterdam.

Computer Programs: Protégé: http://protégé.stanford.edu