Computational Thinking for Beginners: A Successful Experience using Prolog

Silvio Beux, Daniela Briola, Andrea Corradi, Giorgio Delzanno, Angelo Ferrando, Federico Frassetto, Giovanna Guerrini, Viviana Mascardi, Marco Oreggia, Francesca Pozzi*, Alessandro Solimando, and Armando Tacchella

DIBRIS, University of Genoa, Italy – CNR, Italy

Abstract. We discuss a logic-based methodology that we adopted to teach basic concepts of programming to high school students with a scientific profile and very basic knowledge of computer science. For our experiments we combined lectures on inductive reasoning with Prolog, practice on natural language processing and ontologies, and evaluations based on questionnaires before and after the workshop.

1 Introduction

The idea that thinking might be understood as a form of computation, as recently suggested by one of the main experts in knowledge representation and reasoning in artificial intelligence [11], is extremely fascinating. In his book, H. Levesque shows how to support students to make the connection between thinking and computing by learning to write computer programs in Prolog for a variety of tasks that require thought, including understanding natural language.

Taking inspiration from recent experiments during a workshop for high school students organized by the "Guidance, Promotion and Tutoring Committee" of the Computer Science Degrees at the University of Genova[1], we present a proposal for a condensed course for uninitiated aimed at introducing basic and advanced programming concepts using declarative languages like Prolog, following Levesque's "thinking as computation" metaphor. Our work is inspired to seminal proposals by Kowalski [7,8,9] and to more recent works such as the "Prolog programming: a do-it-yourself course for beginners" by Kristina Striegnitz[2].

Although Prolog as a programming language for novices has been heavily criticized in the past because of the misconceptions it may generate [13], many resources for teaching Prolog to beginners can be found on the web. Among them, we can mention implementations like Visual Prolog for Tyros[3], Strawberry

[*] Dr. Francesca Pozzi is affiliated with CNR; all the other authors are affiliated with DIBRIS.

[1] `http://informatica.dibris.unige.it/`.

[2] `http://cs.union.edu/~striegnk/courses/esslli04prolog/`.

[3] `http://www.visual-prolog.com/download/73/books/tyros/tyros73.pdf`.

Prolog[4], Pretty Prolog[5], the book Learn Prolog Now! [3], also available online[6], as well as many tutorials. This abundance of teaching material witnesses that the Prolog community is extremely lively, and convinced us that it was worth teaching Prolog to students with no programming skills. This decision was also motivated by previous attempts with Scratch and its spin-offs Byob and SNAP![7], that – albeit suitable for allowing beginners to write an almost complex program in a few hours – were perceived as not enough professional.

The course requires about 12 hours and it is thought as a crash course for high school students with different profiles. Its template can be instantiated in many different ways, provided that no previous programming skills are assumed. Our experience with the course was carried out during a workshop involving 39 high school students over three days. The course has a leitmotif — which, in our case, was ontology-driven sentiment analysis — and it is organized in modules as follows.

- **Module 1**: a preliminary lecture on a general topic in computer science and artificial intelligence that will provide the main running example for the class and laboratory sessions (1 hour).
- **Module 2**: an introduction to basic concepts of natural language, logic, knowledge representation and programming with the pure fragment of Prolog (2 hours) followed by an introduction to inductive definitions and basic data structures such as lists (2 hours).
- **Module 3**: an introduction to existing tools supporting exercises related to the main application discussed in the first module (1-2 hour).
- **Module 4**: a practical session with Prolog aimed at developing an application related to the main topic of the course (5-6 hours).

In addition to a description of the teaching activities above, in this paper we address also the problem of evaluating the results of the workshop. In particular, we try to assess whether questionnaires compiled at the beginning and at the end of the workshop can help us in evaluating the impact of the event on the students.

2 Introduction to the Course Leitmotif

The first module of the course introduces the course leitmotif, i.e., ontology-driven sentiment analysis.

Sentiment analysis, also known as opinion mining, is a linguistic analysis technique where a body of text is examined to characterize the tonality of the document[8]. It was first introduced by Pang, Lee and Vaithyanathan in 2002

[4] `http://www.dobrev.com/`.

[5] `https://code.google.com/p/prettyprolog/`.

[6] `http://learnprolognow.org/lpnpage.php?pageid=top`.

[7] `https://snap.berkeley.edu/`.

[8] Definition from The Financial Times Lexicon, `http://lexicon.ft.com/Term?term=sentiment-analysis`.

[17]; a survey by two of these three authors dating back to 2008 [14] defines opinion mining and sentiment analysis as areas dealing with the computational treatment of opinion, sentiment, and subjectivity in text. Given the growth of user-generated contents, sentiment analysis is useful in social media monitoring to automatically characterize the overall feeling or mood of groups of people, e.g., how consumers feel with respect to a specific brand or company. To make an example, Thomson Reuters uses sentiment analysis in a number of its different products to provide traders with information about how companies are faring in news articles.

In the context of computer science, an ontology is a specification of a conceptualization. That is, an ontology is a description (like a formal specification of a program) of the concepts and relationships that can exist for an agent or a community of agents [5]. One of the first papers describing the idea of using an ontology to drive the classification of texts by providing lexical variations and synonyms of terms that could be met in the documents was [15]. The domain of interest was that of online product reviews. Among other examples of ontology-driven sentiment analysis we may mention [4,1,6,10].

Our initial lecture was entitled "Ontologies and text classification: two relevant elements to program thinking machines". It first introduced the notion of ontologies and their application, then mentioned the semantic web [2], and finally introduced the general problem of classifying texts and its application to sentiment analysis. As the lecture was only 1 hour long, the topics were just touched upon. A more technical presentation of the steps to be faced in order to detect the polarity of a textual document was given as part of the practical session with Prolog discussed in Section 5.

3 Introduction to Inductive Reasoning with Prolog

The goal of the second module of the course is to introduce the main concepts of the computational paradigm underlying Prolog. To simplify the task, it is convenient to consider the pure fragment of Prolog without use of complex predicates tied to the evaluation strategies of the interpreter, e.g., "cut". The lectures are divided in four parts, discussed in detail in the rest of the Section.

The first part introduces the key ingredients of the Prolog language, i.e., the distinction between a program, a query, and the interpreter. The program is viewed here as the representation of some knowledge, and thus we only considered Datalog-like programs containing predicates and constants. Free variables were introduced together with queries. We started with simple examples of assertions in natural language like: *Mia is a person*, *Jody is a person*, *Yolanda is a person*, *Jody plays guitar*. The above mentioned assertions were represented then via the facts:

```
person(mia).  person(jody).  person(yolanda).  plays(jody,guitar).
```

The example was mainly used to (*i*) explain the difference between a predicate and a constant, and (*ii*) show how unary predicates can be used to describe concepts, constants to describe instances, and binary predicates to describe relations between instances of concepts. Before introducing variables and clauses, we gave an intuition on how the Prolog environment works by using simple queries like `?- person(mia).` We showed examples of successful queries, failures and errors to emphasize the differences between them. Noticeably, students could immediately see the parallel between the Prolog framework and classical examples of human-computer interaction thanks to the yes/no dialog of the interpreter. We explained here that the intelligence of the machine is programmed by writing an appropriate knowledge base, i.e., the programmer's task is to give intelligence to the machine.

In the second part we introduced increasingly complex queries to pave the way towards clauses and inductive definitions. We started from simple queries such as `?- person(X).` showing also how to get multiple answers using the ';' command. We then moved to other examples based on binary predicates such as `?- plays(jody,X), ?- plays(X,Y).` These examples were used to describe the interplay between constants and variables, and to introduce (informally) the notion of matching and unification. The next important concept to introduce was that of conjunctive queries. For this purpose, we used examples such as `?- person(jody) , plays(jody,guitar)`, `?- person(X) , plays(X,guitar)`. We explained the logical interpretation of comma as conjunction, and interpreted the shared variable X as a communication channel between two subqueries: the first one instantiates it, the second one validates the instantiation. We introduced then the notion of clause as a way to define implicit knowledge, i.e., to derive new facts without need of enumerating all of them. The first examples were used to illustrate the syntax of a clause:

```
happy(jim).
hasMp3(jim).
listens2Music(jim) :- happy(jim), hasMp3(jim).
```

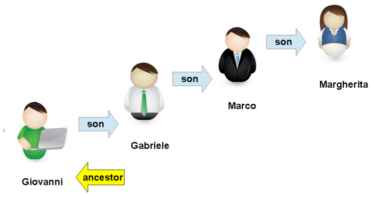
We explained the semantics using the following assertions: *Jim is happy. Jim has an MP3 player. If Jim is happy and has got an MP3 player, he listens to music.* The use of ground rules, which allows only one specific inference, is quickly abandoned by introducing free variables to show how to infer several new facts with a single clause:

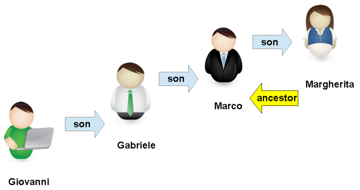
```
guitarist(X) :- plays(X,guitar).
```

The third part was dedicated to inductive definitions. We started from inductive definitions not involving data structures. The classical example was the program defining the transitive closure of a given relation. We used here the relation `son` defined as follows.

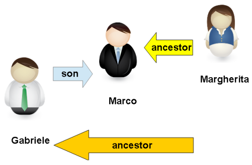
```
son(giovanni,gabriele).  son(gabriele,marco).  son(marco,margherita).
```

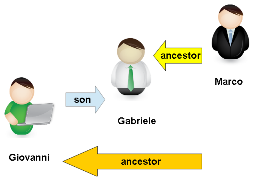
We first juxtaposed the ancestor relation over the different definitions of the son relation. We then showed how to combine the son and ancestor relations in order to deduce all other cases. In the explanation we avoided inductive definitions on ancestor predicates only. The following pictures give a diagrammatic presentation of the inductive definition (introduced step by step in the lecture). We started from the redefining the facts of the `son` relation by using `ancestor`.



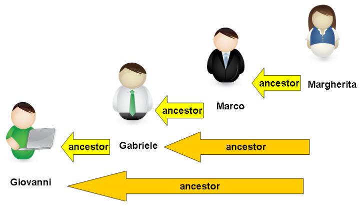


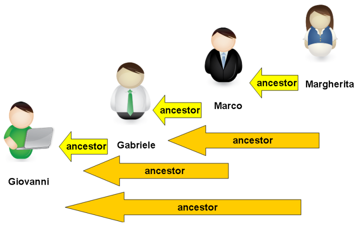
The ancestor relation was introduced step by step using the transitive closure:





Finally we completed the diagram of the new relation as follows:





The use of inductive steps defined by combining `son` and `ancestor` (instead of applying the transitive closure on `ancestor` only) worked well and did not cause ambiguity in the explanations. In this way, we immediately focused the attention on the standard way to avoid non terminating recursive definitions in Prolog. The intuition behind the base step of the inductive definition was first expressed using assertions in natural language like: *For every X, Y, if X is a son of Y, then Y is an ancestor of X.* The assertion was then formalized as the Prolog clause

```
ancestor(Y,X) :- son(X,Y).
```

Again we stressed the fact that clauses with free variables can be used to define new facts on top of existing ones without need of enumerating all of them, i.e., clauses define implicit knowledge. Similarly, the inductive step was first expressed using assertions in natural language like: *For every X, Y, if X is a son of Y and Z is an ancestor of Y, then Z is an ancestor of X.* The assertion was then formalized via the Prolog clause

```
ancestor(Z,X) :- son(X,Y) , ancestor(Z,Y).
```

We concluded the example by showing the possible result of a query.

The fourth and last part of the lectures was dedicated to simple data structures like lists. We first introduced the syntax and the intuition behind the data structures. To explain how to manipulate lists in Prolog, we started by defining a predicate to check if a list contains names of persons only. To explain how the definition works, we consider a procedural interpretation of a recursive definition (consume elements until the list becomes empty). We used the parallel with the transitive closure example to split the definition in base and inductive steps, respectively. However we listed the inductive step before the base step to emphasize the idea that to process a list we first have to consume their elements, and then define what happens when the list becomes empty. We then moved to more complicated examples like `member`. To define the member predicate we first used the assertion in natural language for the base step: *The name X is in the list head. If the name X is in the list L, then X occurs in the list extended with Y different from X.* We presented then the clauses:

```
member(X,[X|L]).
member(X,[Y|L]):-X=/=Y,member(X,L).
```

We used a similar approach to introduce other operations like notin (X is not in a list L), add (add X to a list), and add* (add X if it is not already in L). We also introduced examples of nested lists. Finally, we used (nested) lists to represent and manipulate syntax trees of simple natural language sentences. Specifically, we considered the syntax tree of a text formed by a list of sentences. A sentence was represented as a list of words as specified by the grammar of the considered language, e.g., noun-verb-object We then defined examples of tokens like

```
noun(mia).  noun (jody).  verb(plays).  object(guitar).  object(drums).
```

The assertion *if S is a noun, V is a verb, O is an object, then S V O is a sentence* was modeled via the clause

```
sentence([S,V,O]):-noun(S),verb(V),object(O).
```

This allowed us to give an inductive definition of a text as follows.

```
text([]).
text([F|T]) :- sentence(F) , text(T).
```

Finally, we put all together and showed examples of derivations of queries like `?- text ([ [mia,plays,drums], [jody,plays,guitar] ])`. Again, we used the metaphor of traversal with consumption element-by-element to manipulate nested lists as in the case of simple lists.

4 Practical Session with an Ontology Editor

The third module is meant to introduce the tools useful for the specific course domain — in our case, ontology-driven sentiment analysis. Depending on the domain and related tools, this module may require different amount of time. In our instantiation of the course, we opted for a domain that allowed us to introduce intuitive and user-friendly tools, that high-school students could use with as little training as possible. In particular, the tool session was organized as a practical laboratory session aimed at introducing the Protégé Ontology Editor. The main reasons for our choice is that the course teachers already had a background on this tool, and that the university students helping during the session had seen Protégé during the "Intelligent Systems and Machine Learning" Master's course. A second reason is that the Web Protégé version[9] can be used online without requiring any installation.

Because of hard time constraints we could not make an introductory lecture on Protégé, so the students just learned by doing during the practical sessions. Some students observed that it would have been useful to introduce the tool beforehand, so we plan to find at least half an hour for explaining the tool and the proposed exercises in the next course editions. The exercises we proposed are based on the Newspaper Example available from the Protégé Frames User's Guide[10]. The Newspaper ontology associated with that guide was made available to the students in their temporary home for easier use.

Exercise 1. The first exercise aimed at making the students acquainted with Protégé by exploring the already made `newspaper` ontology. The text was the following:

1. Open the Protégé ontology editor and select the Newspaper ontology. The ontology domain is that of managing the costs and organization of a newspaper. The ontology can answer the following questions:
   - Who is responsible for each section of the newspaper?
   - What is the content of each article in a section, and who is its author?
   - Who does each author report to?
   - What is the layout and cost of each section?
2. Explore the ontology and experiment with addition and removal of new classes, instances, properties.

Exercise 2. The second exercise was related to the course leitmotiv and asked to design and implement a simple ontology for the opinion mining in the hotel reviews domain, using names in English. The students could save their ontology and were informed that they could have used it in the next module. Students were suggested to identify the positive terms that they could expect in a positive review (for example charming, excellent, polite, clean, ...), the negative ones (dirty, bad, unsafe, ...) and the neutral ones. Then they were suggested to organize them in an ontology having three main branches, one for positive, one for negative and one for neutral words in this domain. To take inspiration for the words, we suggested to read some real reviews available for example on TripAdvisor.

[9] `http://webprotege.stanford.edu`.

[10] `http://protegewiki.stanford.edu/wiki/PrF_UG`.

Exercise 3. In case some students still had time, we proposed to create and run some queries on the `newspaper` ontology used in the first exercise, such as

- Find the journal that contains the article “Destination Mars” and save the query;
- Find the journals that either contain the article “Destination Mars” or have less than 100 pages and save the query.

5 Practical Session with Prolog

The last module of the course integrates all the competencies gained in the previous modules and proposes exercises with increasing complexity, aimed at developing a simple but working application for ontology-driven sentiment analysis. It uses pieces of Prolog code developed by the teachers, offering predicates that make use of external libraries.

A short lecture introduces the goal of the pratical Prolog session by means of an example. Given an ontology like the one depicted in Figure 1 and a review like

> *This hotel is beautiful! It is in a great location, easy to walk anywhere around the city. Very nice, comfortable room with lovely views. Staff can speak English. Fantastic breakfast with many different types of foods available. I would stay here again in a heartbeat.*

we asked the students how could we manage to obtain a classification like

```
6,[review]       6,[positive,review]       1,[nice]
1,[lovely]       1,[great]                 1,[comfortable]
1,[beautiful]    1,[available]
```

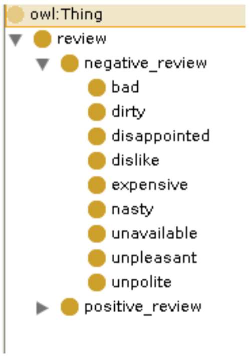


Fig. 1. A basic ontology for sentiment analysis in the hotel review domain (only the negative branch is shown for space constraints).

During this short lecture we emphasized that, in order to reach our goal, we had to fix the language of both the text and the ontology (we agreed on English) and we needed

- a tokenizer for transforming a text into the list of its elements (words, punctuation), in order to operate on lists and not directly on text;
- a list of English stopwords to be filtered out before processing the text, as they do not contribute to the text's semantics;
- a stemmer for removing most common morphological and inflectional endings from English words in order to normalize the terms in both the ontology and the text, to make their matching possible;
- a tool for reading an OWL ontology from file and transforming it into some format easy to manipulate.

Exercises 1, 2. The first practical exercise did not depend on the domain. In particular, it was taken from the SWISH web site `http://swish.swi-prolog.org/example/movies.pl` and it is based on querying and extending a movie database. The second exercise asked the students to implement the predicate for removing a ground item from a ground list.

Exercise 3. The third exercise, whilst still being a classical one for Prolog beginners, started to move towards the actual problem to be solved. We asked to implement a `subtractList` predicate for subtracting a list from another one, but we contextualized the problem supposing to have a list of words that represents all the words that are found in a review, and a list of stopwords and punctuation elements. The goal is to remove the stopwords from the list of words retrieved from a text.

We made available to the students the following material:

- The file `stopwords.txt` containing 430 English stopwords and punctuation marks.
- The `emotions.owl` ontology sketched in Figure 1. Since the students had already completed the laboratory with Protégé, we told them that they could use their own ontology instead of the provided one.
- A `textclassifier.pl` Prolog piece of code offering all the solutions to the exercises, but implemented as predicates with different names. We asked the students to refrain from reading this file thoroughly — as they would have found the solved exercises —, but just to consult it to find implementation of auxiliary predicates. The text classifier used the following SWI Prolog libraries:
  - The RDF database (`library(semweb/rdf_db)`)[11] for reading an ontology and transforming it into a set of Prolog facts.
  - The Porter Stem (`library(porter_stem)`)[12] implementing the Porter stemmer [16].

[11] `http://www.swi-prolog.org/pldoc/man?section=semweb-rdf-db`.

[12] `http://www.swi-prolog.org/pldoc/man?section=porter-stem`.

– A set of reviews of hotels in Genova, downloaded from the Booking.com site[13].

We provided the following suggestions to verify that the code was correct:

1. Consult the `textclassifier.pl` file which contains the Prolog code to implement a basic ontology-driven text classifier. The code offers many useful predicates to make your work easier.
2. Use the predicate `fromTextToList(FileName, List)` that takes as its first argument the name of a file and unifies the second argument with the list of the words found in the file: this predicate will allow you to read the contents of a file, be it a review or the `stopwords.txt` file, and turn it into a list that Prolog can manage.
3. After having called the predicate you implemented for subtracting a list from another one, use the predicate `printList(List)` to print the result.

We also provided an example of goals to call and the expected output:

```
?- [textclassifier].
?- fromTextToList('./review5.txt', Review5List),
   fromTextToList('./stopwords.txt', StopWordsList),
   subtractList(ListReview5, StopWordsList, Review5WithoutSWList),
   printList(Review5WithoutSWList).

extremely comfortable welcoming excellent service conveniently situated
railway station main sights palazzo reale best breakfast buffet
experienced week visit italy adjacent restaurant tralalero good
```

The file `review5.txt` contained

> *Extremely comfortable, welcoming, with excellent service and conveniently situated for the railway station and most of the main sights, such as Palazzo Reale. By far the best breakfast buffet I experienced during a two-week visit to Italy! Its adjacent restaurant, Tralalero, is also very good.*

Exercise 4. The fourth exercise asked to implement the `listStem(LWords, LStem)` predicate for obtaining the list of word stems, from the list of original words. The students could use the auxiliary predicate `extendedPorterStem(X, Y)` offered by `textclassifier.pl` to obtain the stemmed word of `X` and unify it with `Y`.

Exercise 5. With the fifth exercise, we started to practice with an ad-hoc Prolog representation of ontologies. `textclassifier.pl` implements a `loadOntology` predicate which, given the OWL ontology file name, asserts information on the ontology classes and subclass relationships. The asserted fact also provides information on the concept name and its stem. For example, by calling `loadOntology('./emotions.owl')`, the following facts are asserted into the Prolog Knowledge Base:

[13] `http://www.booking.com/reviews/it/hotel/bristol-palace.en-gb.html`.

```
class(http://www.owl-ontologies.com/o.owl#amazing,[amazing],[amaz])
class(http://www.owl-ontologies.com/o.owl#available,[available],[avail])
class(http://www.owl-ontologies.com/o.owl#bad,[bad],[bad])
class(http://www.owl-ontologies.com/o.owl#beautiful,[beautiful],[beauti])
class(http://www.owl-ontologies.com/o.owl#best,[best],[best])
class(http://www.owl-ontologies.com/o.owl#charming,[charming],[charm])
class(http://www.owl-ontologies.com/o.owl#positive_review,
                                            [positive,review], [posit,review])
........
subClass(http://www.owl-ontologies.com/o.owl#amazing,
             http://www.owl-ontologies.com/o.owl#positive_review)
subClass(http://www.owl-ontologies.com/o.owl#available,
             http://www.owl-ontologies.com/o.owl#positive_review)
subClass(http://www.owl-ontologies.com/o.owl#bad,
             http://www.owl-ontologies.com/o.owl#negative_review)
subClass(http://www.owl-ontologies.com/o.owl#beautiful,
             http://www.owl-ontologies.com/o.owl#positive_review)
```

The exercise asked the students to load `emotions.owl` and query the knowledge base in order to answer the following question:

1. Which are the words and stems associated with the ontology class `http://www.owl-ontologies.com/o.owl#amazing`?
2. Which ontology concept has the word list `[tasty]` associated with?
3. Which ontology concept has the stem list `[unavail]` associated with?
4. Which ontology concept has the stem list that contains `posit`?
5. Which is the direct superclass of the class whose word list is `[lovely]`?
6. Which are *all* the superclasses of the class whose stem list is `[excel]`?

We also asked the students to make experiments with the ontology they implemented in the Tools Session.

Exercise 6. The last exercise was the most challenging one, as it asked to put all the bricks implemented so far together, in order to implement an ontology-driven text classifier. We gave very limited written hints and we proposed a few variants of the text classification program for listing not only the words occurring in both the text and the ontology (after being stemmed), but also in counting their occurrences: if "beautiful" occurs twice, it should be counted twice and contribute to a more positive evaluation. Another variation we proposed was to use the subClassOf semantic relations present in the ontology to classify the text not only based on the words that coincide with classes in the ontology, but also with their superclasses.

Exercise 7. Since one implementation of the text classifier was available, we concluded this Prolog practical session by asking the students that could not complete Exercise 6, to experiment with our own implementation in order to see a program at work.

Discussion. Although we did not carefully trace the results achieved by the students during their practical experience, we can say that about 10-12 students out of 39 were able to complete the 6th exercise. This result was extremely surprising for all the instructors. In fact, having taught Prolog for many years to students with a solid background on imperative and object-oriented programming, we were aware of the difficulties that students meet when moving from the logic programming theory to the practice and we did not expect that some students could face and complete Exercise 6.

Without claiming to make a scientifically founded assertion, our feeling was that the lack of skills in imperative programming of the high school students, made them open to "think directly in Prolog", without trying to design imperative algorithms and then fit them into logical rules. We plan to take more precise statistics on the completion of the practical exercises in the next editions of the workshop, and maybe propose this course also to students with a computer science background, in order to confirm these feelings.

6 Evaluation

Our evaluation is targeted to understand — in a quantitative way — whether the students were able to improve their computational thinking capabilities after the workshop.

To this purpose, data were collected the first and the last day of the workshop via a questionnaire, with some overlap w.r.t. the general one. The aim was to collect information about the profile of the students and to check their ability to solve easy logic problems. The questionnaire was proposed before and after the experience in order to test whether the experience increased student's knowledge of the subjects.

A First part (profile part) of the specific questionnaire asked students to provide information about their gender, grade, their favorite subject at school (Humanities, Science, Technical subject, Language or Arts), their daily use of computers (from 1 = less than half an hour, to 4 = more than 2 hours), their perceived level of academic achievement (from 1 = excellent to 4 = barely passing), whether they were planning to enroll in a course at the University (yes, maybe, no) and whether they were planning to enroll in a program in Computer Science (yes, maybe, no). In the Second part (questions part) students were asked to answer questions based on logic skills; for this part no specific knowledge is provided or expected. They were recommended to answer only the questions they were able to solve, without trying to answer randomly. Accordingly, the score for every answer was: +1 if right, 0 for blank, and -0.5 if wrong. With this choice of weights, since every question has three alternatives and only one is correct, choosing randomly between the answers yields an expected score of

$$\frac{1}{3} \cdot 1 - \frac{1}{3} \cdot 0.5 - \frac{1}{3} \cdot 0.5 = 0$$

We first tested whether the experience increased student's knowledge of the subjects. The statistical test is performed by using a paired t-test to check the difference in the results of the Second part before (Time 1) and after (Time 2) the experience. Our results indicate that at Time 2 students are slightly more likely to correctly solve questions, but not sufficiently to have a statistical significance; this emerges even by splitting the population on a subject-base or on a perceived level-base. As a possible explanation, we could consider the relatively small amount of questions provided in the questionnaire, and also to the very condensed format of the training.

C.S.program	Post Cert.	Post Uncert.
Pre Cert.	23	1
Pre Uncert.	7	4

Fig. 2. Interest of enrollment at the Computer Science program: the rows and the columns depict respectively the occurrence or the answers during the test before and after the experience.

Since the experience had the main goal of adequately introducing a Computer Science curriculum, it was also interesting to test whether the experience modified students' intentions to enroll either in the University or in the Computer Science program. The difference between certainty and uncertainty in the intention was tested at Time 1 and Time 2 by performing an exact McNemar test [12]. Regarding the interest to enroll at the University, there was no significant result: only 2 students passed from uncertainty to certainty ($p = 0.47$). The same statistical test was performed on the interest to enroll at the Computer Science program and the result showed that a small proportion of students switched from uncertainty to certainty in their intention to enroll (see Fig. 2) with a $p - value$ of 0.07. Even if this result is higher than our statistical significance threshold (0.05), the fact that the p-value is close to it suggests that the experience might have helped them make up their mind. Despite the limitations of this study (i.e., relatively low sample size, small number of questions, lack of observations by external observers) we can conjecture that with an improved and enlarged set of questions, significant results can be achieved to better guide teachers in their choice of course contents.

7 Conclusions

In this paper we have discussed a format for a tutorial about the basic concepts underlying the computational thinking paradigm using a declarative language like Prolog. The tutorial is centered around a specific application domain. In this paper we discussed the domain of Natural Language Processing, Semantic Web and Ontologies, three hot topics in Computer Science and Artificial Intelligence with several important real-life applications. The domain was adopted as main example in a workshop for high school students taught at the University of Genoa.

The tutorial is based on a crash-course that introduces the application domain, followed by lectures on declarative programming, on the use of tools, and practical sessions to learn the basics of programming, and a final project in which the students apply their new programming skills to a concrete problem. Declarative languages are used here to introduce complex concepts like recursive and inductive definitions with the help of natural language assertions. Ontologies and natural language processing are particularly useful to give an application-oriented flavor to the workshop.

References

1. Matteo Baldoni, Cristina Baroglio, Viviana Patti, and Paolo Rena. From tags to emotions: Ontology-driven sentiment analysis in the social semantic web. *Intelligenza Artificiale*, 6(1):41–54, 2012.
2. Tim Berners-Lee, James Hendler, and Ora Lassila. The Semantic Web. *Scientific American*, pages 29–37, May 2001.
3. Patrick Blackburn, Johan Bos, and Kristina Striegnitz. *Learn Prolog Now!*, volume 7 of *Texts in Computing*. College Publications, 2006.
4. Marcirio Chaves and Cássia Trojahn. Towards a multilingual ontology for ontology-driven content mining in social web sites. In *Proc. of ISWC 2010, Volume I*, 2010.
5. Thomas R. Gruber. A translation approach to portable ontology specifications. *Knowl. Acquis.*, 5(2):199–220, June 1993.
6. Efstratios Kontopoulos, Christos Berberidis, Theologos Dergiades, and Nick Bassiliades. Ontology-based sentiment analysis of twitter posts. *Expert Syst. Appl.*, 40(10):4065–4074, 2013.
7. Robert A. Kowalski. Logic as a computer language for children. In *ECAI*, pages 2–10, 1982.
8. Robert A. Kowalski. Logic as a computer language for children. In *New Horizons in Educational Computing, (ed. M. Yazdani), Ellis Horwood Ltd., Chichester*, pages 121–144, 1984.
9. Robert A. Kowalski. A proposal for an undergraduate degree in the uses of logic. In *Artificial Intelligence in Higher Education, CEPES-UNESCO International Symposium, Prague, CSFR, October 23-25, 1989, Proceedings*, pages 94–97, 1989.
10. Maurizio Leotta, Silvio Beux, Viviana Mascardi, and Daniela Briola. My MOoD, a multimedia and multilingual ontology driven MAS: Design and first experiments in the sentiment analysis domain. In Cristina Bosco, Erik Cambria, Rossana Damiano, Viviana Patti, and Paolo Rosso, editors, *Proceedings of the 2nd Workshop on Emotion and Sentiment in Social and Expressive Media (ESSEM) 2015, a satellite workshop of AAMAS 2015*, 2015.
11. Hector J. Levesque. *Thinking as Computation.* The MIT Press, 2012.
12. Quinn McNemar. Note on the sampling error of the difference between correlated proportions or percentages. *Psychometrika*, 12(2):153–157, June 1947.
13. Patrick Mendelsohn, T.R.G. Green, and Paul Brna. Programming languages in education: The search for an easy start. In J.-M. Hoc, T. R. G. Green, R. Samurçay, and D. J. Gilmore, editors, *Psychology of Programming*, pages 175–200. London, Academic Press, 1990.
14. Bo Pang and Lillian Lee. Opinion mining and sentiment analysis. *Found. Trends Inf. Retr.*, 2(1-2):1–135, January 2008.

15. Jantima Polpinij and Aditya K. Ghose. An ontology-based sentiment classification methodology for online consumer reviews. In *Proc. of IEEE/WIC/ACM WI-IAT'08*, pages 518–524, 2008.
16. Martin F. Porter. An algorithm for suffix stripping. *Program*, 14(3):130–137, 1980.
17. Peter D. Turney. Thumbs up or thumbs down? Semantic orientation applied to unsupervised classification of reviews. In *Proc. of ACL 2002*, pages 417–424, 2002.