TOLC-ASP: a tool for training students for admission tests [⋆]

Edda Dal Santo, Agostino Dovier[*] and Talissa Dreossi

Università degli Studi di Udine, DMIF, Via delle Scienze 206, 33100 Udine

Abstract

We present a tool generating and executing logical tests of the same kind of those administered by the Italian Universities for a sort of admission tests. They are developed by CISIA, usually referred to as TOLC. A set of benchmarks drawn from official repositories has been encoded into ASP and the solutions have been verified. Other tests have been invented starting from them and checked in ASP. Tests have been then parametrized with names of people involved, actions, environments, and so on. This way an exponential number of different tests, w.r.t. the initial collection, can be obtained, basically with random grounding, and their solutions verified by ASP. Furthermore, a set of explanations of why an answer is wrong has been added for didactical purposes. The tool has been tested with students of the first year of the Degree in Computer Science.

Keywords

Didactic of Computer Science, Logic Education, ASP

1. Introduction

The TOLC (CISIA Online Test) is a test developed by the Italian Inter-university Consortium of Integrated Access Systems (CISIA) used by most Italian universities to verify the minimum knowledge required for access to degree courses and/or to guide the choice of the most promising university career. Different types of TOLC are associated with specific study programs. We have focused on the TOLC-S, the entrance test for Scientific degree courses. It is composed of 55 questions divided into six categories: 20 on Mathematics, 15 on “Reasoning, Problems and Text Comprehension”, then 5 each on Biology, Chemistry, Physics and Earth Sciences. The first and second groups of questions are those in which students generally encounter the greatest difficulty, as witnessed by the report [1].

		2021		2022		2023	
	N. questions	Mean	Standard Deviation	Mean	Standard Deviation	Mean	Standard Deviation
Total	50	22,0	11,5	21,8	11,5	21,1	11,5
Mathematics	20	10,0	5,9	10,0	5,8	9,5	5,8
Reasoning and Problems	10	3,5	2,5	3,3	2,5	3,3	2,5
Text Comprehension	10	4,8	2,7	4,8	2,7	4,8	2,7
Sciences	10	3,6	2,6	3,6	2,7	3,5	2,7

Table 1
Score statistics of the TOLC-S by year and by section, as depicted in [1, Table 10.7]; before 2024 the test had a slightly different structure, including 20 questions on Mathematics, 10 on Reasoning and Problems, 10 in Text Comprehension and 10 on Sciences. The table shows that, among all sections, the Reasoning and Problems one records the lowest scores.

CILC 2025: 40th Italian Conference on Computational Logic, June 25–27, 2025, Alghero, Italy

[⋆] Research partially supported by Interdepartment Project on AI (Strategic Plan of UniUD–22-25), and by Unione europea-Next Generation EU, missione 4 componente 2, project MaPSART “Future Artificial Intelligence (FAIR)”

[*] Corresponding author.

dalsantoedda@gmail.com (E. Dal Santo); agostino.dovier@uniud.it (A. Dovier); talissa.dreossi@uniud.it (T. Dreossi)
https://dmif.uniud.it/dovier (A. Dovier)
0009-0008-2628-1740 (E. Dal Santo); 0000-0003-2052-8593 (A. Dovier); 0009-0007-1746-6500 (T. Dreossi)

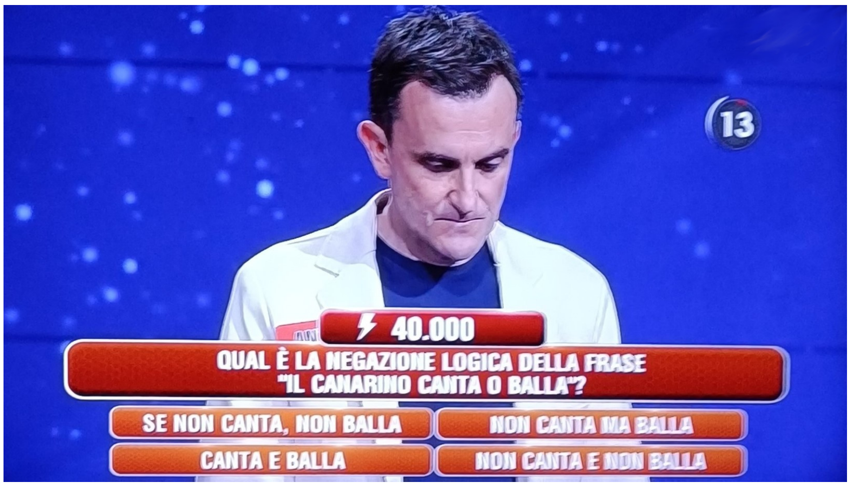


Figure 1: From L'Eredità, May 2nd, 2025. Question: what is the logical negation of the sentence: "the canary sings or dances"? Options: (1) if it does not sings, it does not dance, (2) it dances and it does not sing, (3) it sings and dances, (4) it does not sings and it does not dance.

In particular, the sub-part of logical tests are those that are considered hardest among all. It is not hard to convince researchers in logic programming on the importance of having strong roots in logics. Logical skills are essential for success in scientific studies, for understanding the rationale behind mathematical proofs, and in general for reasoning, problem-solving, programming. Typical errors are inverting formulas, confusing if, only-if, if and only if, reasoning with quantifiers, and so on. These kind of questions are also inserted in the other TOLC tests. CISIA offers on its website [2] teaching material for preparing for the tests, including a text called: *Mentor di Logica* [3] (briefly, Mentor in the remaining part of the paper).

This document represents one of the starting points of this work, namely developing a targeted tool that allows students to practice on this type of questions. In particular, the paper aims to lay the foundations for the future creation of a web or mobile application that represents a digital version of the Logic Mentor.

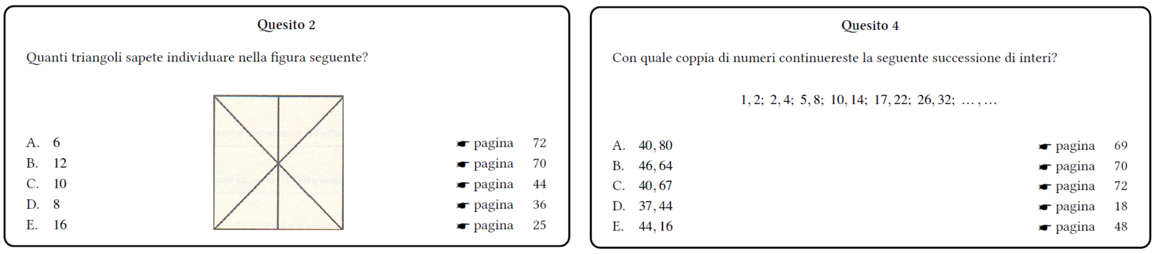
A further example of the difficulty felt in front of logical (even simple) questions comes from the popular italian television game "L'eredità" transmitted on RAI1 exactly on the deadline date of the CILC 2025 conference. The competitor, an engineer and champion of the program, received the question reported in Figure 1. Observe that the value of the answer is 40k€. We don't unveil his answer but curious readers might enjoy it on Raiplay[1].

This kind of puzzles are (currently) also a limit for Large Language Models (LLMs). The lack of a true logical-deductive reasoning module and a "reasoning" relying on statistical correlations learned during training makes these tools still trying to justify their first choice without doing some (hard and time costly) backtracking job.

Answer Set Programming (ASP) [4] instead has among his main features the capability of dealing with logical puzzles if correctly encoded [5, 6, 7]. Results are obtained and justified using formal logic and rigorous deduction under the so called stable model semantics. And they can be explained to experts and non-experts using tools developed for this scope [8]. Experience in High schools for using ASP for encoding and solving puzzles even in absence of other introductory courses have been made and proved successful [9].

The paper is organized as follows: in Section 2 we describe in more details the main elements of the quizzes of the TOLC. In Section 3 we report the basic ideas for the ASP encoding of families of logical puzzles. In Section 4 we present our tool for generating TOLC-like quizzes and how it can help the tester when he/she provides wrong answers. In Section 5 we report some experimental results on the use of our tool. Some side comments are reported in Section 6 and some conclusions are drawn in

[1]https://www.raiplay.it/programmi/leredita — 2 Maggio 2025, minute 51



Quesito 2

Quanti triangoli sapete individuare nella figura seguente?

A. 6 ☛ pagina 72
B. 12 ☛ pagina 70
C. 10 ☛ pagina 44
D. 8 ☛ pagina 36
E. 16 ☛ pagina 25

Quesito 4

Con quale coppia di numeri continuereste la seguente successione di interi?

$$1,2;\ 2,4;\ 5,8;\ 10,14;\ 17,22;\ 26,32;\ \ldots,\ldots$$

A. 40, 80 ☛ pagina 69
B. 46, 64 ☛ pagina 70
C. 40, 67 ☛ pagina 72
D. 37, 44 ☛ pagina 18
E. 44, 16 ☛ pagina 48

Figure 2: Two examples of puzzles from the Mentor not belonging to the propositional/f.o. logics category

Section 7.

2. TOLC Logical Puzzles

The Mentor [3] is a self-learning tool designed to help students practice multiple-choice logic questions, although it is not a strict simulation of the TOLC test. Available in PDF format on the CISIA website (accessible after registration), this guide features a unique interactive structure. The reader begins with an initial question and selects an answer, being then redirected to another page of the document: if the choice is correct, confirmation is given along with a new puzzle to solve; if incorrect, a feedback is provided and the reader is taken back to the previous question. This non-linear navigation (supported by page references and hyperlinks) encourages active reasoning and logical thinking in a low-pressure environment.

The Mentor includes questions on propositional and visual logic, sometimes requiring basic mental calculations. No advanced mathematical knowledge is needed and all texts are written in plain natural (Italian) language. On the other hand, while no specialized preparation is required, careful reading and interpretation are essential to avoid misunderstandings.

The logical puzzles included in the Mentor [3] can be classified into 4 categories:

- logical (propositional/first order) puzzles,
- puzzles that require analysis of a picture or a graphical representation to obtain the solution,
- puzzles that require to complete a series of tables, numbers, pairs of numbers, ...,
- combinatorial puzzles.

For the purpose of the application developed here, we will focus on the first family (32 examples out of 51). We can split them into (1) puzzles with/without the use of quantifiers, or, (2) puzzles in which the student must choose exactly one answer or must exclude exactly one answer. We call the latter puzzle with "dual" answer.

For completeness of presentation, in Figure 2 we report two examples of puzzles of different families.

In Section 3 we show how to encode these puzzles in ASP, demonstrating how to verify the existence of a unique solution while highlighting their underlying logical structure. The encoding section can be viewed as naïve by CILC participants, however, we decided to write it as a tutorial for readers that are non-experts.

3. ASP Encoding of Puzzles

Given a denumerable set of variables $\mathcal{X}$, a set of constant symbols $\mathcal{C}$, and a set of predicate symbols $\mathcal{P}$, an atomic formula (atom) is a formula of the form $p(t_1, \ldots, t_n)$, where $p \in \mathcal{P}$, and $t_1, \ldots, t_n$ are constant symbols or variables. A literal is either an atomic formula or a formula of the form `not` A (naf-literal), where A is an atomic formula. A (general) ASP *rule* is a formula of the form

$$H \leftarrow B_1,\ \ldots,\ B_m,\ \texttt{not}\ C_1,\ \ldots,\ \texttt{not}\ C_n \tag{1}$$

where H, B_i, C_j are atomic formulas. If $m = n = 0$, then it is said to be a *fact*. A *program* is a set of clauses. A literal, a rule, a program is *ground* if it contains no variables. The semantics of a program is based on the notion of *stable model/answer set* [10]. Given a ground program P, a set of atoms S is an answer set of P if it is the (unique) minimum Herbrand model of the so-called "reduct" P^S obtained from P by first removing all rules with $\mathtt{not}\,A$ in their body where $A \in S$ and then removing the negated literals from all remaining rules. In addition to the rules of form (1), the ASP language has been extended over time with several constructs such as denials (rules without the head H), choice rules and cardinality constraints that will be presented and used in the remaining part of this paper.

3.1. Puzzles Encoding

A first challenge has been the one of encoding logical tests from the Mentor in ASP. We report here the main ideas behind the encoding and two detailed examples.

Some domain-related properties must be stated according to the example (e.g., `person(a).` `person(b).` `brother(a,b).` etc). This is different in any puzzle and it requires a minimum of experience in logical modeling. But this is not the difficult part.

In logical tests/puzzles typically there are properties that can be either false or true and their value of truth is constrained by subsequent hints. Moreover, the set of candidate results among which the participant has to select one is often structured as a (Boolean or first-order) sentence. In the next subsections we try to generalize some lessons learned in the encoding of TOLC problems.

Non-deterministic Choice. If p is one of the properties of interest, a choice rule can be written as

```
{p}.
```

A choice rule is a syntactic sugar for the pairs of rules $p \leftarrow \mathtt{not}\,np. \quad np \leftarrow \mathtt{not}\,p.$ where np is an auxiliary predicate introduced for the negation of p.

All unknown properties of the problem can be declared by assigning them a non-deterministic choice. If, from the hints, it can be immediately inferred that p holds this can be forced simply by a fact

```
p.
```

If, instead, it can be immediately inferred that p cannot hold, this can be imposed adding a denial

```
:- p.
```

that, intuitively, impose: *It is impossible that the program admits a stable model in which the Body of the denial is true.* As said before, it is a syntactic extension, there are several well-known ways of implementing it with general rules of type (1).

Disjunctive Hints. A sentence such as *at least one of the following k literals is true* (in other words, their disjunction is true) can be expressed as:

```
h :- ell_1.
h :- ell_2.
...
h :- ell_k.
:- not h.
```

The last line (a denial) excludes the possibility of a stable model in which `not h` is true (e.g., `h` is false). Assume $\ell_1, \ldots, \ell_h$ are positive literals (atoms) while $\ell_{h+1} = \mathtt{not}\,r_1, \ldots, \ell_k = \mathtt{not}\,r_{k-h}$ are naf-literals, for h to be in a stable model it must be that at least one among $\ell_1, \ldots, \ell_h$ hold, supported by other atoms and rules, or that at least one among $r_1, \ldots, r_{k-h}$ is not in the stable model.

In the remaining part of the section we will use `h` for each example, for simplicity. In the encodings, a different predicate name must be assigned to any hint (e.g., $h_1, h_2, h_3, \ldots$).

We can generalize this encoding to sentences such as *At least m and at most n of the following k atoms/literals are true*, as follows:

```
h :- m{ ell_1; ell_2; ... ;ell_k }n.
:- not h.
```

The constructor in the body of the first rule is the so-called cardinality constraint, which is another extremely useful syntactical extension to ASP. Its semantics is exactly the one just sketched: it is true when at least m and at most n of the its k literals are true. Their truth however must be guaranteed by another rule(s).

Cardinality constraints can be rewritten, e.g., by using denials excluding that more than n literals hold and that more than $k - m$ literals do not hold.

Conjunctive Hints. If the hint states that a conjunction of (positive or negative) literals must hold, we can write

```
h :- ell_1, ell_2, ... ,ell_k.
:- not h.
```

Boolean combinations. Any other Boolean combinations can be implemented using the above ides. For instance, in the case of implications given as hint they can be encoded using their disjunctive semantics: $(A \rightarrow B)$ is equivalent to say that it is impossible that A holds and B does not hold:

```
h :- A, not B.
:- not h.
```

Quantifiers. If p is a property (an atom) depending on a parameter x, a hint such as $\exists x\, p(x)$ can be forced as

```
h :- p(X).
:- not h.
```

Similarly, a hint such as $\forall x\, p(x)$ can be expressed as:

```
nh :- not p(X).
:- nh.
```

In this case we say that it is impossible that we have `nh` (think as not h). This means that there cannot exist an x such that $p(x)$ is false, namely that for all x we have $p(x)$.

3.2. Logical Consequence with Stable Model Semantics

ASP solvers can be required to generate all stable models (in the case of `clingo` this can be made using the call option 0). If an atom A holds in all computed stable models of a program P we can say that it is a logical consequence of the program under the stable model semantics. This is briefly denoted as

$$P \models_{sm} A$$

Example 1. *Let us analyze the test n. 6 of the Mentor, namely*

> *There are two brothers, Romolo and Remo. One is always sincere and, conversely, the other is always a liar.*
>
> *If Romolo says that their mother is named Silvia and Remo says that she is blonde, it can be deduced with certainty that:*
> *A. if their mum is blonde, her name is not Silvia.*
> *B. if the name of their mum is Silvia, then her hair is blonde*
> *C. the name of their mother is Silvia and she is not blonde*
> *D. if the name of the mother is not Silvia, then she is not blonde*
> *E. their mother is not called Silvia*

It is interesting to see that ChatGPT answers C.[2] *Of course the correct answer is A. As a matter of fact, there*

[2]April 7, 2025. Used without login.

are two possible scenarios:

Romolo	*sincere*	*Silvia*	*liar*	*not Silvia*
Remo	*liar*	*not blonde*	*sincere*	*Blonde*

By case analysis it is immediate to realize that only the answer A is applicable.
The example can be encoded in ASP as follows:

```
%%% The two brothers
bro(romolo). bro(remo).
%%% Property: being sincere.
{sincere(X)} :- bro(X).
%%% Exactly one is sincere
h1 :- 1 { sincere(romolo); sincere(remo) } 1.
:- not h1.
```

After the first part we have to encode the fact that Romulus is sincere iff her mother is named Silvia (similarly for Remo and blonde).

```
%%% romolo is sincere iff the mother is Silvia
silvia :- sincere(romolo).
:- silvia, not sincere(romolo).
%%% similarly for remo
blonde :- sincere(remo).
:- blonde, not sincere(remo).
```

Then we have to encode the possible answers, assigning them a parameter.

```
%% A. The implication (blonde -> not silvia)
%%%% is viewed as  not blonde or not silvia
opt(a) :- not blonde.
opt(a) :- not silvia.
%% B. (silvia -> blonde)
opt(b) :- not silvia.
opt(b) :- blonde.
%% C. silvia and not blonde
opt(c) :- silvia, not blonde.
%%% D. (not silvia -> not blonde)
opt(d) :- silvia.
opt(d) :- not blonde.
%%% E. not silvia
opt(e) :- not silvia.
```

Running the code with clingo one realizes that `opt(a)` *is the unique option holding in all the (two) stable models (thus, it is a logical consequence under the stable model semantics).*

Example 2. *Let us analyze the test n. 49 of the Mentor, namely:*
Please select the correct negation of the sentence: Umberto has at least a blonde son*:*
A. At least one son of Umberto is not blonde
B. Umberto has no sons or he has only not blonde sons
C. All Umberto's sons are brown haired
D. Not all Umberto's sons are blonde
E. All Umberto's sons are red haired.
First of all we introduce the sentence about Umberto as a hint and we require that its negation is true:

```
i1 :- father(umberto,X), blonde(X).
%%%% It is impossible that the hint is true
%%%% (thus its complement must hold)
:- i1.
```

In order to generate possible scenarios we introduce some possible sons, each of them with their hair information.

```
{father(umberto,X)} :- peanut(X).
%%% Here they are
peanut(sally).   peanut(schroeder).   peanut(charlie).
peanut(lucy).    peanut(piperita).    peanut(peggy_jane).
%%% And their hair color
blonde(sally).   blonde(schroeder).   bald(charlie).
black(lucy).     brown(piperita).     red(peggy_jean).
```

Then we encode the various options:

```
%%%% (exists a non blonde son)
opt(a) :- father(umberto,X), not blonde(X).
%%%% A disjunction
parent :- father(umberto,X).
opt(b) :- not parent.
%%% In the case he has sons, they cannot be blonde
blondesons :- father(umberto,X), blonde(X).
opt(b) :- not blondesons.
%%%%% All the sons are brown - there is no son that is not brown
opt(c) :- not nonbrownsons.
nonbrownsons :- father(umberto,X), not brown(X).
%%% there is a son that is not blonde
opt(d) :- father(umberto,X), not blonde(X).
%%% Like opt(c) with a different color
opt(e) :- not nonredsons.
nonredsons :- father(umberto,X), not red(X).
```

The answer is option B which is true in all 16 stable models. ChatGPT answers correctly to this quiz.

4. TOLC-ASP

The application developed and presented in this section is based on a scheme of 12 quizzes drawn from the Mentor [3], namely those numbered 1, 6, 7, 20, 21, 23, 32, 41, 42, 48, and 49. A further quiz (we call it 0) is obtained from a sample freely downloadable from the CISIA website [2]. Tests 0, 1, 6, 7, 42 are without quantifiers, the others use quantifiers. Test 23 requires a "dual" answer.

The application was developed following the steps outlined below.

4.1. ASP Encoding

The twelve selected quizzes have been encoded in ASP along the lines sketched in the previous section. They have been solved with clingo, in particular using the option 0 to understand if a quiz is under constraints and in any case to verify if an answer is true in all possible stable models.

4.2. New Quizzes

Modifying the above encoded quizzes, new ones have been invented (and their encodings checked). Particular care has been posed in introducing parameters that allow to modify them with new names, actions, attributes and so on. A set of placeholders is used and some dataset for replacing the placeholders (e.g., with different human names) has been populated. The following example shows how the test n. 6 of the Mentor has been transformed into a template and how it is has been subsequently encoded.

Example 3. *There are two people, *name1* and *name2*. One is always sincere and, conversely, the other is always a liar. If *name1* says that *sentence1* and *name2* says that *sentence2*, it can be deduced with certainty that:*
*A. if *sentence2*, then it's not true that *sentence1*.*
*B. if *sentence1*, then it's true that *sentence2**
*C. it's true that *sentence1* and it's not true that *sentence2**
*D. if it's not true that *sentence1*, then it's not true that *sentence2**
*E. it's not true that *sentence1**

*While it is clear what *name1*, *name2* stand for, the placeholders *sentence1*, *sentence2* represent simple proposition of the kind subject-verb-complement (e.g. Adam eats an apple or Eva tricks the snake). The ASP encoding goes as follows:*

```
%%% The two people
person(name1). person(name2).
%%% Property: being sincere.
{sincere(X)} :- person(X).
%%% Exactly one is sincere
h1 :- 1 { sincere(name1); sincere(name2) } 1.
:- not h1.
%%% name1 is sincere iff sentence1
s1 :- sincere(name1).
:- s1, not sincere(name1).
%%% name2 is sincere iff sentence2
s2 :- sincere(name2).
:- s2, not sincere(name2).
%%% OPTIONS %%%
%% A. sentence2 -> not sentence1
opt(a) :- not s2.
opt(a) :- not s1.
%% B. sentence1 -> sentence2
opt(b) :- not s1.
opt(b) :- s2.
%% C. sentence1 and not sentence2
opt(c) :- s1, not s2.
%%% D. not sentence1 -> not sentence2
opt(d) :- s1.
opt(d) :- not s2.
%%% E. not sentence1
opt(e) :- not s1.
```

This way, starting from a limited number of predefined models (12 for now) and some data files listing names, actions and so on, it is possible to generate an exponential number of different quizzes obtained by instantiating the placeholders using a random generator.

4.3. Natural Language Generation

A check of the readability of the generated quizzes was made. Texts are in Italian and the use of conjunctive tense requires a further step to make the sentences clear and fluent. This could be made, in principle, using the help of a LLM.

Initially, an attempt was made to instantiate the quiz templates with real data using the Minerva LLM, developed by Sapienza NLP in collaboration with Future Artificial Intelligence Research (FAIR) and CINECA [11]. However, at the request to replace a template's placeholders with fabricated data, the model was unable to produce the desired outcome, returning a quiz identical to the original template. Since Minerva is rapidly evolving, another attempt will be made as soon as possible.

4.4. Error Correction

In every test (in a parametric way) we added some sentences with specific feedback for the users' incorrect answers. Similarly to what is done in the Mentor, we wanted to create a system that not only indicates the correct solution, but also explains the reason why an answer is wrong. This approach aims to promote active learning, helping users to understand their own errors and to refine their logical reasoning skills. Coding this feedback directly in ASP offered the advantage of being able to directly and automatically verify its logical correctness: by adding the ASP rules for the feedback of the wrong answers to the code and uncommenting (automatically) them one at a time, the output of clingo was useful to explicitly display the counterexamples to the corresponding answer option, thus demonstrating its erroneousness.

4.5. Automatization

A python program implementing the test generation (using a random generator) and the test execution was subsequently developed. This program allows users to visualize the quizzes one at a time, to select an answer and to receive a feedback. Users have an indefinite number of attempts at their disposal, so that they can retry the same puzzle as many times as they need to find the right answer.

Specifically, the program accepts a command-line argument `-feedback` that allows the user to specify whether they wish to receive an hint after each wrong answer. Each question template, organized in JSON files, contains placeholders that are replaced with values defined in a secondary dictionary containing sets of names, actions, attributes and other. As an example, we report the entry corresponding to the test n. 6 of the Mentor previously discussed.

```
{"id": 3,
 "source": 6,
 "text": "There are two people, *name1* and *name2*.\n
     One is always sincere and, conversely, the other is always a liar.\n
     If *name1* says that *sentence1* and *name2* says that *sentence2*,
     it can be deduced with certainty that:",
 "options": {
     "A": "if *sentence2*, then it's not true that *sentence1*",
     "B": "if *sentence1*, then it's true that *sentence2*",
     "C": "it's true that *sentence1* and it's not true that
          *sentence2*",
     "D": "if it's not true that *sentence1*, then it's not true
          that *sentence2*",
     "E": "it's not true that *sentence1*"
   },
   "exact": "A",
   "feedback":{
     "A": "Correct!",
     "B": "Wrong! *name1* and *name2* cannot be sincere at the same
          time",
     "C": "Wrong! There is a case where *name2* tells the truth and
          *name1* tells a lie",
     "D": "Wrong! *name1* and *name2* cannot be liar at the same
          time",
     "E": "Wrong! There is a case where *name1* is sincere"
   }}
```

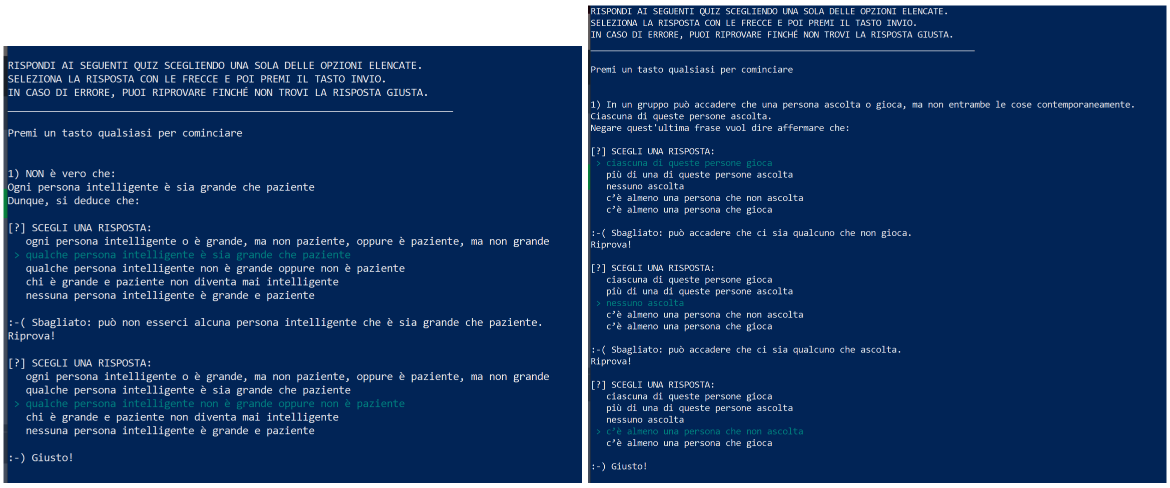
This process is handled by a function that instantiates a different version of each question, shuffles the answer choices and eventually adds the feedback message for each option. The presentation of questions to the user is made by another function that uses the `inquirer` package [12] to show the quiz sequentially. In particular, `inquirer` allows the users to select answers via keyboard arrows. After each question, users receive an immediate message about the correctness of their answer and the possibility to retry if it was wrong, accompanied by an hint if feedback was enabled. The program keeps asking the same question until the user selects the right answer. Not only the number of attempts, but also timestamps and identifiers for the question for each question are tracked in a file that is collected at the end of the session.

5. Experimental Results

Here we report on an experiment carried out testing the above program with first-year Computer Science students at the University of Udine who (more or less) volunteered to participate during their Programming Lab classes at the end of March 2025. The software was installed on several lab computers and each volunteer completed the test anonymously, being allowed to take as much time as needed, in order to simulate a training session rather than a formal assessment.

All participants received the same set of 12 question templates, each instantiated with variable data and appearing in random order. Moreover, the participants were divided into two groups: one group was given the quizzes with extended feedback, while another received only minimal feedback (right/wrong)—see also Figure 3. At the end of the test, students filled out an anonymous questionnaire, where they could express their opinions on the difficulty of the questions and the usefulness of the feedback. The aim of the experiment was, in fact, to evaluate the effectiveness of the feedback provided for incorrect answers and to gather some direct input from users.

A total of 42 students participated in the experiment, each taking an average of 10 to 20 minutes to complete the test: 22 out of 42 participants were given the "extended feedback" version, while 20 were



```
RISPONDI AI SEGUENTI QUIZ SCEGLIENDO UNA SOLA DELLE OPZIONI ELENCATE.
SELEZIONA LA RISPOSTA CON LE FRECCE E POI PREMI IL TASTO INVIO.
IN CASO DI ERRORE, PUOI RIPROVARE FINCHÉ NON TROVI LA RISPOSTA GIUSTA.
______________________________________________________________________

Premi un tasto qualsiasi per cominciare


1) NON è vero che:
Ogni persona intelligente è sia grande che paziente
Dunque, si deduce che:

[?] SCEGLI UNA RISPOSTA:
   ogni persona intelligente o è grande, ma non paziente, oppure è paziente, ma non grande
 > qualche persona intelligente è sia grande che paziente
   qualche persona intelligente non è grande oppure non è paziente
   chi è grande e paziente non diventa mai intelligente
   nessuna persona intelligente è grande e paziente

:-( Sbagliato: può non esserci alcuna persona intelligente che è sia grande che paziente.
Riprova!

[?] SCEGLI UNA RISPOSTA:
   ogni persona intelligente o è grande, ma non paziente, oppure è paziente, ma non grande
   qualche persona intelligente è sia grande che paziente
 > qualche persona intelligente non è grande oppure non è paziente
   chi è grande e paziente non diventa mai intelligente
   nessuna persona intelligente è grande e paziente

:-) Giusto!
```

```
RISPONDI AI SEGUENTI QUIZ SCEGLIENDO UNA SOLA DELLE OPZIONI ELENCATE.
SELEZIONA LA RISPOSTA CON LE FRECCE E POI PREMI IL TASTO INVIO.
IN CASO DI ERRORE, PUOI RIPROVARE FINCHÉ NON TROVI LA RISPOSTA GIUSTA.
______________________________________________________________________

Premi un tasto qualsiasi per cominciare


1) In un gruppo può accadere che una persona ascolta o gioca, ma non entrambe le cose contemporaneamente.
Ciascuna di queste persone ascolta.
Negare quest'ultima frase vuol dire affermare che:

[?] SCEGLI UNA RISPOSTA:
 > ciascuna di queste persone gioca
   più di una di queste persone ascolta
   nessuno ascolta
   c'è almeno una persona che non ascolta
   c'è almeno una persona che gioca

:-( Sbagliato: può accadere che ci sia qualcuno che non gioca.
Riprova!

[?] SCEGLI UNA RISPOSTA:
   ciascuna di queste persone gioca
   più di una di queste persone ascolta
 > nessuno ascolta
   c'è almeno una persona che non ascolta
   c'è almeno una persona che gioca

:-( Sbagliato: può accadere che ci sia qualcuno che ascolta.
Riprova!

[?] SCEGLI UNA RISPOSTA:
   ciascuna di queste persone gioca
   più di una di queste persone ascolta
   nessuno ascolta
 > c'è almeno una persona che non ascolta
   c'è almeno una persona che gioca

:-) Giusto!
```

Figure 3: Two (starting) executions of the tool with the feedback flag set to yes

given the “minimal feedback” version. This uneven distribution is due to the fact that students had to take turns using a computer and those who received minimal feedback took more time to complete the test than the others.

Although the small size of the sample does not allow for statistically meaningful conclusions, some observations can still be made and may serve as a basis for future improvements. On average, the performance between the two groups was relatively similar, with a slight advantage in favor of the group with “extended feedback”: students belonging to the group with “minimal feedback” generally needed more attempts to get the right answer. In detail, the statistics for the number of correct answers on the first attempt are as follows

- extended feedback: mean $= 5.72$, standard deviation $= 2.49$, maximum $= 11$;
- minimal feedback: mean $= 6.1$, standard deviation $= 2.51$, maximum $= 10$.

For the total number of failed attempts, the results show

- extended feedback: mean $= 13, 14$, standard deviation $= 6.39$, maximum $= 24$;
- minimal feedback: mean $= 14.25$, standard deviation $= 6.98$, maximum $= 26$.

The feedback appears to have had a moderate impact. The questionnaires completed at the end of the test indicate that most students found the test to be moderately difficult and thought the feedback was of limited usefulness.

Given these outcomes and the lack of a clear distinction between the two groups, one could argue that the role of feedback is marginal. However, we remain convinced of its importance in a learning tool of this kind: these results simply point out the need to refine the way feedback is designed and delivered, in order to improve its effectiveness and perceived value.

6. Side Work

During the analysis of the TOLC problems, we investigated some well-known logical puzzles and asked some LLMs to solve them, in order to test the models’ proficiency on this kind of logical reasoning task. In particular, we analyzed the famous Zebra Puzzle [13] and we considered three of the most utilized LLMs nowadays: ChatGPT, Gemini and Deepseek. Since this puzzle is already well known in the literature, the models had an easier time solving it. To challenge the LLMs, then, we invented a variant of the same puzzle that says:

> 1. In the wonderful lagoon of Grado there are 4 houses of different colors, listed from west to east. [3]
> 2. In each house lives a family that speaks a different dialect.
> 3. Each family drinks one and only one beverage.
> 4. The Gradese does not live in the pink house.
> 5. The Istrian lives next to the Carnic.
> 6. The Gradese drinks spritz.
> 7. The Carnic does not drink water. Never.
> 8. Near the Gradese, people drink either water or beer.
> 9. Whoever drinks Ribolla does not live in a red or green house.
> 10. The white house is, from left to right, between the green and the red one.
> 11. The person in the white house lives further west than the one who doesn’t drink alcohol.
> 12. The Gradese and the Bisiaco are not neighbors.
> QUESTION: Who lives in the white house?

There is only one possible correct answer, and it is that the Carnic lives in the white house. This answer was obtained encoding the puzzle in ASP and running clingo whose output is:

[3]Actually the correct formulation was “casone” namely small buldings made of wood and hay where fishermen lived

```
lives(1, Gradese, green, spritz)  lives(2, Carnic, white, beer)
lives(3, Istrian, red, water)  lives(4, Bisiaco, pink, ribolla)
```

here 1 is the number assigned to the first house we find to the west and 4 is assigned to the last house to the east.

The answers given by the models mentioned above are the following:

- Gemini[4] answers that the Gradese lives in the white house and argues incorrectly violating several conditions of the riddle;
- ChatGPT[5] offers a more detailed argumentation but its answer - The Istrian lives in the white house - is still wrong; in particular, it seems to misinterpret constraint 10), which is commonly understood to mean that the white house must be between the green and red houses, in left-to-right order and in a consecutive manner;
- DeepSeek[6] provides the correct answer, but its solution shows that constraint 10) is not respected; this is due to the model's difficulty in interpreting the phrase "from left to right" as "from west to east".

In conclusion, no one of these models is completely reliable, but some have better logical reasoning abilities than others. DeepSeek R1 is the model that gets closer to the correct and complete solution, but its reliability depends on how clearly the problem is stated.

7. Conclusions

With the growing popularity of large language models (LLMs), it is natural for students to consider chatbots like ChatGPT, Gemini or Deepseek R1 as potential tools to support their preparation for tests and exams. However, at present, these models do not guarantee a sufficient level of reliability, as their responses are not always consistent or correct. In this scenario, a structured system such as the one described above represents a more robust alternative.

Developing a digital version of the Mentor involves several challenges, including:

- selecting an appropriate collection of questions,
- ensuring sufficient variety in the questions,
- guaranteeing that the answers are consistent and that there is always a single correct answer,
- generating clear and effective feedback for users.

All of these aspects have been addressed in the previous sections. Moreover, thanks to the direct involvement of students in the experimental phase, valuable suggestions emerged for further improvements of the tool. Among the most relevant proposals are: the introduction of a more intuitive and accessible graphical interface; the refinement of the feedback mechanisms, both for incorrect and correct attempts, to better clarify the underlying logical reasoning; the integration of features designed for students with special educational needs (BES). Other improvements could include expanding the repository of templates by adding new quizzes, including combinatorial puzzles.

These insights open up promising avenues for the future evolution of the software, with the goal of meeting the real needs of the users.

Acknowledgments

We would like to express our gratitude to Giovanna D'Agostino for sharing with us her expertise. We thank Dario Della Monica and Claudio Mirolo for allowing us to test the tool during their lab classes and all the (anonymous) students of the first year of the Laurea L31 of the University of Udine that accepted to participate to the experiment.

[4]premium version of December 2024
[5]-o1 premium version of December 2024
[6]R1 free version of February 2025

Declaration on Generative AI

The author(s) have not employed any Generative AI tools.

References

[1] Giorgio Filippi, Vincenzo Falco, I risultati TOLC 2023, https://www.cisiaonline.it/sites/default/files/Divulgazione/2024/Volume-Risultati_TOLC_2023-CISIA.pdf, 2024.

[2] Area riservata test CISIA, https://testcisia.it/studenti_tolc/login_sso.php, 2025.

[3] L. Caire, Paola Suria Arnaldi, Mentor di Logica vol.1, Cisia, 2018.

[4] V. W. Marek, M. Truszczynski, Stable models and an alternative logic programming paradigm, in: K. R. Apt, V. W. Marek, M. Truszczynski, D. S. Warren (Eds.), The Logic Programming Paradigm - A 25-Year Perspective, Artificial Intelligence, Springer, 1999, pp. 375–398. URL: https://doi.org/10.1007/978-3-642-60085-2_17. doi:10.1007/978-3-642-60085-2_17.

[5] A. Dovier, A. Formisano, E. Pontelli, An empirical study of constraint logic programming and answer set programming solutions of combinatorial problems, J. Exp. Theor. Artif. Intell. 21 (2009) 79–121. URL: https://doi.org/10.1080/09528130701538174. doi:10.1080/09528130701538174.

[6] L. Cian, T. Dreossi, A. Dovier, Modeling and solving the rush hour puzzle, in: R. Calegari, G. Ciatto, A. Omicini (Eds.), Proceedings of the 37th Italian Conference on Computational Logic, Bologna, Italy, June 29 - July 1, 2022, volume 3204 of *CEUR Workshop Proceedings*, CEUR-WS.org, 2022, pp. 294–306. URL: https://ceur-ws.org/Vol-3204/paper_29.pdf.

[7] N. Rizzo, A. Dovier, 3cosoku and its declarative modeling, J. Log. Comput. 32 (2022) 307–330. URL: https://doi.org/10.1093/logcom/exab086. doi:10.1093/LOGCOM/EXAB086.

[8] M. Alviano, L. L. T. Trieu, T. C. Son, M. Balduccini, The XAI system for answer set programming xasp2, J. Log. Comput. 34 (2024) 1500–1525. URL: https://doi.org/10.1093/logcom/exae036. doi:10.1093/LOGCOM/EXAE036.

[9] A. Dovier, P. Benoli, M. C. Brocato, L. Dereani, F. Tabacco, Reasoning in High Schools: Do it with ASP!, in: C. Fiorentini, A. Momigliano (Eds.), Proceedings of the 31st Italian Conference on Computational Logic, Milano, Italy, June 20-22, 2016, volume 1645 of *CEUR Workshop Proceedings*, CEUR-WS.org, 2016, pp. 205–213. URL: https://ceur-ws.org/Vol-1645/paper_9.pdf.

[10] M. Gelfond, V. Lifschitz, The stable model semantics for logic programming, in: R. A. Kowalski, K. A. Bowen (Eds.), Logic Programming, Proceedings of the Fifth International Conference and Symposium, Seattle, Washington, USA, August 15-19, 1988 (2 Volumes), MIT Press, 1988, pp. 1070–1080.

[11] Minerva, https://nlp.uniroma1.it/minerva/, 2025.

[12] inquirer 3.4.0, https://pypi.org/project/inquirer/, 2025.

[13] Zebra Puzzle - Wikipedia, https://en.wikipedia.org/wiki/Zebra_Puzzle, 2025.