Eng Int Syst (2010) 0: 1–11

Engineering
Intelligent Systems

A web-based example-tracing tutor for formalising sentences in first order logic

Themistoklis Chronopoulos, Ioannis Hatzilygeroudis, Isidoros Perikos and Kostantinos Kovas

Department of Computer Engineering & Informatics, School of Engineering, University of Patras, Greece.
E-mail: {chronop, ihatz, perikos, kobas}@ceid.upatras.gr

In this paper we present a web-based Example-tracing Tutor for learning how to formalise sentences in first order logic (FOL) which is a basic knowledge representation language. The tutor has been developed using the Cognitive Tutoring Authoring Tool (CTAT) and consists of a full web-based student interface that has been designed according to a structured process of formalising sentences in FOL language. Formalising process is driven by a behaviour graph in which examples of converting sentences from natural language (NL) to FOL, of varying difficulty, have been authored and annotated with hints and misconceptions. Small scale evaluation has given quite satisfactory results.

Keywords: example-tracing tutor, formalisation in logic, knowledge representation tutor.

1. INTRODUCTION

Knowledge Representation & Reasoning (KR&R) is a fundamental topic of Artificial Intelligence (AI). A basic KR language is First-Order Logic (FOL), the main representative of logic-based representation languages, which is part of almost any introductory AI course and textbook [1, 2]. To make automated inferences, Clause Form (CF), a special form of FOL, is used. Teaching FOL as a knowledge representation and reasoning language includes many aspects. One of them is translating natural language (NL) sentences into FOL formulas. It is an ad-hoc process; there is no specific algorithm that can be automated within a computer. This is mainly due to the fact that NL has no clear semantics as FOL does. Also, most of existing textbooks do not pay the required attention to that. They simply provide the syntax of FOL and definitions of the logical symbols and terms [1, 2]. Even more specialized textbooks do the same [3]. At best, they provide a kind of more extended explanations and examples [4]. They do not provide any systematic guidance towards it. Given the above, students usually find difficulties in learning the task of formalizing NL sentences in FOL, which confronts to tutors' common experience.

In [5], we introduced a structured process for guiding students in translating a NL sentence into a FOL one, namely the NL to FOL SIP process. In [6], we present a web-based system implementing the SIP process, i.e. helping students in learning how to convert NL sentences into FOL formulas. Having used the above system for some time, we resulted in the following findings: (a) At a first stage, students may not

be necessary to work with sentences that produce formulas with more than three groups of atoms or with more than one group of formulas. (b) Tutors would like to use a graphical way of describing the SIP steps for each formula and a way of massively inserting them. Also, they would like related hints or feedback messages to be presented to the users in case of errors.

Example-tracing tutors are problem-specific tutors that provide guidance to the students during problem-solving practice by comparing their solution steps to the right solving process of an example that has been recorded in the tutor. They provide hints and error feedback messages, and are flexible to handle multiple solution strategies and paths. [7]

CTAT (Cognitive Tutor Authoring Tools) is an authoring tool for creating example-tracing tutors that satisfies the prementioned requirements [8, 9, 10].

In a recent work [17], we used CTAT to develop an example tracing tutor for the conversion of sentences with simple semantics, from NL to FOL, based on a simplified version of the above NL to FOL SIP process. So tutoring in this stand alone version was limited to a small range of examples.

In this paper, we present a web-based example-tracing tutor which consits of a student interface and a behavior graph for the guided formalising of sentences with richer semantics in FOL. In this version, student interface is expanded to include also the loops of steps which are described in the full NL to FOL SIP process. The sentences that have been used as examples for the formalising process are of varying difficulty.

2. RELATED WORK

There are some systems like Logic Tutor [11], Logic-ITA [12] and P-Logic Tutor [13] which, although they deal with learning and/or teaching logic, they are not concerned with how to use predicate logic as a KR&R language and, most importatnt, they do not deal with how to formalize a NL sentence into FOL.

KRRT (Knowledge Representation and Reasoning Tutor) [14] is a web-based system that aims at helping students to learn FOL as a KR&R language. It deals with both knowledge representation in and reasoning with FOL. The translation form NL to FOL takes place in its KR part. However, the only help provided to the students is at syntactic and logical equivalence levels. The student gives his/her FOL proposal sentence and the system checks its syntax to see whether it is the correct one (here equivalent sentences are acceptable). However, it does not provide any guidance about how to make that translation or even what is the kind of error made.

In [6] a web based system for teaching natural language to first order logic is presented. This interactive system aims at helping students to learn how to convert/translate natural language to first order logic. It tries to achieve it by proving a NL to FOL structured and interactive process (NLtoFOL SIP) for the conversion and (b) guidance and help during the stages of that process. The system during the interaction with the students can adapt the help and the guidance provided according to the student's needs and errors. Also the user interface is dynamically configured during the user interaction to reflect the steps of NLtoFOL SIP process.

The system in [6] has the same objectives as the system presented here, but there are also significant differences that concern (a) the user interface, (b) the way it works internally for student interaction checking, (c) the way hints/help are/is structured and (d) the way new sentences are inserted. The system in [6] is not an example-tracing or cognitive tutor. It is based on a different methodology; it's a web-based interactive and intelligent system.

3. A STRUCTURED AND INTERACTIVE PROCESS FOR NL TO FOL CONVERSION

The process of formalising FOL sentences concerns translation of a given NL sentence into a FOL formula. The main problem in converting natural language into first order logic has to do with the unclear semantics that natural language has. Natural language has no clear semantics as FOL has. However, the main difficulty comes from the lack of a systematic way of making the conversion. The NL to FOL structured and interactive process (NLtoFOL SIP) was initially introduced in [5]. It is a process that guides a student in translating/converting a NL sentence into a FOL one, via the following steps:

1. Spot the verbs, the nouns and the adjectives in the sentence and specify the corresponding predicates or function symbols.
2. Specify the number, the types and the symbols of the arguments of the function symbols (first) and the predicates (next).
3. Specify the quantifiers of the variables.
4. Construct the atomic expressions (or atoms) corresponding to predicates.
5. Divide produced atoms in groups of the same level atoms.
6. Specify the connectives between atoms of each group and create corresponding logical formulas.
7. Divide produced formulas in groups of the same level formulas.
8. If only one group of formulas is produced, specify the connectives between formulas of the group, create the next level formula and go to step 10.
9. Specify the connectives between formulas of each group, create the next level formulas and go to step 7.
10. Place quantifiers in the right points in the produced formula to create the final FOL formula.

To demonstrate the steps of the above process, we present the conversion of the NL sentence "**All farmers who own donkeys beat them**" into a FOL formula.

Step 1. *Spot the verbs, the nouns and the adjectives in the sentence and specify the corresponding predicates or function symbols*.

There are four such items here:

farmers ← predicate: *farmer*

donkeys ← predicate: *donkey*

own ← predicate: *owns*

beat ← predicate: *beats*

Step 2. *Specify the number, the types and the symbols of the arguments of the function symbols (first) and the predicates (next).*

We do that in the following table:

Predicate	Arity	Types	Symbols
farmer	1	*variable*	*x*
donkey	1	*variable*	*y*
owns	2	*variable, variable*	*x*, *y*
beats	2	*variable, variable*	*x*, *y*

Step 3. *Specify the quantifiers of the variables.*

x ← ∀ (because of "All")

y ← ∀ (because of "them", which denotes an "all" for "donkeys")

Step 4. *Construct the atomic expressions (or atoms) corresponding to predicates.*

We construct as many atoms as the predicates:

Atom 1: *farmer*(*x*)

Atom 2: *donkey(y)*

Atom 3: *owns(x,y)*

Atom 4: *beats(x, y)*

Step 5. *Divide produced atoms in groups of the same level atoms.*

This mainly refers to grouping atoms that should be connected with each other with some connective:

AtomGroup1: {*farmer(x)*, *donkey(y), owns(x,y)*}

AtomGroup2: {*beats(x,y)*}

Or alternatively

AtomGroup1: {*farmer(x)*}

AtomGroup2: {*donkey(y), owns(x,y)*}

AtomGroup3: {*beats(x,y)*}

Step 6. *Specify the connectives between atoms of each group and create corresponding logical formulas.*

We form the formulas corresponding to the groups of step 5.

AtomGroup1 ← Form1: *farmer(x)* ∧g*donkey(y)*g ∧ g*owns(x,y)*

AtomGroup2 ← Form2: *beats(x,y)*

Or alternatively

AtomGroup1 ← Form1: *farmer(x)*

AtomGroup2 ← Form2: *donkey(y)*g ∧ g*owns(x,y)*

AtomGroup3 ← Form3: *beats(x,y)*

Step 7. *Divide produced formulas in groups of the same level formulas.*

This usually corresponds to specifying the left and right parts of an implication:

GroupForm1-1: {*farmer(x)* ∧g*donkey(y)*g∧g*owns(x,y)*}

GroupForm2-1: {*beats(x,y)*}

Or alternatively

GroupForm1-1: {*farmer(x)*}

GroupForm2-1: {*donkey(y)*g∧*owns(x,y), beats(x,y)*}

Step 8. *If only one group of formulas is produced, specify the connectives between formulas of the group, create the next level formula and go to step 10.*

Not applicable.

Step 9. *Specify the connectives between formulas of each group, create the next level formulas and go to step 7.*

GroupForm1-2: *farmer(x)* ∧g*donkey(y)*g ∧ g*owns(x,y)*

GroupForm2-2: *beats(x,y)*

Or alternatively

GroupForm1-2 ← Form1-2: *farmer (x)*

GroupForm2-2 ← Form2-2: (*donkey(y)*∧*owns(x,y)*) ⇒ *beats(x,y)*

Step 7.

GroupForm1-2: {(*farmer(x)* ∧g*donkey(y)*g ∧ g*owns(x,y)*), *beats(x,y)*}

Or alternatively

GroupForm1-2: {*farmer(x), (donkey(y)*g∧*owns(x,y))* ⇒*beats(x,y)*}

Step 8.

GroupForm1-2: ←Form1-3: {(*farmer(x)* ∧g*donkey(y)*g ∧ g*owns(x,y))* ⇒ *beats(x,y)*}

Or alternatively

GroupForm1-2: ←Form1-3: {*farmer(x)* ⇒ *((farmer (x)*g∧*owns(x,y))* ⇒ *beats(x,y))*}

Step 10. *Place quantifiers in the right places in the produced formula to create the final FOL formula.*

The produced final FOL sentence is as follows:

$\mathfrak{o}ix)\ (\mathfrak{i}y)\ (farmer(x) \wedge \mathfrak{g}donkey(y)\mathfrak{g} \wedge \mathfrak{g}owns(x,y)) \Rightarrow beats(x,y)$

Or alternatively

$\mathfrak{o}ix)\ ((farmer(x) \Rightarrow (\mathfrak{i}y)\ ((donkey(y) \wedge owns(x,\ y)) \Rightarrow beats(x,y)))$

4. EXAMPLE-TRACING TUTORS

The Cognitive Tutor Authoring Tools (CTAT) [8] is a set of tools that support creation of two types of tutors: example-tracing tutors and cognitive tutors. Example-tracing tutors, in contrast to cognitive tutors or other tutors such as constraint-based tutors, compare student problem-solving steps against the solutions steps of selected problems that have been demonstrated by the author and recorded in a behavior graph [7]. This kind of tutors, provide a step-by-step guidance during the problem-solving process and they support alternative problem solving strategies. Example-tracing tutors are problem-specific and require no AI programming since they are based on tracing specific pre-configured examples [9], whereas cognitive tutors require AI programming and they are based on a rule-based cognitive model [10]. Example-tracing tutors are easy to implement, but provide less flexibility, whereas cognitive tutors is quite more difficult to build, but can be quite more flexible.

Developing an example-tracing tutor in CTAT involves the following steps [7]:

1. Creation of the graphical student interface.

2. Demonstration and recording of alternative correct and incorrect solutions of a problem in a Behavior Graph.

3. Annotation of the solutions steps in the Behavior Graph with hint or error feedback messages.

GUI Builder, is a tool of CTAT for building the user interface of the tutor in the first step. The author with GUI builder can create the tutor interface through drag-and-drop techniques from a "recordable widget" palette added to Java NetBeans.

Additionally, CTAT offers another tool, the *Behavior Recorder*, for building the "behavior graph" of a problem, in which alternate correct and incorrect solution steps are recorded, by the form of nodes and links. Each link represents a solution step and is associated with a corresponding item of the user interface. A correct link can be annotated with hints, whereas an incorrect link can be annotated with an error feedback message.

When the student uses the tutor, CTAT's example-tracing engine implements the example-tracing function which compares each user's problem solving step to the corresponding step in the behavior graph. Based on the results, the tutor provides the appropriate feedback (accepts or rejects the answer and also provides error feedback messages for incorrect student action links).

5. A WEB-BASED EXAMPLE-TRACING TUTOR FOR FORMALISING SENTENCES

In a previous work, we used CTAT for building an example-tracing tutor for the learning process of the convertion of a sentence from NL to FOL, in order to systematically analyze a large number of examples to extract possible cognitive patterns, and also to systematically analyze various types of hints or feedback needed.

More specifically, in [17], we implemented a stand-alone version of an example-tracing tutor for the conversion of sentences from natural language to FOL, according to a structured and interactive process, which was somewhat simplified from that described in 3, as it refers to sentences that result in formulas with at most one group of formulas.

In this paper we present a web-based version of this tutor to support delivering a sequence of tutor problems on the web and experimentation: on-line tests, as well as log recording and analysis.

More specifically, the web-based tutor can provide the following capabilities for teachers and students:

- Teacher can easily update the behavior graph adding new examples or hints since the behavior graph is located on the server.
- Students can use the tutor without time or spatial restrictions.
- Users can log-in the system and by that is guaranteed the uniqueness of each student session
- Teacher can access, through log files on the server, the information about the student's learning progress

Finally, and most important, we have extended the student interface to include formalising of sentences that could result in formulas with two groups of formulas and a more complex behavior graph has been created to include the conversion of more sentences with a varying degree of difficulty. To facilitate authoring of more sentences we have also used a template-based approach using Mass Production facility of CTAT [8].

6. THE STUDENT INTERFACE

According to the process presented in Section 4, we first created the student interface of our system to reflect the NLtoFOL SIP process, as shown in Fig. 1. Actually a separate student interface template has been implemented for each step or group of steps of the process. All those student interface templates are integrated into one interface as different step tabs, through which a student can try to convert a NL sentence into a FOL formula following the NLtoFOL SIP process. Each tab, except first, corresponds to a step of the NLtoFOL SIP process. The first tab ("Atoms") corresponds to steps 1-4 of the process. In each problem solving cycle the student follows the NLtoFOL SIP process selecting one tab at a time, selecting on it the interface elements to work on and performing a problem solving action.

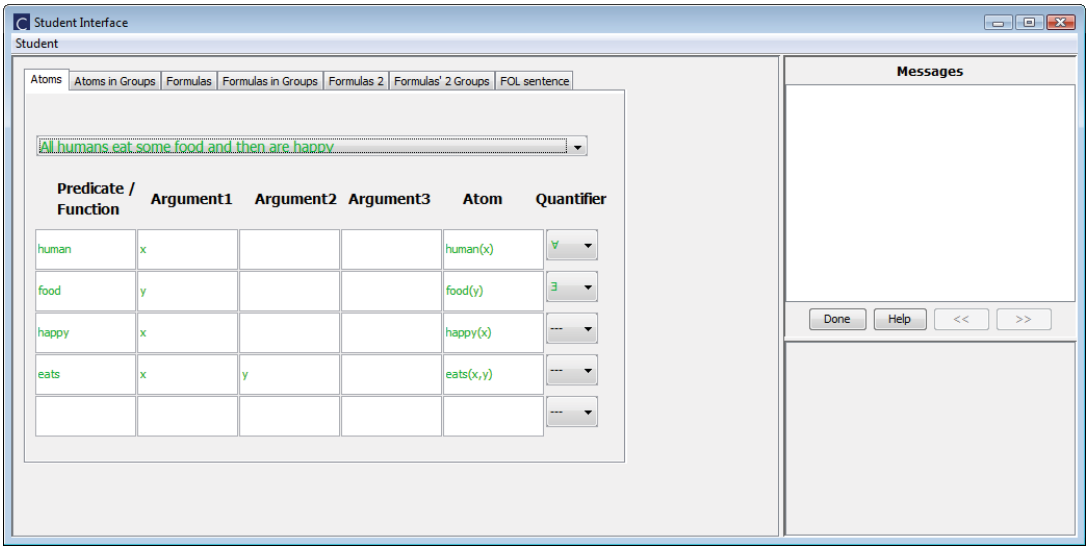


Figure 1 Student graphical web interface.

7. THE BEHAVIOR GRAPH

In the next step, both correct and incorrect problem-solving behavior of several sentences, were recorded in the Behavior Graph, by the Behavior Recorder of CTAT. As a result there were created as many behavior sub-graphs as the sentences. Also, alternatives solution paths for the translations of the same sentences, where applicable, have been recorded as alternative solutions paths. For example, in Figure 2 each subgraph refers to the translation of a different sentence or to an alternative translation of the same sentence (e.g. states 400 and 430) or to a bugy state (e.g. state 138). These cases are used as the basis for Example-Tracing Tutors to provide guidance to students.

The student can select any sentence from the interface of the system (Figure 4). The sentences are presented to the user sorted by their difficulty of the conversion process. There are three levels of difficulty (easy, medium and advanced).

As we have already mentioned, at steps 7 to 9, groups of the same level formulas of a sentence are constructed. Iteration of these steps is necessary in case we have to convert sentences that have more than two levels of groups of formulas. Thus, complex sentences that have many levels of groups of formulas are more difficult to convert from NL into FOL formulas. The level of a sentence's difficulty mainly depends on the number of the groups of formulas it includes. More complex sentences require iteration of theses stages in order to form the FOL formula of the sentence. The difficulty levels are "easy", "medium" and "advanced". We characterize as el"easy" sentences that don't include groups of formulas. Thus their conversion process doesn't involve steps 7-9. The next difficulty level is "medium". Sentences that their conversion process requires implementation of steps 7-9 exactly one time are of medium level. Finally, we characterize as "advanced" sentences that their conversion process involves two or more iterations of steps 7-9. These are more complex sentences with many levels of atoms and groups and probably many logical connectives. Also, there are other factors for difficulty level characterization, like e.g. the number of quantifiers and the number of alternative (equivalent) FOL formulas.

In Table 1, we present some sentences, that have been stored for use in the system, and their difficulty level, based on the total number of quantifiers of their FOL formulas, the number of loops that are necessary to convert the sentence to a FOL formula and the number of alternatives FOL formulas that can be produced.

8. ANNOTATION OF SOLUTION STEPS

CTAT's Example-Tracing Engine compares the student's problem-solving steps to the solution steps which have been recorded in the Behavior Graph and guides him/her through the solution path. When a student's answer or selection reaches a correct action-state of the graph then a positive feedback is provided, otherwise a negative feedback is returned to the student. Tracing the student's step-by-step solution enables the tutor to provide individualized instruction in the problem solving context.

When a student's answer reaches a state of the graph which has been marked as incorrect action, then the attached "buggy" error feedback message is returned to the student. Hints are provided by the tutor almost on each link of the graph, but they are displayed only on student request [15].

The NLtoFOL tutor provides feedback on each problem solving action, by accepting correct actions, which is shown by green color to the student and tagging errors instead of accepting them, which is shown by red color to the student.

8.1 Incorrect steps

In general, any student input that is not recognized by the tutor is marked as incorrect, but *by defining incorrect steps* in the graph, the tutor will be able to provide a customized error feedback message for the specified input. In each message we have included an example to demonstrate the correct use.

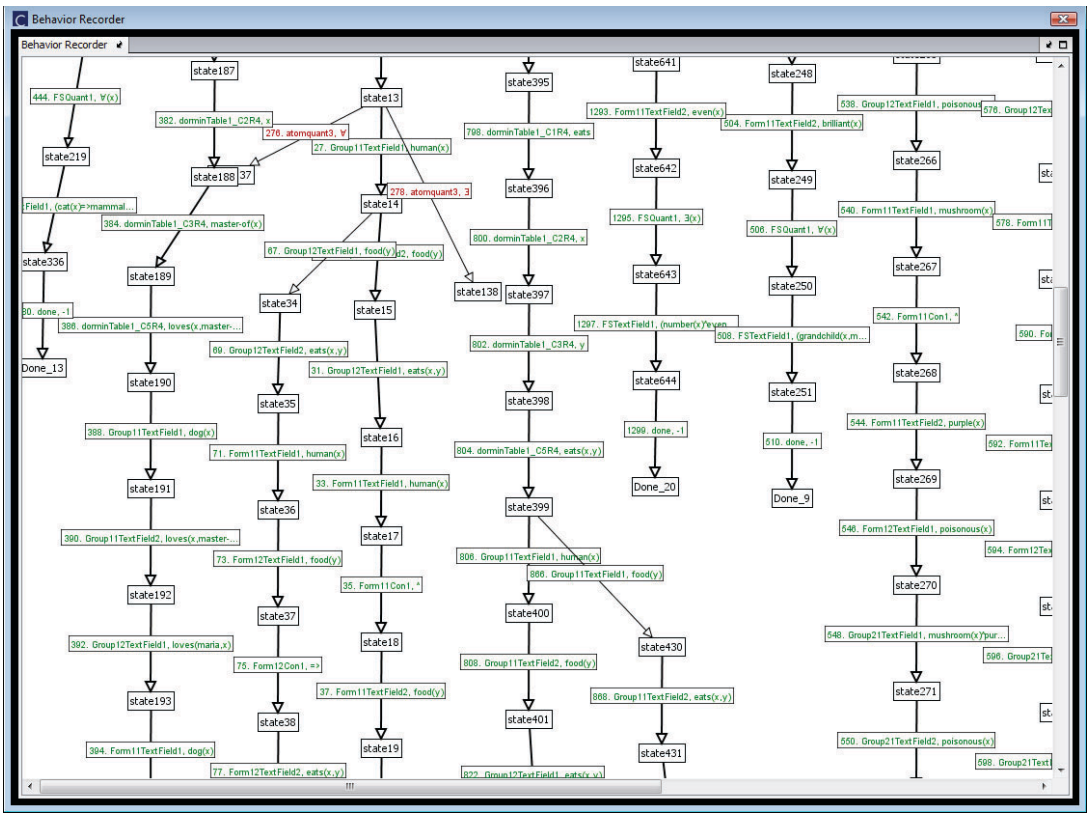


Figure 2 A part of the Behavior Graph.

Table 1 Example sentences and their difficulty factors.

Sentence	#Quantifiers	#Loops	#Alternatives	Difficulty Level
Some numbers are even	1	0	0	easy
All cats are mammals	1	0	0	easy
No purple mushroom is poisonous	1	1	1	medium
All human eats some food	2	1	1	medium
Every farmer who owns a donkey beats it	2	2	2	advanced

We focus on *common errors* that happen at the conversion of sentences from natural language to first order logic language, such as:

- Misuse of AND connective

  Example: “All dogs love playing games.”

  Common error: (∀x) dog(x) ∧ loves(x, playing_games)

  Right formula: (∀x) dog(x) ⇒ loves(x, playing_games)

- The order of quantifiers

  Example: “All love somebody.”

  Common error: (∃ y) (∀ x) loves(x, y)

  Right formula: (∀ x) (∃ y) loves(x, y)

- Use of function

  Example: “Pluto loves its master.”

  Common error: (∀x) master(x, pluto) ⇒ loves(pluto, x)

  Right formula: loves(pluto, master_of(pluto))

- Grouping atoms of the same level

- Grouping formulas of the same level

For example, in Figure 4 the way of recording right and incorrect steps is displayed. The right selection is state209 and a “buggy” state referring to a false selection of quantifier is state646. If student reaches this state, a suitable message will be issued.

We also demonstrated the tutor cases of *errors that are related to the sentence*. For example in the sentence “All humans eat some food”, someone can characterize the “All” as predicate. In such errors the tutor gives feedback that is related to the theory of FOL language. The example-tracing tutor doesn’t accept the answer in which the student fills *partially the right answer* e.g. fills the right predicate but in the wrong number (“humans” instead of “human”).

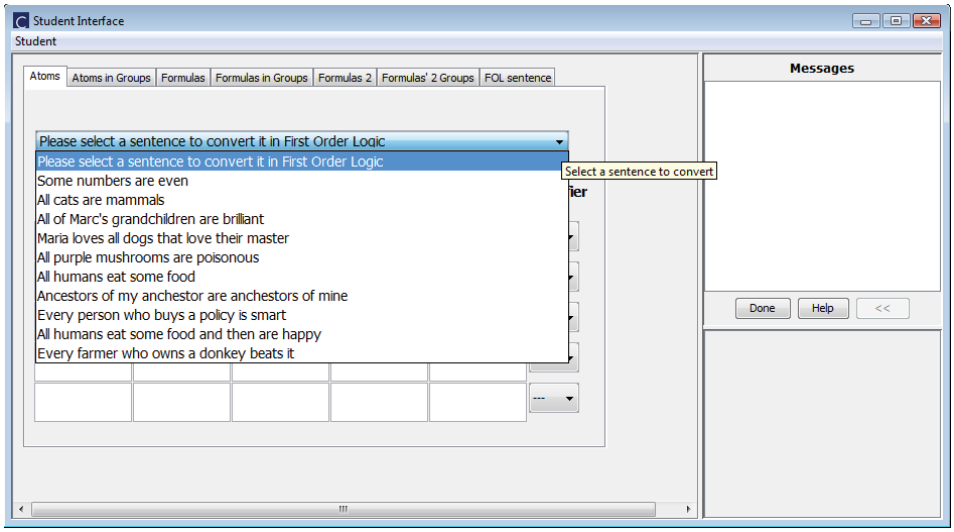


Figure 3 Example sentences included in the example-tracing tutor.

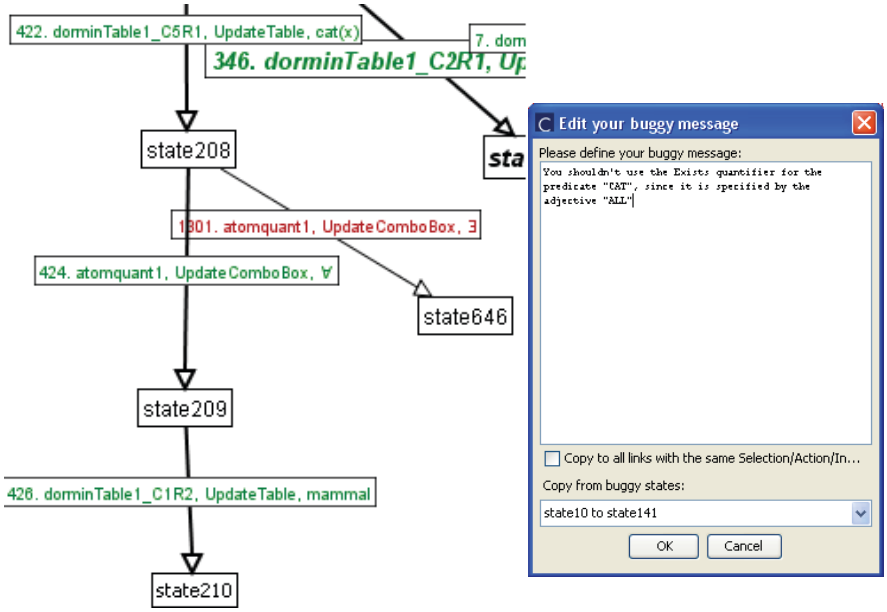


Figure 4 A part of the behavior graph recording correct and incorrect states.

8.2 Annotation of Hints

There are several factors that may affect the choice of a specific hint: tutoring topic, tutoring context, tutoring history, student's answer, and so on. First, to be pedagogically useful, a hint has to be related to the tutoring topic and be useful in helping the student find the expected answer. So the tutoring topic is important. [16]

Our tutor provides advice for building a FOL sentence, upon request of the student. We have implemented *four levels of advice* available for each step of the conversion of the sentence (see Figure 5 for an example). *The first level* reminds or advises the student on the corresponding goal according to the NLtoFOL SIP and a general description of how to achieve the goal. *The second level* provides a hint from the theoretical context of first order logic (definitions, syntactic etc) that is related to the corresponding step. The *third level* provides a hint specific to the case by providing a similar example. Finally, *the fourth level* provides concrete advice on solving the goal in the current context by suggesting the correct solution.

8.3 Knowledge labels

After the completion of the behavior graph of the tutor, we assigned knowledge labels to links in the behavior graph, to represent the skills required for each step of the formalising process.

We have added the following knowledge labels to links in the behavior graph: *FindPredicate*, *FindArgument*, *SpecifyQuantifier*, *ConstructAtom*, *FormulateGroup*, *SpecifyConnective*, *ConstructFormula*.

This is a form of cognitive task analysis, since we determine how the overall problem-solving skill breaks down into smaller components.

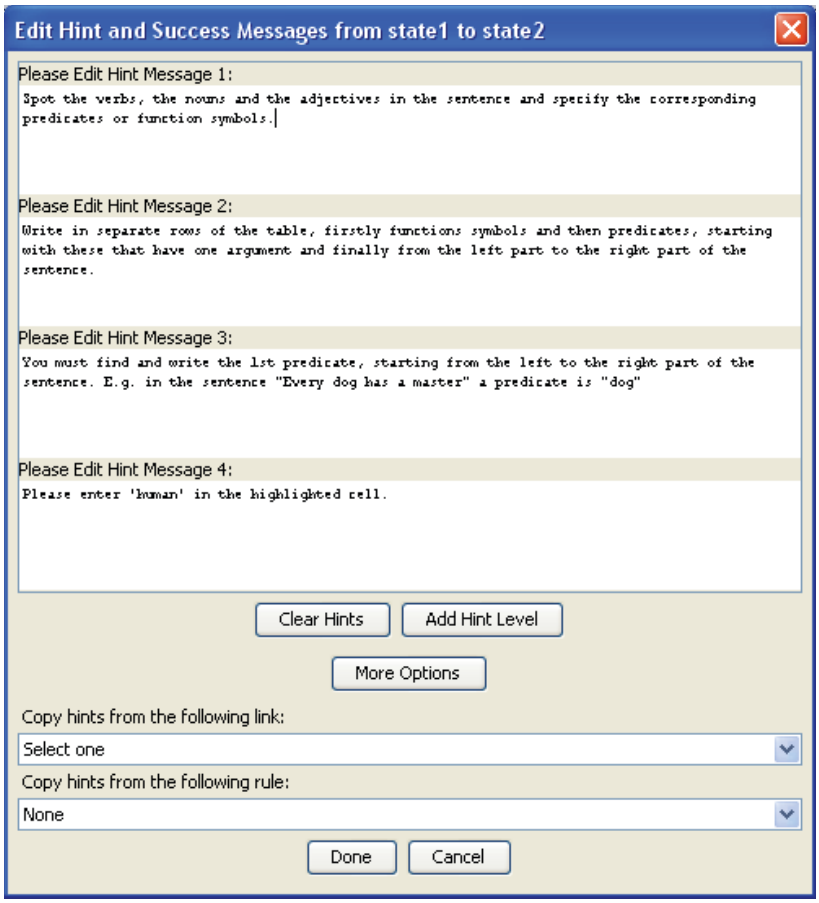


Figure 5 An example set of hints (step 1 of NLtoFOL SIP).

First of all this process can speed up the development of the tutor, since it provides a way to copy hint messages from one step to a similar step.

But, most important, when the tutor is used to evaluate the knowledge of the students (by suppressing student feedback), knowledge labels assist in knowledge tracing.

At the same time, it is a way of planning the cognitive model, since knowledge labels can be used, to create production rules corresponding to each identified skill. [9]

9. MASS PRODUCTION OF THE BEHAVIOR GRAPH

To facilitate the creation of many tutored conversions of sentences from NL to FOL, we have used a facility that CTAT provides for template-based Mass Production.

The idea behind Mass Production is to help an author generate many problems of the same structure without having to demonstrate solutions to each of them.

First, we created a behavior graph template, for the conversion of similar or near similar sentences, substituting the sentence-specific values of the behavior graph (such as the values of steps, hints, and error feedback messages) with variables (Figure 6), and then we defined in the spreadsheet, the values of the variables required for the conversion of each sentence (Figure 7).

Finally, we merged the problems table and behavior graph template and a behavior graph with mass-produced behavior sub-graphs was generated, simplifying and speeding-up the whole authoring process.

10. EVALUATION

We made a small scale evaluation of the system in our Department. Ten students that had been taught about logic during the course lectures were instructed to use the system and then fill in a questionnaire including questions for evaluating usability and learning. The questionnaire included eight questions. The questions 1-3 were based on the following scale [1: a little, 2: neutral, 3: very much]. The results are shown in Table 2.

Finally, questions 4–8 were of open type and concerned strong and weak points or problems faced in using the system. Ten students filled in the questionnaire. Their answers showed that the students in general were helped in learning the conversion of sentences and found the system very interesting to use (70%), although the interface was not considered to be easy to learn. In fact, many students (40%) agreed that it took them more than 20 minutes to get familiar with it. Finally, the majority of the students (over 70%) would recommend the system to the next year's students.

We also conducted a second experiment. We created two groups of ten students each, randomly selected. One group,

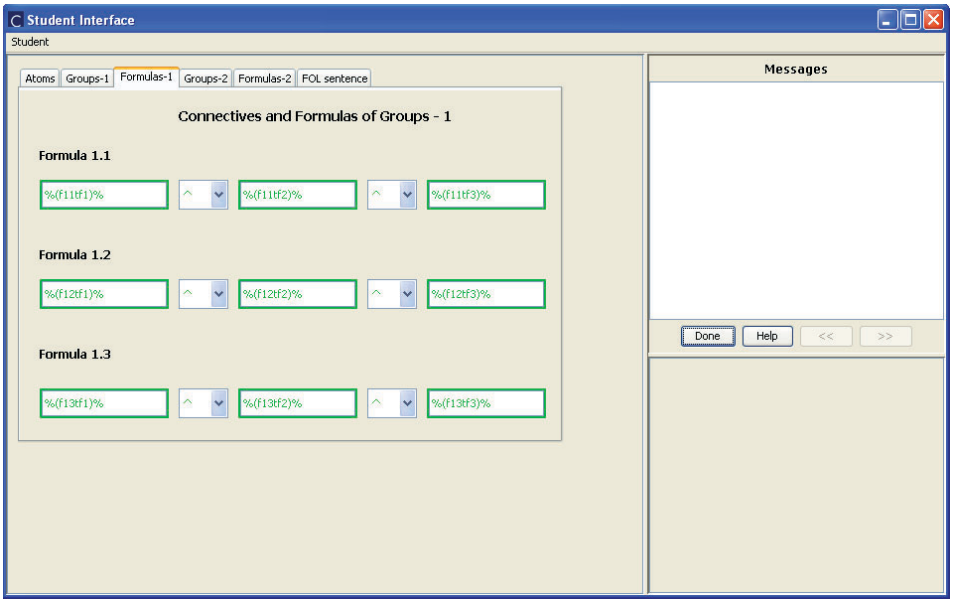


Figure 6 Constructing the behavior graph template for mass production.

Table 2 Questionnaire Results.

Q	Questionnaire			
	Questions	***Answers (%) Total Students 10***		
		a little	neutral	very much
1	Did you find the system inteface easy to use?	20	60	20
2	How much did the system help you to learn the convertion process?	0	30	70
3	Did you find the system interesting to use?	0	40	60

Table 3 Evaluation Test Results.

Group	**Correctly converted sentences (%)**
Lecture based learning	35
Plus System based learning	65

apart from lectures, used the system for learning the NL to FOL conversion, whereas the other didn't. After the study, the students took a test: to convert seven NL sentences into FOL ones. The results, which show the usefulness of the system, are presented in Table 3. The large difference between the performances of the two groups is due not only to the improvement of the skill of the second group in formalizing sentences, but also to the increase of their interest in doing that (as indicated from the results of Table 2) and to the availability of many examples for practicing.

Although our evaluation is based on relatively small groups of students, we believe that its findings are adequately reliable. They are consistent with the intuition that use of a suitable tool, apart from lectures, will give better results.

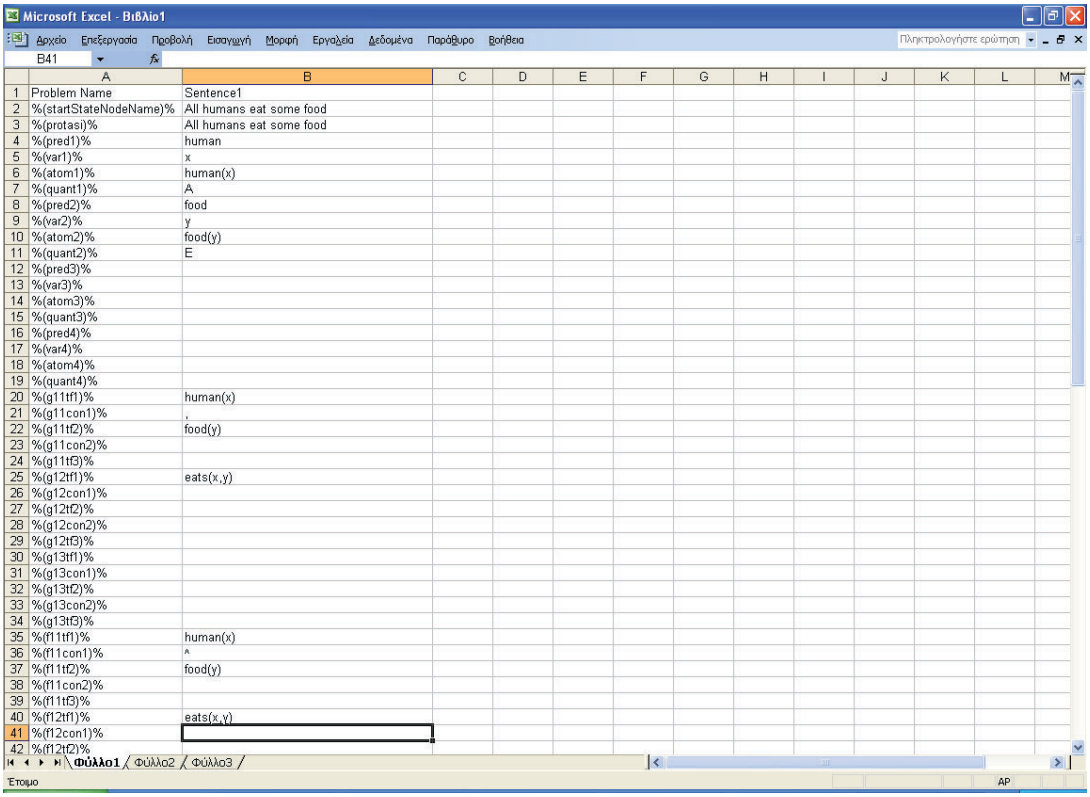
Additionally, we asked the opinion of a few tutors of AI courses and they agreed that the system lacks a more convenient way to add new sentences and examples to it.

11. CONCLUSIONS

In this paper, we present a web-based example-tracing tutor for the conversion of NL sentences into FOL formulas. This topic is one of the important ones in teaching logic as a knowledge representation language. The conversion is based on a structured and interactive process, called the NLtoFOL SIP.

To follow that process, we constructed a suitable interface that reflects the steps of the process, using CTAT. We also designed worked-out examples of conversions for several sentences of different levels of difficulty and created corresponding behavior graphs with the Behavior Recorder. In the behavior graph not only correct steps are recorded, but also incorrect ones, based on common errors. Furthermore, four levels of help to the students have been identified and recorded as annotated hints.

The system has been tested by a group of ten students and the results are quite satisfactory. A problem identified was some difficulty in getting quickly familiar with the system and the process.



	A	B
1	Problem Name	Sentence1
2	%(startStateNodeName)%	All humans eat some food
3	%(protasi)%	All humans eat some food
4	%(pred1)%	human
5	%(var1)%	x
6	%(atom1)%	human(x)
7	%(quant1)%	A
8	%(pred2)%	food
9	%(var2)%	y
10	%(atom2)%	food(y)
11	%(quant2)%	E
12	%(pred3)%	
13	%(var3)%	
14	%(atom3)%	
15	%(quant3)%	
16	%(pred4)%	
17	%(var4)%	
18	%(atom4)%	
19	%(quant4)%	
20	%(g11tf1)%	human(x)
21	%(g11con1)%	,
22	%(g11tf2)%	food(y)
23	%(g11con2)%	
24	%(g11tf3)%	
25	%(g12tf1)%	eats(x,y)
26	%(g12con1)%	
27	%(g12tf2)%	
28	%(g12con2)%	
29	%(g12tf3)%	
30	%(g13tf1)%	
31	%(g13con1)%	
32	%(g13tf2)%	
33	%(g13con2)%	
34	%(g13tf3)%	
35	%(f11tf1)%	human(x)
36	%(f11con1)%	^
37	%(f11tf2)%	food(y)
38	%(f11con2)%	
39	%(f11tf3)%	
40	%(f12tf1)%	eats(x,y)
41	%(f12con1)%	
42	%(f12tf2)%	

Figure 7 A spreadsheet with the required data for the conversion of a given sentence.

Example-Tracing Tutors are quite inflexible as far as internal representation of the behavior graph is concerned. We sometimes need some more flexibility on that. For example, one cannot distinguish the type of error in a predicate (whether it is on syntax or wrong word). Cognitive tutors are another type of tutors provided by CTAT. However, in contrast to example-based tutors, rule-based programming is required to implement a tutoring model. However, they provide more flexibility in the tutoring process with regards to the guidance offered to the user. So, the next step to follow this work is the design and implementation of a cognitive tutor. The already designed examples will be of help to this direction.

REFERENCES

1. Russell, S, Norvig, P (2003). *Artificial Intelligence: a modern approach*. 2nd Edition. Upper Saddle River, NJ, USA: Prentice Hall.
2. Luger, GF (2004). *Artificial Intelligence: Structures and Strategies for Complex Problem Solving*. 5th Edition, Addison-Wesley.
3. Brachman, RJ, Levesque, HJ (2004). *Knowledge Representation and Reasoning*. Elsevier.
4. Genesereth, M R, Nilsson, NJ (1987). *Logical Foundations of AI*. Morgan Kaufmann, Palo Alto.
5. Hatzilygeroudis, I (2007). *Teaching NL to FOL and FOL to CL Conversions*. Proceedings of the 20th International FLAIRS Conf., Key West, FL, May 2007, AAAI Press, pp. 309–314.
6. Hatzilygeroudis, I, Perikos, I (2009). *A Web-Based Interactive System for Learning NL to FOL Conversion*. In: Damiani, E, Jeong, J, Howlett, RJ and Jain, LC (Eds.). New Directions in Intelligent Interactive Multimedia Systems and Services – 2, SCI 266, Springer-Verlag, pp. 297–307.
7. Aleven, V, McLaren, BM, Sewall, J, Koedinger, KR (2009). A new paradigm for intelligent tutoring systems: Example-tracing tutors. *International Journal of Artificial Intelligence in Education*, 19(2), pp. 105–154.
8. Aleven, V, McLaren, BM, Sewall, J, Koedinger, KR (2006). *The Cognitive Tutor Authoring Tools (CTAT): Preliminary Evaluation of Efficiency Gains*. In: the Proceedings of the 8th International Conference on Intelligent Tutoring Systems (ITS-06), pp. 61–70.
9. Koedinger, K, Aleven, V, Heffernan, N, McLaren, B, Hockenberry, M (2004). *Opening the door to non-programmers: authoring intelligent tutor behavior by demonstration*. Proceedings ITS-2004, pp. 162–174, Berlin: Springer.
10. Koedinger, KR, Aleven, V (2003). *Toward a Rapid Development Environment for Cognitive Tutors*. In U. Hoppe, F. Verdejo, J. Kay (Eds.), Proceedings of the 11th International Conference on Artificial Intelligence in Education, AI-ED 2003, pp. 455–457.
11. Abraham, D, Crawford, L, Lesta, L, Merceron, A, Yacef, K (2001). The Logic Tutor: A multimedia presentation. *Electronic Journal of Computer–Enhanced Learning*.
12. Abraham, D and Yacef, K (2002). *Adaptation in the Web-Based Logic-ITA*. In: De Bra, P., Brusilovsky, P., Conejo, R. (Eds). AH 2002, LNCS 2347, pp. 456–461.
13. Lukins, S, Levicki, A, Burg, J (2002). *A tutorial program for propositional logic with human/computer interactive learning*. In: SIGCSE 2002, pp. 381–385. ACM, NY.
14. Alonso, JA, Aranda, GA, Martın.-Matceos, FJ (2007). *KRRT: Knowledge Representation and Reasoning Tutor*. In: Proceedings of EUROCAST 2007, LNCA 4739, pp. 400–407. Springer-Verlag, Berlin Heidelberg.

15. Aleven, V, Sewall, J, McLaren, BM, Koedinger, KR (2006). *Rapid Authoring of Intelligent Tutors for Real-World and Experimental Use*. Proceedings of 6th IEEE international conference on advanced learning technologies (ICALT 2006), pp. 847–851.
16. Zhou, Y, Freedman, R, Glass, M, Michael, JA, Rovick, AA, Evens, MW (1999). *Delivering Hints in a Dialogue-Based Intelligent Tutoring System*. Proceedings of the Sixteenth National Conference on Artificial Intelligence (AAAI-99), pp.128–134.
17. Chronopoulos, T, Perikos, I, Hatzilygeroydis, I (2010). *An example-tracing tutor for teaching NL to FOL conversion*. Proceedings of the 6^{th} IFIP WG 12.5 International Conference, AIAI2010, Larnaca, Cyprus, October 2010.