LOGI: A didactic tool for a beginners' course in logic (system description)

Mario Ornaghi, Camillo Fiorentini, Alberto Momigliano

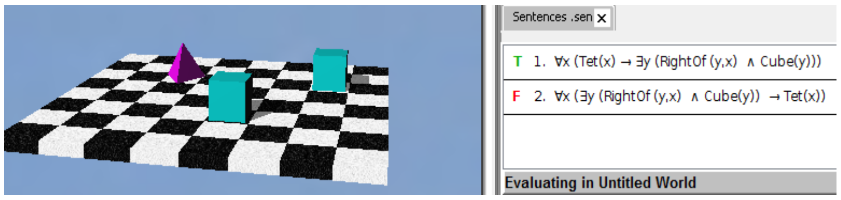
Dipartimento di Informatica, Università degli Studi di Milano, Italy

Abstract. LOGI is a didactic software designed for a basic course of Logic. We developed it as an extension of the course-ware *Language, Proof and Logic* (LPL) by Plummer, Barwise, and Etchemendy, in particular of *Tarski's World* (TW): the latter features simple *virtual worlds* populated by blocks and equipped with a first order language to about them; this helps students to understand the language of logic and its semantics in an intuitive and effective way. A limit is that there is no automatic support for exercises involving other domains. More importantly, if we view logic as a paradigmatic modeling language, this aspect is lost in the predefined blocks world, while it should be one of the educational outcomes of a course of Logic in a Computer Science curriculum. To overcome those limits, LOGI provides an environment where students can model simple "realities" by devising an appropriate (one- or many-sorted) signature, writing finite interpretations that represent possible circumstances, and evaluating the truth/falsity of first order formulas in an interpretation according to Tarski's semantics. Furthermore, it provides a detailed explanation of the result of evaluating a formula in a set of interpretations. Finally, to stress the role of logic in modeling, LOGI supports explicit definitions and allows the user to deal with incomplete information. In this sense, LOGI is not a competitor of TW, but an extension, to be used as a second, more advanced step.

1 Motivation

LOGI is a didactic tool designed for "Logica Matematica", a first year course of the "Corso di Laurea in Informatica" at the University of Milan. The decision of developing LOGI followed three years of experience using the course-ware associated to *Language, Proof and Logic* [1] (LPL, `https://ggweb.gradegrinder.net/lpl`). This is one of the most widely used logic course-ware: it is available since 2002 and includes, among other tools, "Tarski's World" (TW), which provides a virtual environment for students to draw on the screen virtual worlds populated by blocks (see Fig. 1); the aim is to learn the formal language of logic and understand its semantics in an easy and intuitive way.

> "There are two ways to learn a second language. One is to learn how to translate sentences of the language to and from sentences of your native language. The other is to learn by using the language directly. ...In LPL, we adopt the second method for learning FOL [first-order logic]. Tarski's World provides a simple environment in which FOL can be used in many of the ways that we use our native language. ...With LPL, students learn not


Sentences .sen

T 1. $\forall x\, (Tet(x) \rightarrow \exists y\, (RightOf(y,x) \land Cube(y)))$

F 2. $\forall x\, (\exists y\, (RightOf(y,x) \land Cube(y)) \rightarrow Tet(x))$

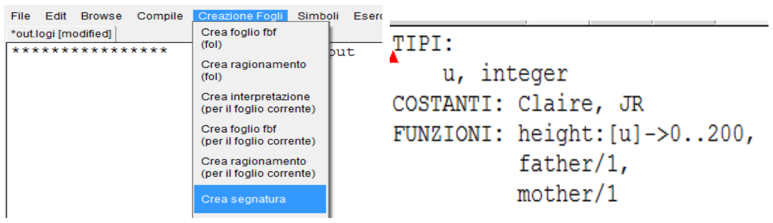
Evaluating in Untitled World

Fig. 1. A Tarski's World screenshot

> just how to prove consequences of premises, but also the equally important technique of showing that a given claim does not follow logically from its premises. To do this, they learn how to give counterexamples, which are really proofs of non-consequence. These will *often* [our emphasis] be given using Tarski's World" ([1], pages 14–5).

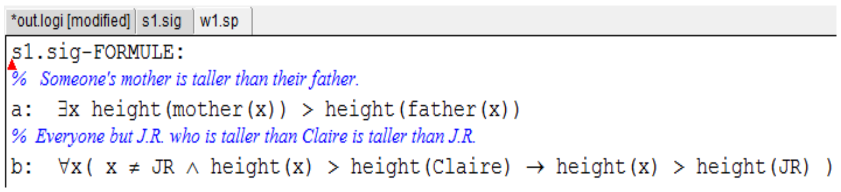
In our experience, TW effectively helps students in gaining some understanding of logic and students do like it. LPL contains a ton of exercises that can be developed and checked via the tools and we used them in lectures and as support for homework. However, most exercises involve signatures and interpretations *different* from TW. For example, there are exercises requiring to translate English statements into FOL, "introducing names, predicates, and function symbols as needed", considering other first order languages and interpretations, or asking to prove that an argument valid in TW is not valid in FOL, etc. These exercises have to be solved by hand. It will come to nobody's surprise that many students present very confused and imprecise solutions, which are in turn difficult to grade. Furthermore, students themselves would like to have a tool to check the correctness of their homework. LOGI aims to address those issues by: *a) Defining first order (possibly many sorted) signatures*, which can be stored and reused. *b) Writing formulas and arguments in the language of a chosen signature.* LOGI checks the correctness of formulas with respect to the signature. If no signature is declared, LOGI tries to infer one from the formulas. *c) Introducing finite interpretations of the chosen signature.* LOGI checks that interpretations are correctly formulated. *d) Evaluating the truth/falsity of formulas in an interpretation or in a set of interpretations.* LOGI shows also a detailed explanation of the computation of an evaluation. *e) Supporting explicit definitions.*

An additional motivation for LOGI is to emphasize the role of *modeling* in software development: FOL can be seen as a *paradigmatic modeling language* with a consolidated formal semantics. LOGI allows students to formalize "contexts", that is the set of "circumstances" which we are reasoning about. In LOGI a formal model of a context is obtained by introducing a signature that allows the user to represent all the circumstances of interest as first order interpretations and to express all related properties by primitively or explicitly defined symbols. Furthermore, to help modeling large examples, LOGI handles *incomplete* information, in the form of partially defined interpretations as well as of partially defined functions, corresponding to situations of partial knowledge typical of software systems. Note that since we also allow integers as a fixed structure, our system is outside of the strict limits of first-order logic.


File
Edit
Browse
Compile
Creazione Fogli
Simboli
Eserc
*out.logi [modified]
Crea foglio fbf (fol)
Crea ragionamento (fol)
Crea interpretazione (per il foglio corrente)
Crea foglio fbf (per il foglio corrente)
Crea ragionamento (per il foglio corrente)
Crea segnatura

```
TIPI:
    u, integer
COSTANTI: Claire, JR
FUNZIONI: height:[u]->0..200,
          father/1,
          mother/1
```

Fig. 2. Creating a new signature `s1.sig`



```
s1.sig-FORMULE:
% Someone's mother is taller than their father.
a:  ∃x height(mother(x)) > height(father(x))
% Everyone but J.R. who is taller than Claire is taller than J.R.
b:  ∀x( x ≠ JR ∧ height(x) > height(Claire) → height(x) > height(JR) )
```

Fig. 3. Sheet `w1.sp` of formulas with signature `s1.sig`

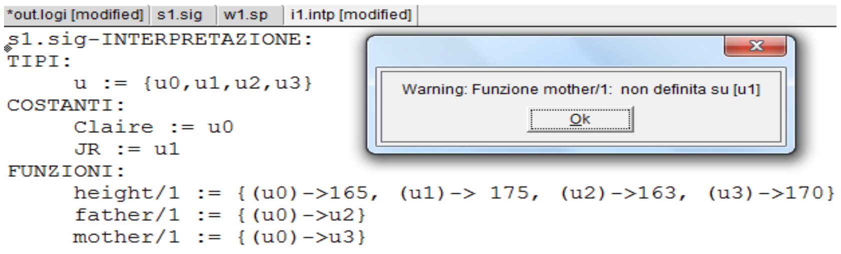
2 Using LOGI

To give a feel for the use of LOGI, we take on Exercise 11.30 of LPL. It requires to "translate various statements into a version of FOL that has function symbols height, mother, and father, the predicate $>$, and names for the people mentioned, and to say which sentences are true", in a (informally) given interpretation.

The first step is to create the required signature via the button "Crea Segnatura" in the "Creazione Fogli" menu (Fig. 2, left). Sheets are the basic units of the tool and an exercise involves the creation of a set of sheets of different kinds: `.sig` for signatures, `.sp` for formulas of a specific signature, `.intp` for interpretations and so on. Fig. 2, right, shows how to give a (many sorted) first order signature. A signature always includes `u`, the "universe" of discourse, while in this example we also have integers, used here to interpret the `height` function. In LOGI, by declaring the sort `integer` the predefined structure of integers is automatically included. If types in constant, function or predicate declarations are different from `u`, they have to be explicitly declared (see the declaration of `height` in Fig. 2); otherwise `u` is assumed. The syntax of declarations is quite intuitive and the tool clearly reports mistakes by highlighting.

The second step is to write down the required translations (Fig. 3). We consider here only two of the statements of the exercise, shown in the comments. Formulas are checked with respect to both FOL syntax and the declared signature. Once the formulas are correct, the sheet is accepted.

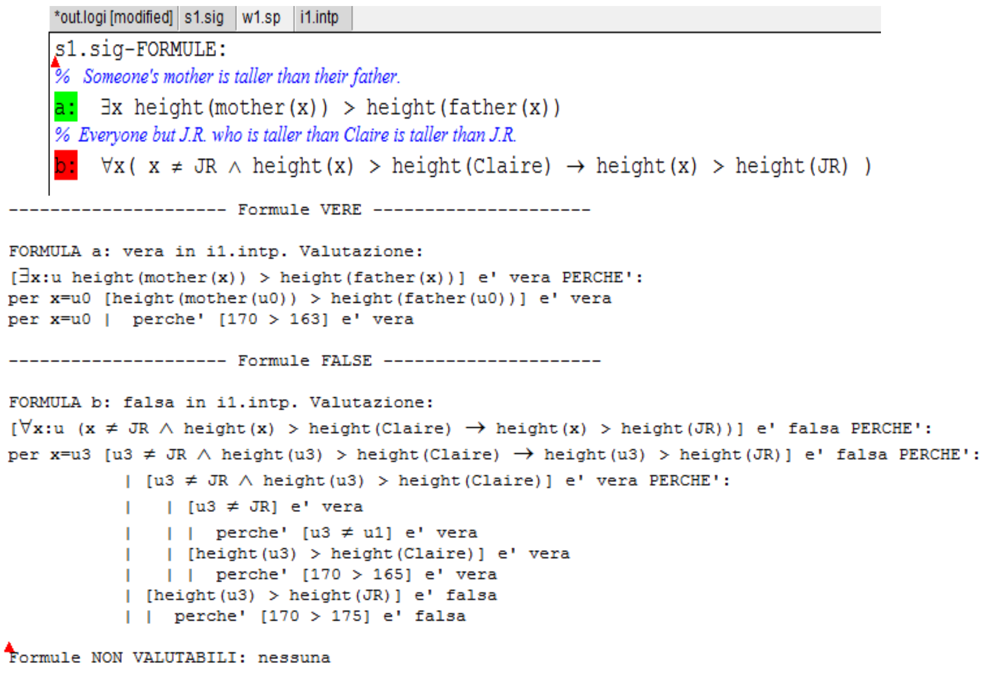
The third step is to formalize the required interpretation, a fragment of which is shown in Fig. 4. Mistakes, such as for example multi-valued functions, are signaled by error messages. In the case of *partially* defined information, the tool just issues a warning, see again Fig. 4. Finally, in exercises such as the one considered here, the last step is to evaluate the truth of formulas in a given sheet w.r.t. the considered interpretation. This is done via the "Esercizi" menu (details omitted). The result is



```
s1.sig-INTERPRETAZIONE:
TIPI:
      u := {u0,u1,u2,u3}
COSTANTI:
      Claire := u0
      JR := u1
FUNZIONI:
      height/1 := {(u0)->165, (u1)-> 175, (u2)->163, (u3)->170}
      father/1 := {(u0)->u2}
      mother/1 := {(u0)->u3}
```

```
Warning: Funzione mother/1:  non definita su [u1]
```

Fig. 4. Interpretation `i1.intp` of the signature `s1.sig`



```
s1.sig-FORMULE:
% Someone's mother is taller than their father.
a:  ∃x height(mother(x)) > height(father(x))
% Everyone but J.R. who is taller than Claire is taller than J.R.
b:  ∀x( x ≠ JR ∧ height(x) > height(Claire) → height(x) > height(JR) )
```

```
--------------------- Formule VERE ---------------------

FORMULA a: vera in i1.intp. Valutazione:
[∃x:u height(mother(x)) > height(father(x))] e' vera PERCHE':
per x=u0 [height(mother(u0)) > height(father(u0))] e' vera
per x=u0 |  perche' [170 > 163] e' vera

--------------------- Formule FALSE ---------------------

FORMULA b: falsa in i1.intp. Valutazione:
[∀x:u (x ≠ JR ∧ height(x) > height(Claire) → height(x) > height(JR))] e' falsa PERCHE':
per x=u3 [u3 ≠ JR ∧ height(u3) > height(Claire) → height(u3) > height(JR)] e' falsa PERCHE':
           | [u3 ≠ JR ∧ height(u3) > height(Claire)] e' vera PERCHE':
           |   | [u3 ≠ JR] e' vera
           |   | |  perche' [u3 ≠ u1] e' vera
           |   | [height(u3) > height(Claire)] e' vera
           |   | |  perche' [170 > 165] e' vera
           | [height(u3) > height(JR)] e' falsa
           | |  perche' [170 > 175] e' falsa

Formule NON VALUTABILI: nessuna
```

Fig. 5. Truth evaluation

shown in Fig. 5: green indicates truth, red falsity and grey that the interpretation lacks the information needed to evaluate the formula. The details of the evaluation are shown on the *output sheet* `*out.logi` of the tool, as shown at the bottom of Fig. 5. Because of the pedagogic nature of LOGI, we paid particular care to what we can call *truth verbalization*, that is explaining the truth-value of a formula in a given interpretation. Logically, this is accomplished recording during the interpretation, and then verbalizing *evaluation forms* [2], which are essentially a kind of realizers of given formulas.

3 The architecture of LOGI

The tool is entirely developed in SWI Prolog — LPL's Java code basis being proprietary — and consists of circa 6000 lines of code, including extensive comments. It contains three main "components": `application`, `parser`, and `GUI`, trying to bend SWI's limited module system to approximate as much as possible the notion of software components; this in order to separate the basic functionalities from the concrete representation of the sheets and from the GUI, simplifying future improvements and modifications. The information flow is the following:

Sheet on the GUI $\leftrightarrow_{(1)}$ list(char) for the parser $\leftrightarrow_{(2)}$ Prolog term for the application

The `GUI` is responsible for the translation (1). Different GUIs may behave in different ways, but all have to produce the same representation for the parser. The current GUI is based on XPCE-Emacs. For better rendering of logical symbols, we use different fonts and encodings for Windows and Unix systems. A custom written DCG `parser`, with a reasonable error messaging, is in charge of the translation (2). We refer to that as a "compilation".

Finally, the `application` component contains the basic functionalities. It defines the Prolog representation of signatures, interpretations and formulas, it checks the conformance of formulas and interpretations to a signature and evaluates the truth-value of formulas in an interpretation. Moreover, a `compilation_manager` module maintains a dynamic database of the compiled sheets and their dependencies: for example, if a signature sheet is modified, the interpretation and formula sheets depending on it in general are no longer correct and thus are deleted from the compiled ones.

4 Conclusions

We have presented LOGI, a didactic tool for a beginners' course in Logic. Being a prototype, LOGI has not been tested in the classroom yet, but we plan to experiment with it in the next edition and collect data, hopefully showing that it can improve students' performances.

We recall that we have designed LOGI to complement TW, addressing the same kinds of problems with the same approach, but opening up the choice of language. From this viewpoint, consistently with LPL's choice, we are not interested (for the time being) in other significant issues in a Logic course, such as automatic proofs and/or counter-model generation; for this we refer to existing tools, which can conceivably be integrated in a later version.

The code for both Windows and for Linux can be found at `https://homes.di.unimi.it/ornaghi/logi`, together with some limited documentation.

References

1. D. Barker-Plummer, J. Barwise, and J. Etchemendy. *Language, Proof, and Logic: Second Edition.* Center for the Study of Language and Information/SRI, 2nd edition, 2011.
2. P. Miglioli, U. Moscato, M. Ornaghi, S. Quazza, and G. Usberti. Some results on intermediate constructive logics. *Notre Dame Journal of Formal Logic*, 30(4):543–562, 1989.