Introduction to Iltis: An Interactive, Web-Based System for Teaching Logic*

Gaetano Geck
TU Dortmund
Germany
gaetano.geck@tu-dortmund.de

Artur Ljulin
TU Dortmund
Germany
artur.ljulin@tu-dortmund.de

Sebastian Peter
TU Dortmund
Germany
sebastian.peter@tu-dortmund.de

Jonas Schmidt
TU Dortmund
Germany
jonas2.schmidt@tu-dortmund.de

Fabian Vehlken
TU Dortmund
Germany
fabian.vehlken@tu-dortmund.de

Thomas Zeume
TU Dortmund
Germany
thomas.zeume@tu-dortmund.de

ABSTRACT

Logic is a foundation for many modern areas of computer science. In artificial intelligence, as a basis of database query languages, as well as in formal software and hardware verification — modelling scenarios using logical formalisms and inferring new knowledge are important skills for going-to-be computer scientists.

The Iltis project aims at providing a web-based, interactive system that supports teaching logical methods. In particular the system shall (a) support to learn to model knowledge and to infer new knowledge using propositional logic, modal logic and first-order logic, and (b) provide immediate feedback and support to students.

This article presents a prototypical system that currently supports the above tasks for propositional logic. First impressions on its use in a second year logic course for computer science students are reported.

CCS CONCEPTS

•**Applied computing** → **Interactive learning environments;**
•**Theory of computation** → **Logic;**

KEYWORDS

Logic, interactive learning environment

ACM Reference format:
Gaetano Geck, Artur Ljulin, Sebastian Peter, Jonas Schmidt, Fabian Vehlken, and Thomas Zeume. 2018. Introduction to Iltis:
An Interactive, Web-Based System for Teaching Logic. In *Proceedings of 23rd Annual ACM Conference on Innovation and Technology in Computer Science Education, Larnaca, Cyprus, July 2–4, 2018 (ITiCSE'18),* 7 pages.

*The authors acknowledge the financial support from the state of North Rhine-Westphalia in the form of funding for the improvement of university education as well as the financial support by a Fellowship for Innovation in Digital University Education by the state of North Rhine-Westphalia and the Stifterverband held by the last author.

ITiCSE'18, Larnaca, Cyprus

978-1-4503-5707-4/18/07…$15.00
DOI: 10.1145/3197091.3197095

DOI: 10.1145/3197091.3197095

1 INTRODUCTION

Logical formalisms play an important role in many modern computer science applications. In artificial intelligence, knowledge is often modelled by logical formulas and logical inference mechanisms are used for inferring new (though implicit) knowledge. The foundation of many modern database query languages such as SQL, Cypher and XQuery are logical formalisms. In software and hardware verification, desirable properties are often specified by logic-based specification languages such as the temporal logics LTL and CTL, and the correctness of systems with respect to such specifications is verified using logical procedures.

Learning logical formalisms and in particular logical modelling is therefore inevitable for computer science students. Typical lectures introducing logic to computer science students focus on teaching how to model computer science scenarios by logical means and how to infer new knowledge from such a representation.

For example, the first weeks of the introductory logic lecture for computer scientists at the University of Dortmund cover

(a) how to model real world scenarios by propositional formulas,
(b) the transformation of formulas into an adequate normal form, and
(c) the inference of new knowledge (represented as propositional formulas) using an inference mechanism such as propositional resolution.

Afterwards a similar process is introduced for modal logic and first-order logic, thereby equipping students with the means to later learn similar logical languages from application areas such as artificial intelligence, verification, or databases by themselves.

Each of the tasks (a)–(c) is simple and can be performed quite mechanically. Yet, as most advanced logical topics require fluency in these tasks, it is essential that students practice each of them and see how they play together in solving problems.

After learning the steps (a)–(c) for propositional logic, students are expected to be able to solve problems such as the following.

Example 1.1. After carefully investigating a faulty software system, Julia has found the following dependencies between the three components of the system:

(1) If the database is faulty, then so is the back end.

(2) The back end is only faulty if both the database and the user interface are faulty.
(3) Not all three components are faulty.

Julia concludes that the database is correct. Can you verify her conclusion by modelling the situation in propositional logic and inferring Julia's conclusion using propositional resolution? □

Goal of this Work and Contribution. The goal of the Iltis project is to develop a web-based, interactive system that (1) supports teaching the modelling process (a)–(c) for propositional logic, modal logic and first-order logic, and (2) provides immediate, didactically valuable feedback and assistance when required. The system shall allow for easy inclusion of further logics, other typical tasks, and additional feedback mechanisms.

In this article we present a web-based prototypical application and its underlying framework that support the modelling process for propositional logic outlined above. Each step required to solve the problem from Example 1.1 is implemented as a task. The framework allows to specify exercises by combining such tasks in a flexible way in XML. Tasks can provide feedback using a generic mechanism, which is implemented in detail to provide feedback for a task where students have to model statements by propositional formulas. A preliminary evaluation of the framework has been performed in the winter term 2017/2018.

We acknowledge that students usually struggle rather with modal and first-order logic than with propositional logic. However, focusing on propositional logic so far has allowed us to create a robust architecture and gather experience in how to design tasks and useful feedback mechanisms. In the conclusion we sketch our vision for the future of the project.

Technically the Iltis framework is implemented in Java using the Google Web Toolkit (GWT), offering a webpage user interface based solely on HTML and Javascript. We intend to publish the source code as open source as soon as a stable version is available.

Related work. We concentrate on web-based systems as they are accessible to most students. Several existing systems cover some of the tasks we are aiming at. The *LogEX* system allows for training the transformation of propositional formulas [8, 9]. The inference of new knowledge using calculi that are close to natural inference is supported by many systems, see e.g. [2, 4, 6, 11]. Resolution, which is used in the introduction to logic in Dortmund, is to the best of our knowledge only supported by the *AELL* system [6] (which is not publicly available and only has support for the Spanish and Catalan language). In *Tarski's World* students can learn how to evaluate first-order logic in a 3D-world. A playful but prototypical approach towards topics in an introductory logic course is taken in [10].

Digital tools are also used in some interactive logic books. For example, the teaching environment [3] allows for transforming textual statements into logical formulas by a mark-and-replace technique. Many small interactive tasks can be found in the interactive book [12] as well as on the support websites of *Power of Logic* [13]. An inspiration for presenting models for modal formulas (i.e. Kripke structures) can be found in [7].

An overview over further, also older systems, can be found in [5].

In summary, some of the aspects Iltis aims at are covered by other systems. Yet, an integration of these systems seems to be impracticable or even impossible due to technological diversity.

Organization. The system is introduced from a teacher's perspective in Section 2. After presenting the architecture and components of the Iltis system in Section 3, we report on the feedback generation for writing propositional formulas in Section 4. In Section 5 we discuss first class room experiences. We conclude in Section 6.

2 THE ILTIS SYSTEM: THE TEACHER'S PERSPECTIVE

The Iltis system allows teachers to specify exercises in XML, which are then presented to students in the web (see Figures 1 and 2). Each exercise is built from one or more small tasks. As an example, the exercise described in the introduction can be built from tasks for (1) choosing suitable propositional variables, (2) formulating natural language statements as propositional formulas, (3) stating the inference goal as a satisfiability question for a formula, (4) transforming the formula into conjunctive normal form, and finally (5) applying resolution to decide satisfiability.

So far these tasks have been implemented for propositional logic. In addition to purely logical tasks, several helper tasks for questionnaires, for collecting feedback and data, and for starting new exercises from within an exercise have been implemented. While such tasks are helpful in designing exercises and tutorials, we focus on the more interesting logic related tasks in the following. Table 1 provides an overview of the currently available tasks.

A teacher can create a new exercise by specifying a sequence of tasks in an XML format. The specification of a task includes the input and output of a task, and thereby allows to connect interdependent tasks. For example, consider a modelling task, followed by a transformation task. In the modelling task, the student is asked to provide a formula ψ for a natural language statement. The statement and a solution formula φ forms the input of the modelling task and, if equivalent to the solution formula, ψ forms its output. The following transformation task in turn, receives formula ψ. See Figure 1 for an example XML specification that describes the exercise from Example 1.1; some of the resulting tasks in the web interface are illustrated in Figure 2a–2c.

Most tasks come with several methods for providing feedback. These methods can be combined by the teacher in a flexible way. For example, when a student translates a statement into a formula, the system can provide feedback on (1) whether the formula is correct, and (2) whether certain propositional variables should (not) be used, but it can also analyse the formula more deeply and point out, among others, (3a) the explanation of a logical operator that has been used in a wrong way by the student, and (3b) wrong parts of the formula. From these methods, a teacher can choose those that are adequate for the current progress of her students, e.g., extensive feedback at the beginning of a course and less extensive feedback in exercises aimed at the preparation for examinations.

Learning technique

```
<?xml version="1.0" encoding="UTF-8"?>
<Exercise name="Faulty Software System Exercise">
  <Title>Faulty Software System</Title>
  <Description> <p> After cafrefully investigating a...</p></Description>
  <Task type="PickVariables" feedbackLevels="0">
    <Title>Step 1: Choosing suitable propositional variables</Title>
    ...
    <Output>VARIABLES</Output>
  </Task>
  <!-- State formulas for the statements -->
  <Task type="CreateFormulas" feedbackLevels="0,1,2"
        assimilationGenerator="syntaxServer">
    <Input>VARIABLES</Input>
    <Title>Step 2: Modelling the statements</Title>
    <Description>
      Devise a formula for each observation! Use the propositional...
    </Description>
    <Formula>
      <Description>
        If the database is faulty then so is the back end.
      </Description>
      <Solution>D ⟶ B</Solution>
    </Formula>
    <Formula>
      <Description>
        The back end is only faulty if both the database and the user...
      </Description>
      <Solution>B ⟶ (D ∧ U)</Solution>
    </Formula>
    <Formula>
      <Description>Not all three components are faulty.</Description>
      <Solution>¬(B ∧ D ∧ U)</Solution>
    </Formula>
    <!-- Feedback generator -->
    <FeedbackGenerator>
      <Feedback type="VariableNames">
        <Variable name="U">the user interface</Variable>
        <Variable name="B">the back end</Variable>
        <Variable name="D">the database</Variable>
      </Feedback>
    </FeedbackGenerator>
    <Output>FORMULAE</Output>
  </Task>
  <!-- State formula for the conclusion -->
  <Task type="CreateFormulas"...>
    <Input>VARIABLES</Input>
    ...
    <Output>CONCLUSIONFORMULA</Output>
  </task>
  <!-- Inferring the conclusion -->
  <Task type="CompleteFormula">
    <Input>FORMULAE</Input>
    <Input>CONCLUSIONFORMULA</Input>
    <Title>Step 4: How to infer the conclusion</Title>
    <Description>
        State an equivalence from which Julia's conclusion can be inferred.
    </Description>
    <Output>COMPLETEFORMULA</Output>
  </Task>
  <Task type="transformToCnf">...</task>
    <Input>COMPLETEFORMULA</Input>
    <Title>Step 5: Transformation into conjunctive normal form</Title>
    <Description>Transform the created formula to ....</Description>
    <Output>CNF_FORMULA</Output>
  </Task>
  <Task type="Resolution">
    <Input>CNF_FORMULA</Input>
    <Title>Step 6: Propositional resolution</Title>
    <Description>Prove that Julia's conclusion is...</Description>
  </Task>
</Exercise>
```

Figure 1: A sample XML specification of the task from Exercise 1.1. Outputs of some tasks are inputs to other tasks.

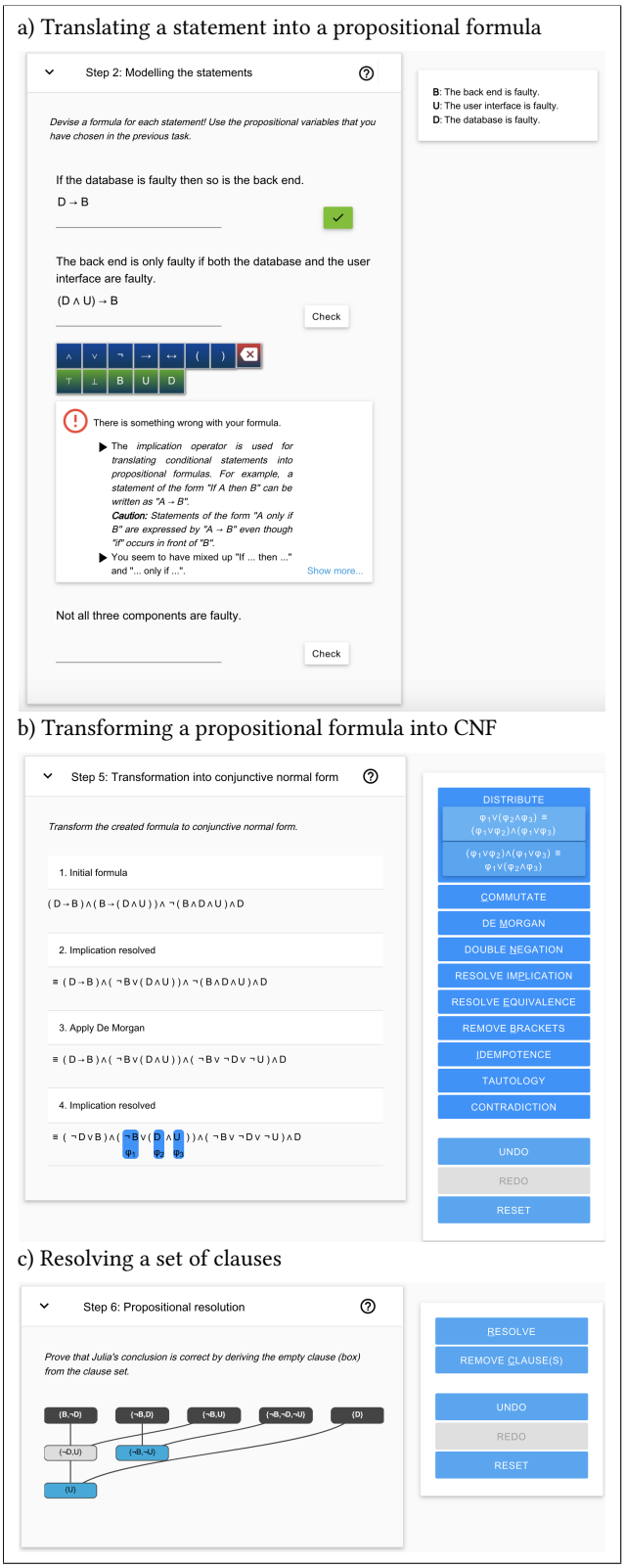

Figure 2: Some of the tasks specified in the XML file from Figure 1 as presented by the web interface.

Task	Description	Input	Output
Logical tasks			
PickVariable	Choose suitable propositional variables from a list.	—	variables $A_1, \ldots, A_m$
CreateFormula	Translate statements into formulas.	variables $A_1, \ldots, A_m$	formulas $\varphi_1, \ldots, \varphi_k$
InferenceFormula	Combine formulas $\varphi_1, \ldots, \varphi_k$ and a formula φ into a formula ψ that is unsatisfiable if and only if $\varphi_1, \ldots, \varphi_k$ imply φ.	formulas $\varphi_1, \ldots, \varphi_k$ and φ	a formula ψ
ManualTransformation	Textfield-based transformation of a formula φ into conjunctive, disjunctive or negation normal form, or into another formula.	a formula φ	the transformed formula ψ
GuiTransformation	Same as previous, but graphical user interface.	a formula φ	the transformed formula ψ
Resolution	Resolve the empty clause from the clauses of the CNF of φ.	a formula φ	—
Administrational tasks			
Questionaire task (ask a list of multiple choice questions), tasks to display messages, and a task to collect data and feedback from students.			

Table 1: List of currently implemented tasks for propositional logic.

3 THE ILTIS SYSTEM: THE DEVELOPER'S PERSPECTIVE

One of the goals of the Iltis framework is to allow for a modular inclusion of new types of tasks. A typical developer designing a new task shall be able to focus on the functionality of the task at hand. To this end, the mechanisms to read exercises from an XML file, to create exercise objects from such a description, and the execution of the tasks stored in such an object is implemented in a generic, task-independent way.

We focus on the internals of tasks in the following. Afterwards we summarize some further aspects that are interesting from a developer's perspective.

Internals of Tasks. All tasks follow the model-view-controller pattern. In order to implement a new task, a developer has to specify a task model that stores the current state of the task, and a task view that represents the task in the graphical user interface; a task controller is generated automatically. In addition the developer has to specify task specific actions and feedbacks.

Upon user input, the task view generates an action that is sent to the controller. The task controller executes the action, if it is applicable, and then returns feedback to the task view which is then displayed. For example, when a user does a resolution step in the resolution task, a resolution step action is generated. The execution of this action, triggered by the controller, includes the verification that two clauses are selected, that they can be resolved, and – if so – the modification of the resolution graph stored in the resolution task model. The task model then triggers an update of the view.

The feedback is created by feedback generators. A specific task can have several feedback generators, that are arranged in a cascading fashion. For example, the task for translating statements into formulas can have two feedback generators, one for checking whether only available variables are used and one for determining where a given formula is wrong (see Section 4 for more details). Developers provide implementations of the feedback generators, which can then arranged dynamically by teachers in exercise XMLs.

Other Features. The framework supports anonymous data collection. All tasks come with specific loggers that have been used to collect the data for the evaluation presented in Section 5. Internationalization is supported and general texts are currently provided in German and English.

4 CASE STUDY: FEEDBACK FOR STATING PROPOSITIONAL FORMULAS

Feedback is essential for the learning process. Even a simple yes/no-answer gives students a feeling of their skills and allows them to seek help if necessary. All tasks implemented in Iltis provide this rudimentary feedback. Yet, the feedback mechanism allows for more nuanced feedback.

As an example of how to use the feedback mechanism, we describe its instantiation for the task where a student shall translate a statement to a propositional formula. In this task, the propositional variables to be used as well as their intended meaning is provided by the task statement. The feedback mechanism receives a correct propositional formula φ for the natural language statement, henceforth called *solution formula*, and the formula ψ provided by a student, called the *student formula*.

Rudimentary feedback can be obtained by checking whether φ and ψ are logically equivalent, and, if they are not, providing an assignment that distinguishes the student formula from the solution formula. Also checks such as verifying that only available and necessary variables are used can be performed easily.

A deeper analysis of typical errors is necessary for more meaningful feedback. In a preliminary study, we collected typical errors made by students in a final written examination. The collected errors include, among others, the use of wrong logical operators, interchanging antecedent and consequent of implications (in particular in "only if" statements as the one in statement (2) of Example 1.1) and not modelling parts of the natural language statement. Furthermore, all typical errors can be found in diverse combinations.

We used these error types to implement a more advanced feedback mechanism. The idea of the mechanism is to try to transform the student formula ψ into a formula φ' that is equivalent to the solution formula φ by reverting student mistakes. To this end, we extracted general declarative reversion rules from the list of typical errors. Each such rule searches for a pattern in ψ that might result from a mistake, and transforms ψ locally around this pattern.

Example 4.1. As an example, the error where a student interchanged antecedent and consequent is specified by the reversion rule $\rho : \$X \to \$Y \rightsquigarrow \$Y \to \$X$. Here, the pattern $\$X \to \$Y$ searches for a subformula which is an implication and assigns its antecedent and consequent to $\$X$ and $\$Y$, respectively. The right

hand side of the rule ρ specifies how the error can be reverted, in this case by swapping $\$X$ and $\$Y$.

When a student erroneously wrote $\psi = (D \wedge U) \rightarrow \neg B$ instead of $\varphi = \neg B \rightarrow (D \wedge U)$, the rule ρ is applied subsequently to all subformulas that match the left hand side of ρ. As the interchanging of antecedent and consequent is the only mistake, one of those applications yields a formula equivalent to the solution formula.

From the type of the reversion rule as well as from its match in ψ, more meaningful feedback can be constructed (see Figure 2a). □

In general, a student can make several mistakes $m_1, \ldots, m_k$ in one formula. In this case applying the corresponding reversion rules $\rho_1, \ldots, \rho_k$ yields a formula equivalent to the solution formula.

Of course, the computational resources necessary to try to find k such rule applications to the (possibly various) matching locations grow exponentially in k. Fortunately, typical formulas are very small, and therefore a rule in most cases matches at very few locations. Further, if a student made more than two mistakes in a small formula, the information that the formula is wrong is usually more useful than more precise feedback. We determined experimentally that restricting the length of reversion sequences to two yields sufficient performance, while still providing good feedback.

For the feedback in Figure 2a the feedback generator was parametrised such that feedback for wrong formulas is generated and displayed in the following order:

Pode ter interesse para o projeto

(1) "The formula is wrong."
(2) If the formula is syntactically wrong: "Please enter a propositional formula".
(3) If the formula is semantically wrong:
   (a) If a sequence of reversion rules that yields a correct formula is found:
      (i) A general description of a probable misconception is displayed. For example, if the antecedent and consequent are interchanged, then an explanation of how "If…then…" and "…only if…" statements are modelled by implications is displayed.
      (ii) A precise description of the error is displayed. For example: "You seem to have interchanged 'If…then…' and '…only if…' ".
      (iii) The wrong part of the formula is highlighted.
   (b) If no sequence of reversion rules could be found: An example assignment that distinguishes the student formula from the solution formula is displayed.

The feedback thus provided usually gives a good idea of how to improve a formula.

Naturally, the feedback is not the best possible. A more advanced feedback mechanism could take advantage of other available information. By annotating subformulas of the solution formula by information about the natural language formulations, the feedback can be made more precise. In the long run, we plan to model the state of learning for each student, and thereby allow for individualized feedback generation. The current implementation is the foundation for both of these extensions.

5 FIRST CLASSROOM EXPERIENCES

In the winter term 2017/2018 the Iltis system has been used in the introductory course "Logic for Computer Scientists" at the Technical University Dortmund, aimed at second year students.

The course has fourteen weeks which are almost evenly distributed between propositional, modal and first-order logic. In addition to a two hour lecture per week, the course includes biweekly two hour exercise groups in which students solve problems and a weekly two-hour tutorial session for students with a need for assistance. Solutions to exercises have to be handed in by students every two weeks for being admitted for final written examinations.

The Iltis system was used in a voluntary web-tutorial during has the part on propositional logic. We also evaluated the feedback for the modelling task described in Section 4 systematically.

Accompanying Web-Tutorial. An interactive tutorial covering (i) the basic connectives and their use for modelling real world scenarios, (ii) propositional equivalences and normal forms, as well as (iii) satisfiability in propositional logic has been published in parallel to the lectures. Each part of the tutorial includes an introduction to the topic, and small hands-on exercises that are checked immediately by the system. Also exercises of the full modelling process have been included into the tutorial. At the time of writing this article, the tutorial has been accessed ¿700

Evaluation of the Modelling Task. The feedback provided for the modelling task, as described in the previous section, was evaluated experimentally. In their first exercise session on propositional modelling, the students were partitioned into four groups, each of which had to solve the same three web-based exercises. Each exercise consisted of four statements to be modelled by propositional formulas. All three exercises contained statements of the same type, e.g., all of them contained an "only…if" and an "either…or" statement.

In the first and last exercise the knowledge of all students was tested, i.e. they modelled the statements without receiving help or feedback. In the second exercise, help and feedback provided by the system differed with the group of students. The control group (CG, $n_{\text{CG}} = 57$) received no feedback; the first experimental group (EG1, $n_{\text{EG1}} = 43$) received feedback provided by the system (as described in Section 4); the second experimental group (EG2, $n_{\text{EG2}} = 98$) was presented with a short repetition of how to model statements using propositional logic; and the third experimental group (EG3, $n_{\text{EG3}} = 51$) received the same repetition as well as feedback. To study the impact of help and feedback, the system logged all errors made by the students. From this data we extracted the error rate of each group for each statement type and each exercise. Before outlining the results we discuss some shortcomings of the set-up, and remark that the results should be taken with caution. First, the natural language statements of the three exercises are necessarily different. Therefore error rates even for statements of the same type may differ significantly over the three exercises. Further, for simplicity of the set-up, all students from a given exercise group of the logic course were assigned to the same group in the experiment. We cannot rule out that there are dependencies between students of the same exercise groups (e.g., students with mathematics as minor could be clustered in one of the groups etc.). In addition, the level of detail of feedback is not taken into account by the experiment;

Table 2: Results of the evaluation for the translation of natural language statements into propositional formulas. See Section 5 for the experimental set-up. The error rate and the rate of the most frequent error are the percentage of students of the group that made a mistake when translating the statement and the percentage of students that made the most common error, respectively.

Group	CG			EG1			EG2			EG3		
Exercise	1	2	3	1	2	3	1	2	3	1	2	3
Only-if statement												
error rate	0.44	0.49	0.32	0.53	0.47	0.23	0.43	0.49	0.44	0.45	0.49	0.37
most frequent error	0.19	0.30	0.05	0.33	0.42	0.02	0.28	0.41	0.17	0.33	0.41	0.08
Either-or statement												
error rate	0.47	0.51	0.29	0.58	0.42	0.07	0.47	0.38	0.27	0.50	0.44	0.17
most frequent error	0.27	0.29	0.18	0.26	0.35	0.05	0.15	0.20	0.12	0.17	0.25	0.06

it would be interesting to see how detailed feedback compares to only providing students with whether their answer was correct or wrong. We plan to repeat the study in the future and to account for these problems.

For the discussion of the results, we focus on the data for the "only…if" and "either…or" statements. For each of these statements, Table 2 shows the error rate of each group with respect to each of the exercises. In addition, the error rate of the most common error is depicted. For both the "only…if" and "either…or" statement the most common errors (swapping antecedent and consequent/using "or" instead of "either…or") were the same over all groups and exercises. For these two errors, the system provides feedback as explained in detail in Section 4.

After receiving feedback in the second exercise, EG1 and EG3 perform much better in the third exercise. For example, the error rate for the "only…if" and "either…or" statements for EG1 drops from 0.47 and 0.42 to 0.23 and 0.07, respectively. The error rate of the most frequent error even drops from 0.42 and 0.35 to 0.02 and 0.05. As discussed above, this could be due to the different phrasings of the statements. However, the drop of the error rates of these two groups is larger than of the other two groups CG and EG2.

The results suggest that students learn from the feedback provided by the system. In a questionnaire after the evaluation, 74.6 percent of the students evaluated the system as good or very good.

6 CONCLUSION AND VISION

The current state of the Iltis project is a first step towards supporting students to learn logics in a modern, interactive way – at home, in trains or at the beach. While the modelling process for propositional logic is implemented, a lot of challenges remain.

Extension of the System. From a students perspective it is important to include more advanced logical formalisms as soon as possible. Currently we work on the inclusion of modal logic and predicate logic. As a foundation, an abstract term framework has been implemented that allows for including new logical mechanisms in a simple and general way.

For modal logics and first-order logic, the ability to illustrate models is essential. For example, it is easier for students to see why a modal formula is wrong, when they see for which structure it deviates from the solution formula. Currently we work on including a framework for illustrating and manipulating graph-like structures.

In the long run we also plan to include other topics from computer science. Many parts of the Iltis framework are likely to be reusable. For example, regular expressions can be expressed by the abstract term framework, and the declarative feedback mechanism developed for propositional logic is likely to be easily adaptable. Also typical tasks for studying and manipulating finite state automata, as well as feedback mechanisms for those tasks, can likely be obtained by adapting tasks performed on Kripke structures.

The inclusion and development of techniques to increase the motivation of students is planned as well.

Feedback Generation. Feedback generation for more expressive logics than propositional logics is, of course, not easy. For modal logic the declarative mechanism designed for propositional logic can likely be adapted. In addition, errors can be illustrated by counter examples as explained above.

Verifying the correctness of a first-order formula, proposed as solution by a student, cannot be performed algorithmically in general (since it is undecidable). We plan to investigate workarounds. One such workaround is to treat first-order formulas as selection mechanism (i.e. as queries). Imagine a graph in which several nodes are marked, and a student is asked to write a formula with one free variable that selects the marked nodes. Then an error in a student formula can be easily illustrated by highlighting the set of nodes selected by the formula (which differs from the marked nodes). Another workaround is to identify a decidable fragment of first-order logic that includes typical formulas provided by students (i.e short formulas with few quantifications and quantifier alternations).

Didactical Research. The Iltis project offers the opportunity to study how students learn logic. For example, anonymised data can be collected about the types of errors made when constructing formulas, and the amount of time spent on each formula. The analysis of such data will likely lead to a better understanding of typical misconceptions of students.

Acknowledgements. We thank Nils Vortmeier for the collection of an initial data set of student mistakes. We are grateful to Johannes Fischer, Marko Schmellenkamp, and Thomas Schwentick for valuable feedback on a draft of this article.

REFERENCES

[1] Blackburn, P., van Ditmarsch, H., Manzano, M., and Soler-Toscano, F., editors (2011). *Tools for Teaching Logic - Third International Congress, TICTTL 2011, Salamanca, Spain, June 1-4, 2011. Proceedings*, volume 6680 of *Lecture Notes in Computer Science*. Springer.

[2] Ehle, A., Hundeshagen, N., and Lange, M. (2015). The sequent calculus trainer - helping students to correctly construct proofs. *CoRR*, abs/1507.03666.

[3] Fricke, M. (2012). Software and tutorials for instruction in symbolic logic. http://softoption.us. Accessed: 2017-12-28.

[4] Gasquet, O., Schwarzentruber, F., and Strecker, M. (2011). Panda: A proof assistant in natural deduction for all. A gentzen style proof assistant for undergraduate students. In [1], pages 85–92.

[5] Huertas, A. (2011). Ten years of computer-based tutors for teaching logic 2000-2010: Lessons learned. In [1], pages 131–140.

[6] Huertas, A., Humet, J. M., López, L., and Mor, E. (2011). The SELL project: A learning tool for e-learning logic. In [1], pages 123–130.

[7] Kirsling, R. (2015). Software and tutorials for instruction in symbolic logic. http://rkirsling.github.io/modallogic/. Accessed: 2017-12-28.

[8] Lodder, J. and Heeren, B. (2011). A teaching tool for proving equivalences between logical formulae. In [1], pages 154–161.
[9] Lodder, J., Heeren, B., and Jeuring, J. (2015). A pilot study of the use of LogEx, lessons learned. *CoRR*, abs/1507.03671.
[10] Schäfer, A., Holz, J., Leonhardt, T., Schroeder, U., Brauner, P., and Ziefle, M. (2013). From boring to scoring - a collaborative serious game for learning and practicing mathematical logic for computer science education. *Computer Science Education*, 23(2):87–111.
[11] Sieg, W. (2007). The apros project: Strategic thinking & computational logic. *Logic Journal of the IGPL*, 15(4):359–368.
[12] Velleman, J. D. (2010). blogic: A web logic textbook. www.nyu.edu/classes/velleman/blogic. Accessed: 2017-12-28.
[13] Wasserman, R., Howard-Snyder, D., and Howard-Snyder, F. (2012). *The Power of Logic*. McGraw-Hill Education.