A domain reasoner for propositional logic

Josje Lodder

Bastiaan Heeren

Johan Jeuring

Technical Report UU-CS-2015-021
December 2015

Department of Information and Computing Sciences
Utrecht University, Utrecht, The Netherlands
www.cs.uu.nl

ISSN: 0924-3275

Department of Information and Computing Sciences
Utrecht University
P.O. Box 80.089
3508 TB Utrecht
The Netherlands

A domain reasoner for propositional logic

Josje Lodder, Bastiaan Heeren, Johan Jeuring
Open University of the Netherlands, Heerlen, The Netherlands
josje.lodder@ou.nl, bastiaan.heeren@ou.nl, J.T.Jeuring@uu.nl

Abstract. Students learn propositional logic in programs such as mathematics, philosophy, computer science, law, etc. An important topic in courses in propositional logic is rewriting propositional formulae with standard equivalences, and the application of this technique in exercises on rewriting a formula in normal form, proving the equivalence between two formulae or proving that a formula is a consequence of a set of formulae. Existing learning environments for propositional logic offer limited feedback and feed forward. This paper analyses what kind of feedback is offered by the various learning environments for rewriting propositional logic formulae, and discusses how we can provide these kinds of feedback in a learning environment. To give feedback and feed forward, we define solution strategies for several classes of exercises. We offer an extensive description of the knowledge necessary to support solving this kind of propositional logic exercises in a learning environment. This description serves as an illustration of how to develop the artificial intelligence for a particular domain.

Keywords. propositional logic, normal forms, learning environment, domain reasoner, feedback, feed forward, intelligent tutoring

INTRODUCTION

Students learn propositional logic in programs such as mathematics, philosophy, computer science, law, etc. For example, the ACM IEEE Computer Science Curricula 2013[1] mentions the topics *propositional logic* and *normal forms*, and the learning outcomes *computing normal forms* and *using the rules of inference to construct proofs in propositional logic* in its Core.

Two essential factors underpinning successful learning are ‘learning by doing’ and ‘learning through feedback’ (Race, 2005). Students learning propositional logic practice by solving exercises about rewriting propositional formulae into DNF or CNF, proving the equivalence of two formulae or deriving that a formula is a consequence of a given set of formulae, using standard equivalences and consequences. Most textbooks for propositional logic (Benthem et al., 2003; Hurley, 2008; Vrie and Lodder et al, 2009; Burris, 1998; Kelly, 1997) contain these kinds of exercises. Such an exercise is typically solved in multiple steps, and may be solved in various correct ways. Textbooks sometimes describe how such exercises are solved, and give examples of good solutions. Because there often are many good solutions, it is

[1]`http://www.acm.org/education/CS2013-final-report.pdf`

infeasible to give all of them in a textbook, or provide them online. A key learning outcome in problem-solving domains such as logic is the development of flexible and effective procedural knowledge, where a learner applies multiple strategies adaptively to a range of situations (Star and Rittle-Johnson, 2008; Waalkens et al., 2013).

How do students receive feedback when working on exercises in propositional logic? Many universities organise exercise classes or office hours to help students with their work. However, it is not always possible to have a human tutor available. A student may be working at home, studying at a distance teaching university, or the teaching budget may be too limited to appoint enough tutors. In these cases, access to an intelligent tutoring system (ITS) (VanLehn, 2006) might be of help. An ITS provides several services to students and teachers. Important services of ITSs are selecting tasks to be solved by the student, recording the level and progress of the student, diagnosing student actions, and giving feedback and feed forward to students. An ITS that follows the steps of a student when solving a task can be almost as effective as a human tutor (VanLehn, 2011).

Traditionally, the architecture of an intelligent tutoring system (ITS) is described by means of four components (Nwana, 1990): an expert knowledge module, a student model module, a tutoring module, and a user interface module. In this paper we focus on the services provided by the expert knowledge module and the tutoring module, and do not address the student module. Most of the systems we discuss in the related work section also do not have a student module. We use the term learning environment (LE) to denote a system that does not necessarily contain a student module. The expert knowledge module is the component responsible for reasoning about the problem. Following Goguadze (2011), we will use the term *domain reasoner* for this component.

Feedback is an important aspect of an LE. Usually an LE offers various kinds of feedback: a diagnosis of a student step, a hint for the next step to take, in various level of detail, or a completely worked-out solution. A diagnosis of a student step may analyse the syntax of the expression entered by the student, whether or not the step brings a student closer to a solution, or whether or not the step follows a preferred solution strategy, etc. What kind of feedback do LEs for propositional logic give? There are quite a few tutoring systems for logic available (Huertas, 2011). In this paper we look at systems that deal with standard equivalences and consequences, in which a student has to learn to rewrite formulae, either to a normal form or to prove an equivalence or a consequence. The various tutoring systems or learning environments for this part of propositional logic offer all kinds of feedback, but no system combines all desired kinds of feedback in a single learning environment. For example, hardly any system checks that a student step follows a preferred solution strategy.

Example interactions in an LE for propositional logic

This subsection gives some examples of interactions of a student with a logic tutor with advanced feedback facilities.

Suppose a student has to solve the exercise of rewriting the formula

$$\neg((p \vee q) \wedge (p \rightarrow q)) \vee \neg q$$

into DNF. If the student knows how to solve this exercise, she enters all intermediate steps, which are recognised to be correct by the LE. The LE informs the student that the constructed solution is correct.

Another (hypothetical) student goes through the following steps:

$$\neg(p \vee q) \wedge \neg(p \rightarrow q)) \vee \neg q$$

If a student submits this expression the LE reports that a parenthesis is missing in this formula. After correction the formula becomes:

$$(\neg(p \vee q) \wedge \neg(p \rightarrow q)) \vee \neg q$$

The LE reports that this formula is equivalent to the previous formula, but it cannot determine which rule has been applied. The LE has an external equivalence procedure that is used to test semantic equivalence. Besides the equivalence procedure, it has knowledge about which rules should be applied, in which order, to obtain a formula in DNF. There are at least two possible reasons why the LE fails to recognise the rule applied: either the student performs multiple steps, or applies an incorrect step. In this case the student has very likely made a mistake in applying the DeMorgan rule. Correcting this, the student submits:

$$\neg(p \vee q) \vee \neg(p \rightarrow q) \vee \neg q$$

Now the LE recognises the rule applied (DeMorgan), and adds the formula to the derivation. Note that the parentheses around the first disjunction are omitted. The LE allows implicit applications of associativity. The student now eliminates the implication:

$$\neg(p \vee q) \vee \neg(\neg p \vee q) \vee \neg q$$

The LE recognises the application of the implication definition rule, and appends this step to the derivation.

Suppose the student does not know how to proceed here, and asks for a hint. The LE responds with: Use DeMorgan. The student does not know where to apply the DeMorgan rule, and asks for more information. Now the LE responds with: Use DeMorgan to rewrite the formula into $\neg(p \vee q) \vee (\neg\neg p \wedge \neg q) \vee \neg q$. The student asks the LE to perform this step, and from there on continues with:

$$(\neg p \vee \neg q) \vee (\neg\neg p \wedge \neg q) \vee \neg q$$

The LE reports that this step is not correct, and mentions that when applying DeMorgan's rule, a disjunction is transformed into a conjunction. Since the new formula is not equivalent to the old one, the LE searches for common mistakes (also called buggy rules), and finds a common mistake in the application of the DeMorgan rule. Note that in the second step of this hypothetical interactive session, the student made the same mistake, but since the formulae were accidentally semantically the same, the LE did not search for common mistakes there. The student corrects the mistake:

$$(\neg p \wedge \neg q) \vee (\neg\neg p \wedge \neg q) \vee \neg q$$

and the LE appends this step to the derivation, together with the name of the rule applied (DeMorgan). The next step of the student:

$$(\neg p \wedge \neg q) \vee \neg q$$

is also appended to the derivation, together with the name of the rule applied (Absorption). At this point, the student may recognise that the formula is in DNF, and ask the LE to check whether or not the exercise is completed. She also has the possibility to further simplify the formula into $\neg q$.

As a second example we look at an exercise in which a student has to prove that two formulae are equivalent:

$$(\neg q \wedge p) \rightarrow p \Leftrightarrow (\neg q \leftrightarrow q) \rightarrow p$$

The LE places the right-hand side formula below the left-hand side formula, and the student has to fill in the steps between. It is possible to enter steps top-down or bottom-up, or to mix the two directions. For this particular class of exercises, a student not only has to enter the individual proof steps, but also the name of the rule that is applied in each step.

The student chooses to enter a bottom-up step and to rewrite $(\neg q \leftrightarrow q) \rightarrow p$ into:

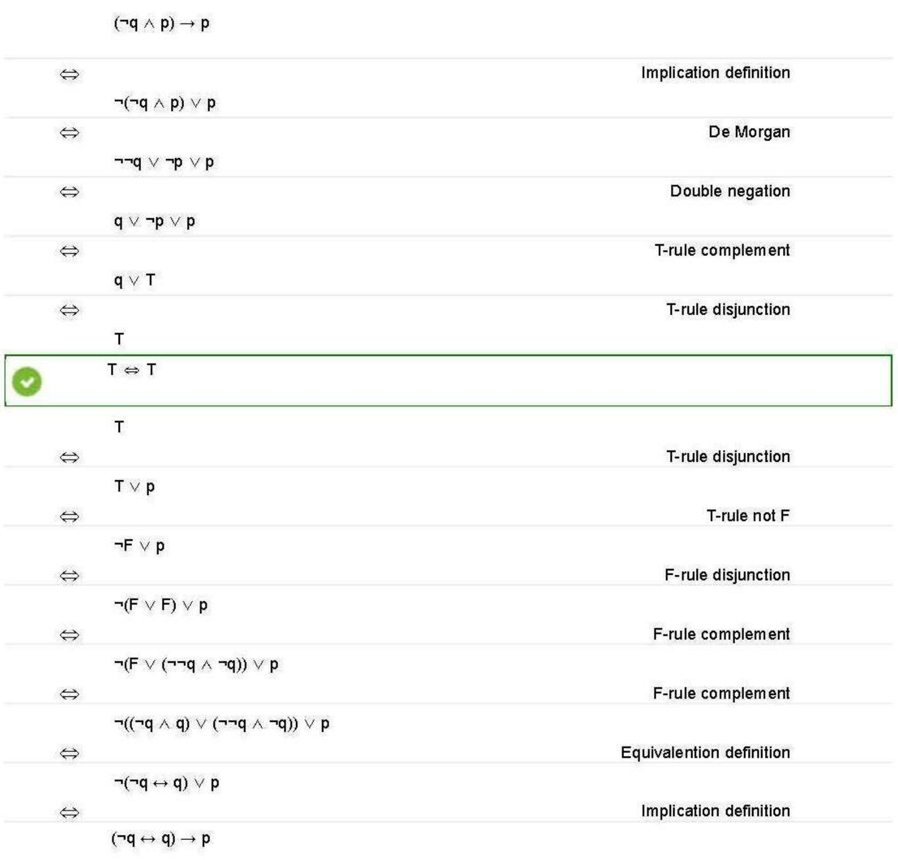
$$(\neg\neg q \vee q) \rightarrow p \qquad \{\text{Equivalence definition}\}$$

Since the student enters the name of a rule, we can now more easily than in the previous example conclude that the student has made a mistake in applying the equivalence definition. The LE searches for common mistakes, and reports that implication definition has been used instead of equivalence definition. The student corrects her mistake, and the new formula is added to the proof. If she does not know how to proceed, and asks for a hint, the LE will suggest to add a top-down step first, and a next hint will advise her to use implication definition. The student can continue to finish the exercise, but she can also ask the LE to provide a complete solution, see Figure 1.

Thus far, the student received feedback on the level of single steps. This implies that she could have finished the exercise correctly, but in a rather inefficient way. To provide partial feedback on the solution strategy used, our LE calculates at each step the number of steps remaining. The student can use this to see if her step is correct, but not very efficient. In this case she can undo the step, and choose a more efficient step. On the other hand, if she finds a clever way to solve the exercise, she will see that the number of remaining steps decreases by more than one.

This paper

This paper analyses what kind of feedback is offered by the various learning environments for rewriting propositional logic formulae, and discusses how we can provide these kinds of feedback in a learning environment. We discuss how we can diagnose student interactions, and give feedback and feed forward in an LE for propositional logic. To give feedback and feed forward, we define solution strategies for several classes of exercises. This paper gives an extensive description of the knowledge necessary to support solving this kind of propositional logic exercises in an LE, and as such, serves as an illustration of how to develop the artificial intelligence for a particular domain. Some interesting aspects of solving exercises in propositional logic are:



$(\neg q \wedge p) \rightarrow p$

$\Leftrightarrow$ Implication definition

$\neg(\neg q \wedge p) \vee p$

$\Leftrightarrow$ De Morgan

$\neg\neg q \vee \neg p \vee p$

$\Leftrightarrow$ Double negation

$q \vee \neg p \vee p$

$\Leftrightarrow$ T-rule complement

$q \vee T$

$\Leftrightarrow$ T-rule disjunction

T

✓ $T \Leftrightarrow T$

T

$\Leftrightarrow$ T-rule disjunction

$T \vee p$

$\Leftrightarrow$ T-rule not F

$\neg F \vee p$

$\Leftrightarrow$ F-rule disjunction

$\neg(F \vee F) \vee p$

$\Leftrightarrow$ F-rule complement

$\neg(F \vee (\neg\neg q \wedge \neg q)) \vee p$

$\Leftrightarrow$ F-rule complement

$\neg((\neg q \wedge q) \vee (\neg\neg q \wedge \neg q)) \vee p$

$\Leftrightarrow$ Equivalention definition

$\neg(\neg q \leftrightarrow q) \vee p$

$\Leftrightarrow$ Implication definition

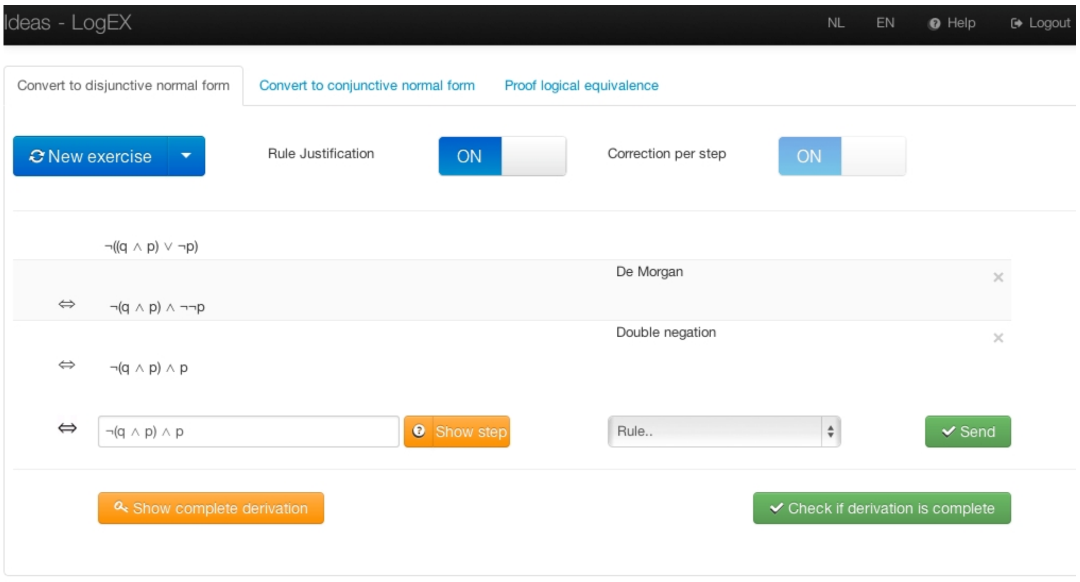
$(\neg q \leftrightarrow q) \rightarrow p$

Fig.1. Screenshot of a worked out proof

- Some exercises, such as proving equivalences, can be solved by working from left to right (or top to bottom), or the other way around, or a mix of the two directions. How do we support solving exercises in which a student can make progress at different positions?

- Proving the equivalence of two formulae, or deriving that a formula is a consequence of a given set of formulae, requires heuristics. These heuristics support effective reasoning and the flexible application of solution strategies in these proofs. It is not possible to always find the 'best' proof using heuristics, but we can get quite far for beginners' exercises. How do we formulate heuristics in our solution strategies for solving these kind of exercises?

- Reuse and adaptivity play an important role in this domain: different teachers or LEs for propositional logic allow different rules, rewriting to normal form is reused, in combination with heuristics, in proving equivalences, etc. How can we support reusing and adapting solution strategies for


Ideas - LogEX
NL
EN
Help
Logout
Convert to disjunctive normal form
Convert to conjunctive normal form
Proof logical equivalence
New exercise
Show step
Rule..
Send
Show complete derivation
Check if derivation is complete

Rule Justification ON

Correction per step ON

¬((q ∧ p) ∨ ¬p)

⇔ ¬(q ∧ p) ∧ ¬¬p   De Morgan

⇔ ¬(q ∧ p) ∧ p   Double negation

⇔ ¬(q ∧ p) ∧ p

Fig.2. Screenshot of our learning environment for logic

logic exercises?

Besides describing the knowledge necessary to support solving propositional logic exercises in an LE, the contributions of this paper are solutions to the above aspects of solving propositional logic exercises.

The focus of this paper is on the components necessary for providing feedback and feed forward in an LE for propositional logic. However, we have also developed an LE on top of these components (Lodder et al., 2006): see Figure 2 for a screenshot.[2] We briefly describe some experiments we performed with our LE.

The paper is organised as follows. The following section gives an overview of existing LEs for logic together with their characteristics. We identify a number of aspects that have not been solved satisfactorily in existing tools for teaching propositional logic, and describe our approach to tutoring propositional logic in the next section. We then show how to use this approach to support solving logic exercises for rewriting logic expressions to disjunctive or conjunctive normal form, for proving the equivalence of two logical formulae, and for proving that a logical formula is a consequence of a set of logical formulae. We conclude with describing the results of some experiments we performed with an LE built on top of our work.

[2]`http://ideas.cs.uu.nl/logex/`

EXERCISES AND FEEDBACK IN LOGIC TOOLS

This section gives an overview of possible features of a tool for teaching logic, and describes several existing tools. We start with a brief description of some important characteristics of tutoring systems that we will use in our comparison, and then compare existing tools based on these characteristics. Figure 3 offers an overview of these tools together with their characteristics.

Characteristics of tutoring systems

This section introduces a number of characteristics of tutoring systems, which we will use for the comparison of existing LEs for logic in the following subsection. This is not a complete description of the characteristics of LEs, but large enough to cover the most important components, such as the inner and outer loop of tutoring systems (VanLehn, 2006), and to compare existing tools. The outer loop of an ITS presents different tasks to a student, in some order, depending on a student model, or by letting a student select a task. The inner loop of an ITS consists of the interactions between a student and a system when a student is solving a particular task. Important aspects of the inner loop are the size of the steps a student can take, the analyses performed by the system, and the feedback provided by the system. For the interactions in the inner loop, some aspects are specific for LEs for logic.

Tasks. The starting point of any LE is the tasks it offers. The kind of tasks we consider in this paper are calculating normal forms (NF; in the text we introduce the abbreviations used in the overview in Figure 3), proving an equivalence (EQ), and deriving a consequence (CO).

An LE may contain a fixed set of exercises (FI), but it may also randomly generate exercises (RA). Some LEs offer the possibility to enter user-defined exercises (US).

Exercises can be grouped in difficulty levels (YES if present, NO if not), and if this is the case a student can pick a particular difficulty level, or the LE can offer an exercise corresponding to the level of the student. This latter feature is only possible if the LE has some kind of student model.

Interactions in the inner loop. In the inner loop of an LE, a student works on a particular task. The way in which a student can solve a task in an LE varies. Sometimes a student can only provide a final answer, but in other cases a student must perform many small steps to arrive at a solution. In most LEs for rewriting logical formulae a student can submit intermediate steps. Some systems allow a student to rewrite a formula without providing the name of a rewrite rule (FO), in other systems she chooses a rule and the system rewrites the formula using that rule (RU). Some systems offer a choice between either entering a formula, or providing a rulename (RoF). Another class of systems demands that she provides both the name of the rule to apply, together with the result of rewriting with that rule (RaF).

The interactions in the inner loop are facilitated by the user-interface. A user interface for an LE for logic needs to satisfy all kinds of requirements; too many to list in this paper. For our comparison, we only look at offering a student the possibility to work in two directions when constructing a proof (2D).

Feedback. How does an LE give feedback on a step of a student? To distinguish the various types of feedback, we give a list of possible mistakes. Examples of these mistakes are shown in the example session in the introduction.

- A syntactical mistake.
- A mistake in applying a rule. We distinguish two ways to solve an exercise in an LE depending on whether or not a student has to select the rule she wants to apply. If she indicates the rule she wants to apply, she can make the following mistakes:
  - perform an incorrect step by applying the rule incorrectly.
  - perform a correct step that does not correspond to the indicated rule.

  If a student does not select the rule she wants to apply, the categories of possible mistakes are somewhat different:
  - rewrite a formula into a semantically equivalent formula, but the LE has no rule that results in this formula. This might be caused by the student applying two or more rules in a single step, but also by applying an erroneous rule, which accidentally leads to an equivalent formula.
  - rewrite a formula into a semantically different formula.
- A strategic mistake. A student may submit a syntactically and semantically correct formula, but this step does not bring her closer to a solution. We call this strategic mistakes.

Thus we distinguish three categories of mistakes: syntactic errors, errors in applying a rule, and strategic errors. Narciss (2008) characterises classes of feedback depending on how information is presented to a student. After determining whether or not a student has made an error, we can provide the following kinds of feedback: Knowledge of result/response (KR, correct or incorrect), knowledge of the correct results (KCR, description of the correct response), knowledge about mistakes (KM, location of mistakes and explanations about the errors), and knowledge about how to proceed (KH). Although Narciss (and others) call this last category also feedback, others use the term feed forward (Hattie and Timperley, 2007), which we also will use in this paper.

Feed forward. To help a student with making progress when solving a task, LEs use feed forward: they may give a hint about which next step to take (HI, in various levels of detail), they may give the next step explicitly (NE), or they may give a general description of the components that can be used to solve an exercise (GE). It is important to offer help in any situation, also when a student does not follow the preferred path towards a solution. If steps can be taken both bottom-up and top-down, is feed forward also given in both directions (FF2), or just in one of the two directions (FF1)?

Solutions. Another way to help students is by offering some worked-out examples (WO), or by offering solutions to all exercises available in the tool (SOL).

Adaptability. A final aspect we consider concerns flexibility and adaptability. Can a teacher or a student change the set of rules or connectives (YES, NO)?

A comparison of tools for teaching logic

In the rest of this section we describe some LEs for logic using the characteristics from the previous subsection. We build upon the overview of tools for teaching logic presented by Huertas (2011) in 2011. Some of the tools described by Huertas no longer exist, and other, new tools have been developed. We do not give a complete overview of the tools that currently exist, but restrict ourselves to tools that support one or more of the exercise types of our tool: rewriting a formula in DNF or CNF, proving an equivalence using standard equivalences as rewrite rules, and deriving a consequence using standard equivalences and one or more inference rules. Quite a few tools for logic support learning natural deduction, some of which have extensive feed forward and feedback facilities, but they are out of scope for our comparison. We did not find tools for practicing the derivation of consequences using rewrite rules combined with only one or two inference rules. However, some tools combine natural deduction with rewriting using standard equivalences, and we therefore include some of these tools in our comparison. We summarise our findings in Figure 3, and draw conclusions at the end of the section.

Rewriting a formula in DNF or CNF

Gateway to Logic. Gateway to Logic[3] consists of several modules, including a proof checker for natural deduction (without rewriting rules), and a tool that transforms a formula into DNF or CNF. A user can enter a formula which is transformed by the system into the normal form chosen. The tool reports any syntactic error in the formula entered. Since Gateway to Logic performs the transformation, this is the only kind of feedback that the user receives from the tool. The transformation can be shown stepwise, so the tool can be used to provide examples, but for this type of exercise it is impossible to submit and check a solution. The tool only provides feed forward, either by showing the application of a rule to obtain a next step, or by rewriting the formula into normal form in a single step without showing the intermediate steps.

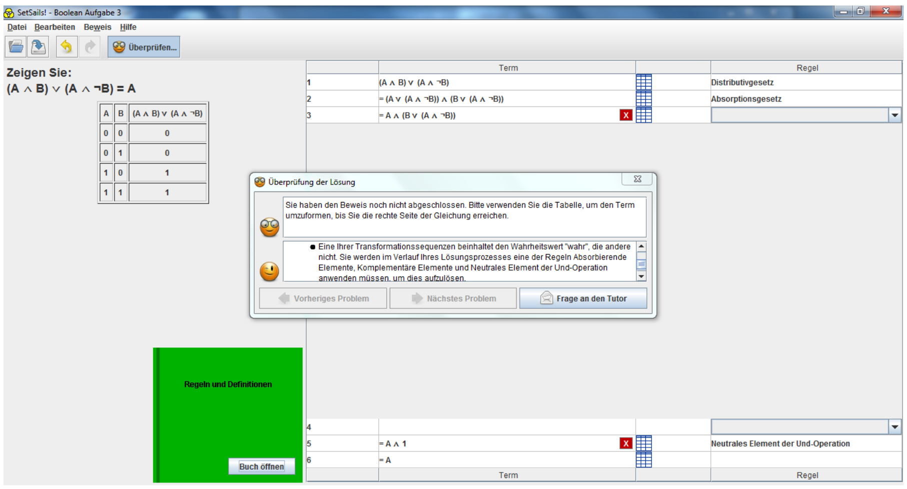
Organon. Organon[4] (Dostálová and Lang, 2011, 2007) offers the possibility to practice rewriting propositional formulae into DNF or CNF. It automatically generates exercises, based on a set of schemes that are instantiated. A student cannot indicate a rule she wants to apply when taking a step. If a rewritten formula is semantically equivalent Organon accepts it, even if the student probably made a mistake, as illustrated in the example session in the introduction. When a student enters a syntactically erroneous or non-equivalent formula, Organon gives a KR error message. In training mode, a student can ask for a next step. The steps performed by Organon are at a rather high level: it removes several implications in a single step, or combines DeMorgan with double negation. A student can ask for a demo, in which case Organon constructs a DNF stepwise. The training mode is part of a full learning management system with modules for homework, and exams that can be graded automatically.

[3]`http://logik.phl.univie.ac.at/~chris/gateway/`
[4]`http://organon.kfi.zcu.cz/organon/`

	outer loop			interactions		feedback			feed forward		
tool	type	exercises	levels	input	direction	syntax	rule	str	hint	solution	adaptation
Gateway to Logic	NF	US	n.a.	-	n.a.	KR	n.a	n.a.	NE	SOL	NO
Organon	NF	RA	NO	FO	n.a.	KR	KR	-	NE	WO	NO
Logicweb	NF*	FI, US	NO	RU	n.a.	KR	n.a.	KR	-	-	NO
SetSails	EQ	FI, US	NO	RaF	2D	KCR	KCR	-	GE***	-	YES
Logic Cafe	EQ, CO	FI, US	YES	RaF	2D	KR	KCR	-	GE, FF1	WO	NO
FOL equivalence	EQ**	FI	?	RaF	?	?	KM?	-	?	-	NO
Deep Thought	CO	FI, US	NO	RaF	2D	KR	KCR	-	HI, NE***, FF1	-	NO
Logic Coach	CO	FI, US	YES	RoF	2D	KR	KCR	-	NE, FF2	-	YES

Fig.3. Comparison of logic tools and their characteristics

Type	NF: normal form; EQ: equivalence proofs; CO: consequence proofs; *: normal forms as part of a resolution proof; **: equivalence proof in first order logic
Exercises	US: user defined exercises; RA: randomly generalised exercises; FI: fixed set
Levels	YES: different difficulty levels; NO: no levels; n.a.: if the tool offers no exercises; ?: unknown
Input	FO: input a formula; RU: input a rule name; RaF: input a rule name and a formula; RoF: input a rule name or a formula
Direction	2D: student can work forwards and backwards; n.a.: not applicable, because tool does not offer these exercises
Syntax	n.a.: not applicable; KR: correct/incorrect; KCR: correction of (some) syntax errors
Rule	n.a.: not applicable; KR: correct/incorrect; KCR: explanation i.e. a rule-example
Str	n.a.: not applicable; KR: step does/does not follow a desired strategy
Hint	NE: LE provides next step; GE: list of possible useful rules, subgoal, etc.; HI: hint, suggestion which rule to use next ***: not always available; FF1: feed forward only in one direction (top down); FF2: feed forward in two directions (top down and bottom up)
Solution	SOL: worked out solution for each exercise; WO: worked out demos
Adaptation	YES: users may adapt the rule set; NO: users cannot adapt the rule set



Zeigen Sie:
$(A \wedge B) \vee (A \wedge \neg B) = A$

A	B	$(A \wedge B) \vee (A \wedge \neg B)$
0	0	0
0	1	0
1	0	1
1	1	1

	Term	Regel
1	$(A \wedge B) \vee (A \wedge \neg B)$	Distributivgesetz
2	$= (A \vee (A \wedge \neg B)) \wedge (B \vee (A \wedge \neg B))$	Absorptionsgesetz
3	$= A \wedge (B \vee (A \wedge \neg B))$	
4		
5	$= A \wedge 1$	Neutrales Element der Und-Operation
6	$= A$	

Überprüfung der Lösung

Sie haben den Beweis noch nicht abgeschlossen. Bitte verwenden Sie die Tabelle, um den Term umzuformen, bis Sie die rechte Seite der Gleichung erreichen.

- Eine Ihrer Transformationssequenzen beinhaltet den Wahrheitswert "wahr", die andere nicht. Sie werden im Verlauf Ihres Lösungsprozesses eine der Regeln Absorbierende Elemente, Komplementäre Elemente und Neutrales Element der Und-Operation anwenden müssen, um dies aufzulösen.

Fig.4. Screenshot of a session with SetSails

Logicweb. Logicweb[5] is a tool for practicing resolution (and semantic trees), and strictly spoken not a tool for rewriting a formula into normal form. However, to solve an exercise, a student starts with rewriting the given formulae in clausal form (conjunctive normal form), using standard equivalences. Logicweb is an example of a tool where rewriting is performed automatically. At each step, the student selects a formula and the tool offers a list of (some of the) applicable rules. The student picks the rule to apply, and the tool applies it. In this way, a student does not have to worry about applying rules, but can focus on the strategy to solve an exercise. The only mistake a student can make is choosing a 'wrong' rule, i.e. a rule that does not bring the student closer to a solution. Rules can only be applied in one direction, hence the only possible 'wrong' rule is distribution of and over or, since that rule can bring a student further from a clausal form. If a student chooses to distribute and over or, the tool can tell the student that this is not the correct rule to apply at this point in the exercise. The tool contains a fixed set of exercises, but user-defined exercises are also possible. In the latter case the tool reports syntactic errors.

Proving an equivalence

SetSails. SetSails[6] is a tool developed within the Sail-M project, in which several e-learning tools for mathematics are developed (Zimmermann and Herding, 2010; Herding et al., 2010). Figure 4 shows a screenshot of a session with SetSails. SetSails exploits the idea that set-algebra is a special case of Boolean algebra. It offers two kinds of exercises: prove that two expressions in set-algebra denote the same set, or prove that two formulae from propositional logic are equivalent. Here we will only look at

[5]`http://ima.udg.edu/~humet/logicweb`
[6]`http://sail-m.de/`

the last kind of exercises. SetSails contains a (small) set of predefined exercises, but it is also possible to enter an exercise.

SetSails provides immediate feedback on the syntax of a formula and automatically adds parentheses if a formula is ambiguous. In each step a student chooses a rule, and the system suggests possible applications of this rule, from which the student picks one. However, some of these alternatives are deliberately wrong: in some cases another rule is applied, or the suggested formula contains a common mistake. Choosing an alternative is thus a kind of multiple choice exercise, but it is also possible to enter your own formula, in case it is missing in the list of suggested formulae. Further feedback, such as corrections on the applied rules and hints, is only available when the student asks the system to check the proof, which can be done at each step. The hints and feedback are rather global. The system recognises if a new formula is equivalent to the old one, but cannot be obtained by rewriting with a particular rule, and also recognises when the rule name does not correspond to the rule used. Although the alternative rewritings offered by the LE seem to be generated by some buggy rules, these are not mentioned when a student chooses a wrong alternative. The hints mention the rules you might need, but not how you apply them, and the list of the rules you need is not complete. This form of feed forward is based on rule-level analysis, combined with some general observations on strategies for solving the exercises. The system does not provide next steps or complete solutions. After entering an exercise, a teacher (or even a student) can choose rules from a predefined set or add new rules that can be used in a derivation. This makes it possible to adapt the rule set, or to use previous proofs as lemmas in new proofs. However, the tool does not guarantee that an exercise can be solved with the set of rules provided by a user. A user might have forgotten to include some essential rules from the rule set. A user can work both forwards and backwards, but the tool does not give advice about these directions. A nice feature of the system is the availability of truth-tables, which enables the user to check the equivalence or non-equivalence of the next step with the previous one.

Logic Cafe. Logic Cafe[7] contains exercises covering most of the material of an introductory logic course. The part on natural deduction contains some exercises in which a student has to rewrite a formula by using standard equivalences, and some exercises about proving consequences using standard equivalences together with one or two derivation rules. If a student makes a mistake in a step, it is not accepted, and in some cases Logic Cafe gives global feedback about the reason, for example that the justification should start with the number of the line on which the rule is applied, or that the justification contains an incorrect rule name. When a student asks for a hint, she gets a list of rules she has to apply, and in the more advanced exercises in which rewriting is combined with natural deduction, the feed forward is divided in two or more steps (a first hint tells which rules the user should start with, and a second hint lists some rules to use in the rest of the proof). However, this kind of feed forward is only available for predefined exercises. A student can enter her own exercise, but then there are no hints available. The LE contains some small animations that illustrate the construction of a proof, and some example derivations in which the LE tells a student exactly what to do at each step.

[7]`http://thelogiccafe.net/PLI/`

The FOL equivalence system. In the FOL equivalence system (Grivokostopoulou et al., 2013) a student practices with proving the equivalence between formulae in first order logic. We describe the tool here because it uses a standard set of rewriting rules for the propositional part of the proof. A student selects an exercise from a predefined set of exercises. To enter a step she first selects a rule, and then enters the formula obtained by applying the rule. The system checks this step, and gives a series of messages in case of a mistake. The first message is about correctness. Successive messages are more specific and give information about the mistake and how to correct it. As far as we could determine, the system does not give a hint or a next step if a student does not select a rule, and does not provide complete solutions. It is not clear whether a student can work forwards, backwards, or both. The system is in Greek and is not freely available.

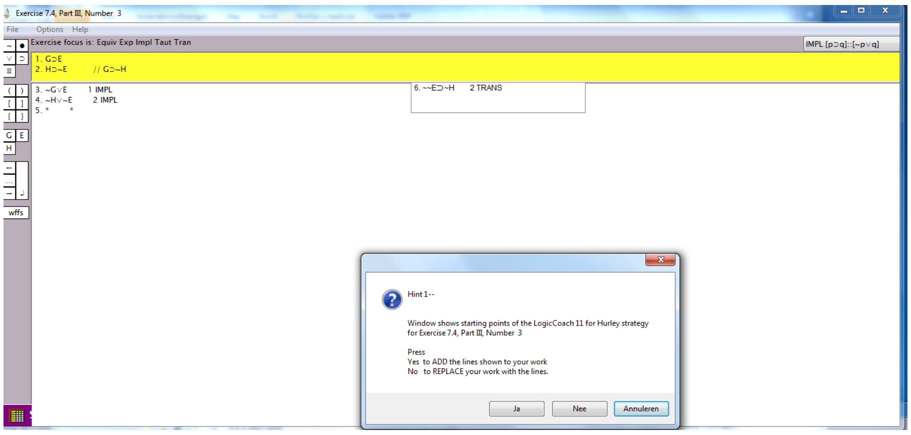
Deriving a consequence

Most of the tools that support deriving a consequence combine rewriting with a form of natural deduction.

Deep Thought. Deep Thought[8] is a tool for proving logical consequences with the use of rewrite rules (standard equivalences) and inference rules (Stamper et al., 2011a,b). It offers three predefined sets of exercises, about inference, inference more difficult, and inference and replacement. A student stepwise builds a proof tree in an exercise. At each step she selects a rule and one or more nodes of the tree where the rule should be applied. If several applications are possible, the student has to enter the next step, otherwise this step is calculated automatically. The tool gives feedback if a student chooses a non-applicable rule, makes a mistake in applying a rule, or applies more than one rule at a time. In all these cases the same standard error message is given. Students can work forwards and backwards, but rewrite rules can only be applied forwards. The distinguishing characteristic of this tool is the way hints are provided. Hints are generated by a hint factory. The hint factory uses a database of sample solutions. The different solutions of an exercise are used in a solution tree. When a student asks for a hint, the system tries to find the current state of the student in this solution tree. Four possible hints can be generated: a subgoal to be derived, which rule to apply, the nodes on which the rule should be applied, and a complete next step.

Logic Coach. Logic Coach is an extensive e-learning environment that supports the material covered in Hurley's textbook (Hurley, 2008) in (philosophical) logic. The section on propositional logic contains a series of exercises in proving consequences with the use of inference rules and standard equivalences (natural deduction). This fixed set of exercises is ordered in a didactic way, starting with exercises where students have to recognise which rules are applicable, followed by simple proofs. Each next set contains more complicated exercises, for example to train a special set of rules. Students have to enter new formulae together with a rule name and a line number to justify their step. However, they can use wildcards, for either the formula or the justification. In the first case the rewriting is done by the system, in the second case the system tries to find a justification. Feedback consists of mentioning that a rule name is missing, or a mistake is made in applying a rule. In the latter case the general form of the rule

[8]`http://itsxserve.uncc.edu/philosophy/tallc/applets/core/dt_hint/dt_hint.html`



Exercise 7.4, Part III, Number 3

Exercise focus is: Equiv Exp Impl Taut Tran

IMPL [p⊃q]::[~p∨q]

```
1. G⊃E
2. H⊃~E        // G⊃~H
3. ~G∨E       1 IMPL
4. ~H∨~E       2 IMPL
5. *       *
```

```
6. ~~E⊃~H      2 TRANS
```

Hint 1--

Window shows starting points of the LogicCoach 11 for Hurley strategy for Exercise 7.4, Part III, Number 3

Press
Yes to ADD the lines shown to your work
No to REPLACE your work with the lines.

Fig.5. Screenshot of feed forward by Logic Coach

is shown or, for simple exercises, the correct application is given. A student can ask for feed forward by entering a wildcard for both the formula and the justification. In this case the system provides a part of a standard solution. The hints do not depend on the work already done by the student: a first hint always consists of a first part of the proof, and a second and third hint provide the rest. Often, the second hint contains one or more backward steps. Figure 5 provides an example of this feed forward. If a student adds a backward step to the proof and asks the tool to check whether this step is correct, the tool complains about a missing justification, since each step has to be justified by a previous line. This means that a student can work in both directions, but when working backwards only the complete proof can be checked. A student can also construct her own proof and have it checked by the tool. Clicking the help button gives a summary of the theory that is needed to solve the exercise. It is possible to adapt the rule set by adding or removing rules. However, it is only possible to add rules from a predefined set, and it is not guaranteed that the exercise remains solvable with the user-defined rule set.

The comparison

We compare the above tools by means of the aspects described at the beginning of this section.

Most tools contain a predefined set of exercises, some offer the possibility to enter an exercise (and check whether or not the exercise is solvable), and a third category of tools offers randomly generated exercises. Most tools do not distinguish between difficulty levels, but Logic Coach, for example, has structured its exercises in such a way that a student starts with very simple exercises and proceeds to more difficult ones.

There is lot of variation in the kind and content of the feedback. This partly depends on the way a student works in the tool. For example, if a student does not enter a formula, the tool does not provide feedback on syntax, as in Logicweb and Gateway to Logic. The other tools offer some kind of feedback on syntax. The feedback might be to not accept a non-grammatical formula (as in Logic Cafe), or to automatically add missing parentheses (SetSails). Feedback on the rule-level may consist of mentioning that a rule name is not correct, or that a mistake has been made. Here, SetSails and Deep Thought give a general form of the correct rule to be applied. FOL equivalence is the only tool that claims to give error specific feedback. None of the other tools report common mistakes (KM). Logicweb is the only tool that can give feedback on the strategic level, namely when the student uses the wrong distribution rule.

The feed forward varies a lot between the different tools too. There is some correlation between the type of exercise and the kind of feed forward. For the 'easy' exercises (rewriting into normal form), tools do provide feed forward, such as complete solutions to exercises as given by Gateway to Logic and Organon. For the other tools, feed forward is more restricted, and mainly consists of general observations about the rules you might need to solve the problem. Only Deep Thought provides much more detailed feed forward in some cases. For some steps it starts with a general observation about the goal to be reached, then suggest rules to apply, ending with the actual next step. Because this feed forward is based on a database of worked-out solutions, it is not always possible to receive feed forward, especially when a student diverges from the standard strategy.

Most tools for proving equivalence of formulae and the tools for proving consequences offer the possibility to work in two directions. However, apart from Logic Coach none of these tools offer hints on whether to perform a forward or a backward step, and it is impossible to receive a next backward step.

SetSails and Logic Coach are the only tools that offer the user the possibility to define her own set of rules. However, these tools do not adapt the feed forward to this user set.

In conclusion, there already are a number of useful LEs for propositional logic, but there remains a wish-list of features that are not, or only partially, realised in these LEs. The main feature missing in almost all tools is feed forward: only the LEs for practicing normal forms offer next steps or complete solutions in any situation. Tools on proving equivalences or consequences do not provide feed forward, or provide feed forward only in a limited number of situations. This might be caused by the fact that the decision procedures for solving these kinds of exercises are not very efficient or smart. A good LE cannot only provide feed forward for a standard way to solve an exercise, but also for alternative ways. It should also support students that use both forward and backward steps in their proofs.

The feedback provided in the existing LEs for propositional logic is also rather limited. A good LE should, for example, have the possibility to point out common mistakes (KM).

We hypothesise that the number of tools for propositional logic is relatively high because different teachers use different logical systems with different rule sets. An LE that is easily adaptable, with respect to notation, rule sets, and possibly strategies for solving exercises, might fulfil the needs of more teachers.

FEEDBACK SERVICES

Traditionally, the architecture of an intelligent tutoring system (ITS) is described by means of four components (Nwana, 1990): the expert knowledge module, the student model module, the tutoring module, and the user interface module. In this arrangement, the expert knowledge module is the component responsible for 'reasoning about the problem', i.e., for managing the domain knowledge and calculating feedback and feed forward. Typically, this component also includes a collection of exercises, and knowledge about the class of exercises that can be solved. Following Goguadze (2011), we will use the term *domain reasoner* for this component. We now discuss how to construct a domain reasoner for propositional logic that meets the expectations for all characteristics that were introduced in the previous section.

There are several approaches to develop a domain reasoner: we follow an approach that reflects in the software how a human tutor would solve an exercise, which allows us to report not only the correctness of a step and a possible next step, but also to explain why a particular step is helpful. This approach is known as the principle of cognitive fidelity Beeson (1998). At the end of this section we discuss alternative approaches to provide feedback and feed forward, and we reflect on their advantages and limitations.

A domain reasoner provides feedback services to an ITS (Heeren and Jeuring, 2014). More specifically, we promote a client-server style in which the stateless feedback services of the domain reasoner can be used by learning environments by sending JSON or XML requests over HTTP. Three categories of feedback services can be identified: services for the outer loop, services for the inner loop, and services that provide meta-information about the domain reasoner or about a specific domain, such as the list of rules used in a domain. The feedback services are domain independent, and can be used for many domains, including rewriting to DNF or CNF, proving two formulae equivalent, and deriving a consequence.

Services for the outer loop

The feedback services that we offer to support the outer loop are:

– give a list of predefined *examples* of a certain difficulty
– *generate* a new (random) exercise of a specified difficulty
– *create* a new user-defined exercise

The last service (*create*) was recently added to support user-defined exercises: the service checks that the new exercise can be solved by the solution strategy for the particular class of exercises.

We have defined a random formula generator that is used for the DNF and CNF exercises, but we do not generate random pairs for equivalence proofs (or consequences). In the formula generator, we have taken some precautions to reduce the number of unnatural formulae that are generated: we remove top-level disjuncts that are already in DNF, we prevent occurrences of $p \vee p$ (and similarly for the other connectives), and we check that the exercise can be solved in a reasonable number of steps, depending on the difficulty level (i.e., not too many or too few steps). The constants true and false are only used in 'easy' mode, and the frequency of equivalences and implications is lower than for the other connectives.

Services for the inner loop

The feedback services targeted at the inner loop are more diverse. For generating feedback, the *diagnose* service is used. *diagnose* analyses a student step and can detect different types of mistakes, such as syntactical mistakes, common misconceptions, strategic errors, etc. Note that we had to extend this service to also take into account which rule was selected by the student, and to recognise a mismatch between the rule that was recognised and the rule that was selected. Feed forward can be calculated with the *allfirsts* service, which returns a list of possible next steps based on a strategy, in combination with services for producing worked-out solutions, calculating the number of steps remaining, the rules that can be applied, etc.

To provide feedback services for a class of exercises in a particular domain, we need to specify (Heeren and Jeuring, 2014):

– The *rules* (laws) for rewriting and common misconceptions (buggy rules). Later in this section we present rules for propositional logic.
– A *rewrite strategy* that specifies how an exercise can be solved stepwise by applying rules. In the remainder of the paper we define various strategies for exercises in the logic domain.
– Two relations on terms: semantic *equivalence* of logical propositions compares truth tables of formulae, whereas syntactic *similarity* compares the structure of two formulae modulo associativity of conjunction and disjunction. These relations are used for diagnosing intermediate solutions.
– Two predicates on terms. The predicate *suitable* identifies which terms can be solved by the strategy of the exercise class. The predicate *finished* checks if a term is in a solved form (accepted as a final solution): for instance, we check that a proposition is in some normal form, or that an equivalence proof is completed.

The rules and the rewrite strategies have an explicit representation, and we argue that this improves adaptability and reuse of these components. We will come back to the issue of adaptability later.

Alternative approaches

There are different ways to construct feedback or feed forward for logic exercises. Feedback can be hard coded, but this is very laborious, especially since solutions are not unique. A second problem with this approach is to also provide feedback or hints when students deviate from the anticipated solution paths. One way to overcome this is to use a database with example solutions. This paradigm had been described by Aleven et al. (2009), and an implementation of this idea for a logic tutor is described by Stamper et al. (2011b). With data mining techniques based on Markov decision processes it is possible to automatically derive complete solutions and intermediate steps. These solutions and steps are then hard coded. In this way, the authors claim to be able to provide a hint in 80% of the cases. Another advantage of using example solutions over using solution strategies as we describe above, is that it is not always clear how to define such a strategy. The tool of Stamper, Deep Thought, offers exercises in proving consequences, where students can use a greater set of standard consequences than we use. This makes the problem of finding good strategies much harder, and the use of a database of examples can be a good solution in this case.

The use of example solutions also has some disadvantages. In our experience with Deep Thought, if a solution diverges from a ‘standard’ solution, there are often no hints available. Furthermore, the system can only solve exercises that are equal to (or resemble) the exercises in the database. Another, more serious drawback of the method of Stamper is that although the tool supports both forward and backward solving of a problem, there are only hints available for the forward way of proving.

The use of services in the LogEx learning environment

We have developed a domain reasoner for logic, which is used in the LogEx learning environment[9]. In this section we describe how LogEx deals with the characteristics given in Figure 3. LogEx presents exercises on rewriting a formula into normal form and on proving equivalences. We use all three kinds of exercise creation: users can enter their own exercises, LogEx generates random exercises for normal form exercises, and LogEx contains a fixed set of exercises for proving equivalence. The generated exercises and the fixed set of exercises have three levels of difficulty.

A student enters formulae. When proving equivalences a student also has to provide a rule name. In the exercises about rewriting to normal form this is optional. Equivalence exercises can be solved by taking a step bottom-up or top-down.

LogEx only accepts the application of one rule at a time. In exercises on proving equivalences it is obvious that the use of more than one rule in a step is not allowed since students have to provide a rule name. The reason why we do not allow the application of more than a single step in an exercise on rewriting to normal form is that it is impossible to reconstruct the individual steps in such a student solution, and hence, impossible to give KM feedback on the level of rules. It is not hard to see that a reconstruction of the steps a student has taken is impossible if more than one rule is applied. There are infinitely many different ways to rewrite a formula ϕ into an equivalent formula ϕ', because the set of formulae that are equivalent to ϕ is infinite, and for any ψ in this set, there is a sequence of rewrite steps from ϕ into ϕ' via ψ. The same holds if a student makes a mistake during this rewriting, since in this case the possible rule set which is used to reach ϕ' is extended with buggy rules. Since it is impossible to reconstruct the sequence of steps taken by a student, it is also in general impossible to detect the rule application in which the mistake was made. Even worse, if a student rewrites ϕ into an equivalent ϕ', it is still possible that the student made one or more mistakes on the path from ϕ to ϕ'. Of course, there are situations in which an expert can point out in which rule a mistake was made with a high level of certainty.

For more advanced students, KM feedback is less important. In the future we want to offer a variant of the normal form exercises that allows the application of more than a single rule in a step.

Most of the feedback on syntax is of the KR type: only if parentheses are missing LogEx gives KCR feedback. LogEx provides KM feedback on the level of rules. It not only notes that a mistake is made, but also points out common mistakes, and mentions mistakes in the use of a rule name. LogEx does not support strategic feedback. LogEx accepts any correct application of a rule, even if the step is not recognised by the corresponding strategy. In such a case the domain reasoner restarts the strategy recogniser from the point the student has reached. Thus, LogEx can give hints even if a student diverges from the strategy.

LogEx gives feed forward in the form of hints on different levels: which formula has to be rewritten (in case of an equivalence proof), which rule should be applied, and a complete next step. Feed forward is given for both forward and backward proving, and even recommends a direction. LogEx also provides complete solutions.

LogEx does not offer the possibility to adapt the rule set. Later on in this paper we will sketch an

[9]`http://ideas.cs.uu.nl/logex/`

CommOr:	$\phi \lor \psi \Leftrightarrow \psi \lor \phi$
CommAnd:	$\phi \land \psi \Leftrightarrow \psi \land \phi$
DistrOr:	$\phi \lor (\psi \land \chi) \Leftrightarrow (\phi \lor \psi) \land (\phi \lor \chi)$
DistrAnd:	$\phi \land (\psi \lor \chi) \Leftrightarrow (\phi \land \psi) \lor (\phi \land \chi)$
AbsorpOr:	$\phi \lor (\phi \land \psi) \Leftrightarrow \phi$
AbsorpAnd:	$\phi \land (\phi \lor \psi) \Leftrightarrow \phi$
IdempOr:	$\phi \lor \phi \Leftrightarrow \phi$
IdempAnd:	$\phi \land \phi \Leftrightarrow \phi$
DefEquiv:	$\phi \leftrightarrow \psi \Leftrightarrow (\phi \land \psi) \lor (\neg\phi \land \neg\psi)$
DefImpl:	$\phi \rightarrow \psi \Leftrightarrow \neg\phi \lor \psi$
DeMorganOr:	$\neg(\phi \lor \psi) \Leftrightarrow \neg\phi \land \neg\psi$
DeMorganAnd:	$\neg(\phi \land \psi) \Leftrightarrow \neg\phi \lor \neg\psi$
ComplOr:	$\phi \lor \neg\phi \Leftrightarrow T$
ComplAnd:	$\phi \land \neg\phi \Leftrightarrow F$
DoubleNeg:	$\neg\neg\phi \Leftrightarrow \phi$
NotTrue:	$\neg T \Leftrightarrow F$
NotFalse:	$\neg F \Leftrightarrow T$
TrueOr:	$\phi \lor T \Leftrightarrow T$
FalseOr:	$\phi \lor F \Leftrightarrow \phi$
TrueAnd:	$\phi \land T \Leftrightarrow \phi$
FalseAnd:	$\phi \land F \Leftrightarrow F$

Fig.6. Rules for propositional logic

approach to supporting adaptation.

Rules

All LEs for propositional logic use a particular set of logical rules to prove that two formulae are equivalent, or to derive a normal form. There are small differences between the sets used. The rule set we use is taken from the discrete math course of the Open University of the Netherlands (Vrie and Lodder et al, 2009), see Figure 6. Variants can be found in other textbooks. For example, Burris (1998) defines equivalence in terms of implication, and Huth and Ryan (2004) leave out complement and true-false rules from the rule set.

Sometimes, using a particular rule set is rather tedious, and derivations may get long when strictly adhering to a rule set. For this reason we implicitly allow associativity in our solution strategies, so that associativity does not need to be mentioned when its application is combined with another rule. This makes formulae easier to read, and reduces the possibility of syntax errors. Commutativity needs to be explicitly applied, but we offer all commutative variants of the rules in Figure 6.

In our services we use generalised variants of the above rules. For example, generalised distribution distributes a subterm over a conjunct or disjunct of n different subterms, and we recognise a rewrite of $\neg(p \lor q \lor r \lor s)$ into $\neg p \land \neg(q \lor r) \land \neg s$ as an application of a generalised DeMorgan rule. These generalised rules are more or less implied by allowing implicit associativity. We use a generalization of absorption in our strategy to prove equivalences or consequences: $\phi \lor \psi$ can be rewritten into ϕ provided that ϕ and ψ are both conjunctions of literals[10] and all the literals of ϕ are contained in ψ. The same holds for the dual $\phi \land \psi$. For example, $(p \land r) \lor (\neg q \land p \land s \land r)$ can be rewritten into $p \land r$.

The solution strategies we use in our services are based on the strategies explained in our textbook (Vrie and Lodder et al, 2009). Besides the above rules, this textbook allows two logical consequences, see Figure 7. The rules of Figure 6 and Figure 7 suffice to solve all the exercises in the

[10]a literal is an atom or a negation of an atom

CONJINTRO: $\{\phi_1, \phi_2, ..., \phi_n\} \Rightarrow \phi_1 \wedge \phi_2 \wedge ... \wedge \phi_n$
CONJELIM: $\phi_1 \wedge \phi_2 \wedge ... \wedge \phi_n \Rightarrow \phi_i$

Fig.7. Rules for logical consequences

textbook.

Buggy rules describe common mistakes. An example of a buggy rule is given in the introduction of this paper, where a student makes a mistake in applying DeMorgan and rewrites $\neg(p \vee q) \vee (\neg\neg p \wedge \neg q) \vee \neg q$ into $(\neg p \vee \neg q) \vee (\neg\neg p \wedge \neg q) \vee \neg q$. This step is explained by the buggy rule $\neg(\phi \vee \psi) \not\Leftrightarrow \neg\phi \vee \neg\psi$; a common mistake in applying DeMorgan. In case of a mistake, our diagnose service tries to recognise if the step made by the student matches a buggy rule. The set of buggy rules we use is based on the experience of teachers, and includes rules obtained from analysing the log files of the diagnose service.

A strategy language

Some of the learning goals of rewriting exercises in propositional logic are applying standard equivalences correctly, recognising applicable rules, converting a formula into normal form, and obtaining strategic insight in manipulating logic expressions. With strategic insight we mean the mental processes involved in making choices about different possible courses of actions (Craig, 2007). In the case of exercises in logic, this means that a student considers the various steps that can be taken, and makes an informed choice between these. Thus a student thinks ahead, for example to recognise that a rule that normally complicates a formula, can be used in certain cases to simplify the formula in a next step. Although some textbooks give strict procedures for converting a formula into normal form (Huth and Ryan, 2004), most books only give a general description (Vrie and Lodder et al, 2009; Burris, 1998), such as: first remove equivalences and implications, then push negations inside the formula using DeMorgan and double negation, and finally distribute and over or (DNF), or or over and (CNF). In general, textbooks do not describe procedures for proving equivalences or consequences, and these procedures do not seem to belong to the learning goals. We hypothesise that the text books present these exercises and examples to make a student practice with the use of standard equivalences (Dalen, 2004; Ben-Ari, 2012). The only exception we know of is the work of Copi et al. (2011), who discusses proof patterns, and gives some heuristics. Since we want to provide both feedback and feed forward, we need solution strategies for our exercises.

We use *rewriting strategies* to describe procedures for solving exercises in propositional logic, to generate complete solutions and hints, and to give feedback. To describe these rewrite strategies we use the strategy language developed by Heeren et al. (2010). Here we recapitulate the main components of this language, and extend it with a new operator. The logical rules (standard equivalences) that a student can apply when rewriting a formula in normal form or proving the equivalence of two formulae are described by means of rewriting rules. These rules are the basic steps of a strategy. We use combinators to combine two strategies, so a strategy is a logical rule r, or, if s and t are strategies then:

– $s <\star> t$ is the strategy that consists of s followed by t

– $s <|> t$ is the strategy that offers a choice between s and t
– $s >|> t$ is the strategy that offers a choice between s and t, but prefers s
– $s \rhd t$ is a left-biased choice: t is only used if s is not applicable
– $repeat\ s$ repeats the strategy s as long as it is applicable

We offer several choice operators. The preference operator is new, and has been added because we want to give hints about the preferred next step, but allow a student to take a step that is not the preferred step. For example, consider the formula:

$$(p \lor s) \land (q \lor r) \land (u \lor v)$$

To bring this formula into DNF we apply distribution. We can apply distribution top-down (to the first conjunct in $(p \lor s) \land ((q \lor r) \land (u \lor v))$) or bottom-up (to the second conjunct). A diagnosis should accept both steps, but a hint should advise to apply distribution bottom-up, because this leads to a shorter derivation.

The first step in proving that a formula is a consequence of a given set of formulae is often an application of the rule conjunction-introduction. This rule is only applicable if the set of assumptions contains more than one formula. For an application of such a rule we use the combinator try:

– $try\ s$ is the strategy that applies s when it is applicable

We also want to specify the direction in which a strategy is applied: bottom-up or top-down. For example, consider the strategy $simplify$, which we might use in a strategy for rewriting a formula into normal form.

$$simplify = \text{TRUEAND} <|> ....$$

This strategy can be used anywhere in a formula, but requires fewer steps when it is applied top-down. For example, in the formula $T \land (p \lor q \lor F)$ we can apply the rules TRUEAND and FALSEOR. When we apply the first rule we reach a DNF in one step, otherwise it takes us two steps. In other situations it does not matter where we apply the strategy. Therefore, we use the following combinators:

– $somewhere\ s$ applies s somewhere in the formula
– $oncetd\ s$ applies s once top-down
– $oncebu\ s$ applies s once bottom-up

Besides these general combinators we use some special combinators: we introduce these combinators when they are first used.

STRATEGIES FOR PROPOSITIONAL LOGIC EXERCISES

This section gives strategies for rewriting a logic formula to normal form, for proving the equivalence of two logical formulae, and for proving that a logical formula is a consequence of a set of logical formulae. Furthermore, we show how a strategy can be adapted in various ways.

A strategy for rewriting a formula to DNF

There are several strategies for rewriting a formula to DNF. A first strategy allows students to apply any rule from a given set of rules to a formula, until it is in DNF. Thus a student can take any step and find her own solution, but worked-out solutions produced by this strategy may be unnecessarily long, and the hints it provides will not be very useful. A second strategy requires a student to follow a completely mechanic procedure, such as: first remove implications and equivalences, then bring all negations in front of atomic formulae by applying the DeMorgan rules and removing double negations, and conclude with the distribution of conjunctions over disjunctions. This strategy teaches a student a method that always succeeds in solving an exercise, but it does not help to get strategic insight. This strategy also does not always produce a shortest solution. The problem of finding a shortest derivation transforming a formula into DNF is decidable, and we could define a third strategy that only accepts a shortest derivation of a formula in DNF. There are several disadvantages to this approach. First, it requires a separate solution strategy for every exercise. If a teacher can input an exercise, this implies that we need to dynamically generate, store, and use a strategy in the back-end. This might be computationally very expensive. Another disadvantage is that although such a strategy produces a shortest derivation, it might confuse a student, since the strategy might be too specialised for a particular case. For example, to rewrite a formula into DNF, it is in general a good idea to remove implications and apply DeMorgan before applying distribution. However, in the formula

$$\neg(q \vee (p \rightarrow q)) \wedge (p \rightarrow (p \rightarrow q))$$

a derivation that postpones rewriting the implication $p \rightarrow q$ and starts with applying DeMorgan and distribution takes fewer steps than a derivation that starts with removing the three implications. However, if the strategy provides the hint to apply DeMorgan, a student might not understand why this hint is given in this case. We think that a strategy does not always have to produce a shortest derivation. This implies that there might be situations in which a student can construct a solution with fewer steps than the strategy. As long as the LE accepts such a solution, this need not be a problem.

We choose to develop a variant of the second strategy: a strategy based on a mechanical procedure that allows a student to deviate from the procedure, and which uses heuristics to implement strategic insights. This strategy offers a student a basis to start from when she works on an exercise, but also stimulates finding strategically useful steps that lead to shorter derivations. These steps include steps that at first sight seem to complicate formulae, but offer possibilities for simplification later on (Schoenfeld, 1987). Our strategy is a more specific variant of the strategy described in the Open University textbook (Vrie and Lodder et al, 2009). It prescribes an order in which substrategies are applied. For example, simplifying a formula has highest priority while distributing and over or has lowest priority. However, when a formula is a disjunct, both disjuncts can be rewritten separately, in any order. For example, a student might solve the following exercise

$$(\neg\neg p \wedge (q \vee r)) \vee (p \rightarrow \neg\neg q)$$

by first rewriting the first disjunct, reducing it to DNF in two steps:

$$\begin{array}{ll}(\neg\neg p \wedge (q \vee r)) \vee (p \rightarrow \neg\neg q) & \Leftrightarrow \\ (p \wedge (q \vee r)) \vee (p \rightarrow \neg\neg q) & \Leftrightarrow \\ (p \wedge q) \vee (p \wedge r) \vee (p \rightarrow \neg\neg q) & \end{array}$$

If a strategy requires that removing double negations is applied before distribution, this last step is not allowed, because the right-hand disjunct should be rewritten first. On the other hand, applying distribution before removing double negations leads to duplicating the subformula $\neg\neg p$. For such a situation we introduce the combinator $somewhereOr$:

- $somewhereOr\ s$: checks whether a formula is a disjunction and applies s to one of the disjuncts, otherwise s is applied to the complete formula.

If a formula is a disjunction, we can rewrite it to DNF by rewriting the disjuncts separately. However, sometimes it is possible to apply a simplification rule on multiple disjuncts. For example,

$$(\neg\neg p \wedge (q \vee r)) \vee (p \rightarrow \neg\neg q) \vee \neg(p \rightarrow \neg\neg q)$$

is best rewritten by using the complement rule on the last two disjuncts. For such cases we introduce a set of rules that we try to apply at top-level. This set consists of all simplification rules that are applicable to a disjunction: FALSEOR, TRUEOR, IDEMPOR, ABSORPOR, and COMPLOR. The substrategy $orRulesS$, the definition of which is omitted, applies one of these rules, if possible.

When applying the $orRulesS$ substrategy is no longer possible, the strategy for DNF continues with rewriting a formula into negation normal form (NNF). A formula in negation normal form does not contain implications or equivalences, and all negations occur in front of an atom. To obtain an NNF we introduce three substrategies:

- $simplifyStep$: simplify a formula by applying $orRulesS$, or the dual strategy $andRulesS$, or one of the three rules DOUBLENEG, NOTFALSE, or NOTTRUE
- $eliminateImplEquivS$: remove implications and equivalences by applying DEFIMPL or DEFEQUIV
- $deMorganS$: move negations down by applying DEMORGANOR or DEMORGANAND

The substrategy $nnfStep$ combines these three substrategies:

$$nnfStep = simplifystep \triangleright (eliminateImplEquivS >\!\!|\!\!> deMorganS)$$

The $nnfStep$ strategy performs at most one step. We use the $repeat$ combinator to perform all steps necessary to obtain an NNF. The resulting strategy always tries to simplify a formula first. After simplifying it removes implications or equivalences, simplifying in between, and then applies DeMorgan. In some situations it might be better to use another order, for example applying DeMorgan, before simplifying. The strategy does not give another order in its hints, but by using the preference combinator it accepts solutions using another order.

After obtaining an NNF we distribute conjunctions over disjunctions to obtain a DNF. This is achieved by the strategy $distrAndS$, the definition of which is omitted, which applies GENERALDISTRAND or DISTRAND, preferring the first rule.

We now have all the ingredients of a strategy for rewriting to DNF:

$$\mathit{dnfStrategy} = \mathit{repeat}\ (\mathit{orRulesS} <\!|\!> \mathit{somewhereOr}\ (\mathit{nnfStep} \triangleright \mathit{distrAndS}))$$

Note the use of repeat at the beginning of the strategy, which ensures that at each step we apply the complete strategy again, and hence simplify whenever possible. It also ensures that the strategy continues as long as there are still simplification rules applicable, even if a DNF has been obtained. When a student asks whether the formula is in DNF in this case, the predicate *finished* returns true. This implies that a student can stop when a formula is in DNF, even when simplification steps are still possible, but the strategy accepts further simplification steps.

In some cases the $\mathit{dnfStrategy}$ does not recognise that a formula can be further simplified. The strategy simplifies a formula such as $p \lor (q \land \neg q \land r)$, but not $p \lor (q \land r \land \neg q)$, because q and $\neg q$ are not adjacent formulae in the conjunct. For this reason we introduce a strategy $\mathit{specialGroupLiterals}$, the definition of which is omitted, that checks if rearranging conjuncts makes it possible to apply a simplification. If this is the case, conjuncts are rearranged such that equal or negated conjuncts appear next to each other. We also use this strategy in our strategy for proving the equivalence between two formulae later in this section.

A second strategy that we add to our $\mathit{dnfStrategy}$ is a specialization of the distribution of disjunction over conjunction. In general, we do not allow distribution of disjunction in our strategy. However, if the distribution can be followed by a complement rule, it simplifies the formula. For example, applying distribution to $p \lor (\neg p \land q)$ leads to $(p \lor \neg p) \land (p \lor q)$, which can be simplified to $p \lor q$. For the same reason, if a formula is of the form $\phi \land (\neg\phi \lor \psi)$, distributing and over or before a possible application of DeMorgan shortens the derivation. We define a substrategy $\mathit{specialDistrNot}$ that is only used if after an application of a distribution rule a complement rule is applicable.

A third strategy, $\mathit{specialDeMorganNot}$, checks whether an application of DeMorgan leads to the possibility to simplify.

We obtain the following improved definition of $\mathit{nnfStep}$ when we include these three strategies:

$$\begin{aligned}\mathit{nnfStep} = {} & \mathit{simplifyStep} \\ & \triangleright (\mathit{specialGroupLiterals} >\!|\!> \mathit{specialDistrNot} >\!|\!> \mathit{specialDeMorganNot}) \\ & \triangleright (\mathit{eliminateImplEquivS} >\!|\!> \mathit{deMorganS})\end{aligned}$$

Adapting a strategy

We hypothesise that one of the reasons for the many variants of LEs for logic is that every teacher uses her own strategy or rule set. Our framework supports adapting strategies or rule sets.

Most textbooks describe how to rewrite a formula into DNF at an abstract level: first remove implications and equivalences, then bring all negations in front of atomic formulae by applying DeMorgan rules and removing double negations, and conclude with the distribution of conjunctions over disjunctions. In general, our own strategy, as described in the previous section, does not accept solutions produced by this global strategy, since it prescribes that simplifications should be made whenever possible, thus also before removing implications or equivalences. However, in our language we could easily define a strategy that follows this textbook description. Moreover, the preference operator makes it possible to define strategies that accept a broad range of solutions. For example, our strategy prefers applications of the generalised DeMorgan rule, but also accepts the simple DeMorgan rule.

Another way in which a teacher can adapt how our feedback services give support to students is by changing the rule set. Our DNF strategy is structured in a way that makes it easy to adapt the strategy for users who apply more, fewer, or different rules. Our basic strategy contains five substrategies that can be considered as sets of rules: $\mathit{orRulesS}$, $\mathit{simplifyStep}$, $\mathit{eliminateImplEquivS}$, $\mathit{deMorganS}$ and $\mathit{distrAndS}$. Note that the first four substrategies turn a formula into NNF, and the last substrategy turns a formula in NNF into DNF. To modify a strategy, a teacher can change the content of any of these five sets. We only want to accept a user-defined rule set if the resulting strategy rewrites formulae into DNF. For this purpose we use a 'minimal' strategy, in which each of these sets is as small as possible, so that removing a rule from a set implies that it is no longer guaranteed that a normal form is reached. A user-defined set should contain at least the rule sets of such a minimal strategy. In a minimal strategy the first set is empty, the second set contains the double negation rule, $\mathit{eliminateImplEquivS}$ contains exactly one rule that defines implication in terms of conjunctions, disjunctions and/or negations, and exactly one rule that defines equivalence in terms of implications, conjunctions, disjunctions and/or negations, $\mathit{deMorganS}$ contains the basic DeMorgan rules, and $\mathit{distrAndS}$ the rule for distributing a conjunction over a disjunction. For example, in a strategy with minimal rule sets, the set $\mathit{eliminateImplEquivS}$ may consist of the two rules $\{\phi \rightarrow \psi \Leftrightarrow \neg\phi \vee \psi, \phi \leftrightarrow \psi \Leftrightarrow (\phi \rightarrow \psi) \wedge (\psi \rightarrow \phi)\}$, or any other pair of rules that defines implication and equivalence.

A simple way to add more rules to the strategy is to use a preference combinator to add an extra substrategy $\mathit{userDefinedRules}$ at the end of the strategy. A user can include her own rules in $\mathit{userDefinedRules}$, but these are not used to produce feed forward, except for the case where also non-preferred possible steps are given. This guarantees that the feedback services still produce correct hints and solutions. If we add new rules to one of the five sets of our strategy, they *are* used when producing feed forward. We have to check that a rule can be used in the strategy and, if so, to which set it should be added. Our strategy has two properties: it is terminating, and if it terminates the resulting formula is in DNF. The last condition is met because of the minimal rule set, which guarantees that there are rules applicable whenever a formula is not in DNF. Adding a new rule may cause non-termination. To determine if a rule can be safely added, we introduce the concept of a *simplification rule*. To define this concept we need the notions conjunction depth (conjd) of a conjunction and negation depth (nd) of a negated (sub)formula:

$$\begin{aligned} &\mathit{conjd}\ p &&= 0 \\ &\mathit{conjd}\ (\phi \wedge \psi) &&= \mathit{bcd}\ (\phi \wedge \psi) \end{aligned}$$

where binary connective depth (bcd) is defined by:

$$\begin{aligned} &\mathit{bcd}\ p &&= 0 \\ &\mathit{bcd}\ (\neg\phi) &&= \mathit{bcd}\ \phi \\ &\mathit{bcd}\ (\phi \,\square\, \psi) &&= \mathit{bcd}\ \phi + \mathit{bcd}\ \psi + 1 \end{aligned}$$

where p is an atom, and $\square$ a binary connective. Negation depth is defined by:

$$\begin{aligned} nd\ p &= 0 \\ nd\ (\neg\phi) &= bcd\ \phi \end{aligned}$$

Let ϕ contain n_i negated subformulae with negation depth i, and let max be the maximum of these negation depths. We define the total negation depth $tnd\ (\phi)$ as the string:

$$n_{max}, n_{max-1}, ..., n_0$$

Let ϕ contain c_i conjunctive subformulae with conjunction depth i, and let max be the maximum of these conjunction depths. We define the total conjunction depth $tcd\ (\phi)$ as the string:

$$c_{max}, c_{max-1}, ..., c_0$$

We use the standard ordering $\leqslant$ on tnd and cnd.[11]

A rule $\phi \rightsquigarrow \phi'$ is a simplification rule if the following four conditions hold:

- $\phi \rightsquigarrow \phi'$ does not introduce implications or equivalences
- the length of ϕ' is less than the length of ϕ
- $tcd\ \phi' \leqslant tcd\ \phi$
- $tnd\ \phi' \leqslant tnd\ \phi$

These conditions prevent non-termination. We first show that the substrategy that consists of the first four rulesets (an NNF strategy) terminates. First remark that from these sets, the rules to eliminate implication and equivalence and DeMorgan are the only rules that increase the length of a formula. Implication elimination and equivalence elimination can only be applied a finite number of times. These are also the only rules that introduce new occurrences of negations. An application of DeMorgan on a formula ϕ reduces the total negation depth of a subformula of ϕ. Hence, DeMorgan can only be applied a finite number of times. This implies that also the number of steps in which the length of a formula increases is finite (equal to the sum of implication eliminations, equivalence eliminations, and applications of DeMorgan). Hence, after a finite number of steps, the strategy reaches the fifth subset. At this point all negations have a negation depth of zero.

And over or distribution can also be applied a finite number of times, since this rule reduces the total conjunction depth, and any other rules that will be applied in this phase of the strategy do not increase this total conjunction depth. Hence, the DNF strategy terminates after a finite number of steps.

We add a rewriting rule $r = \phi \rightsquigarrow \phi'$ to the first set to which it belongs, which is defined by:

- r belongs to *orRulesS* if it is a simplification rule and ϕ is a disjunction
- r belongs to *simplifyStep* if it is a simplification rule
- r belongs to *eliminateImplEquivS* if it defines implication in terms of conjunctions, disjunctions and/or negations, or defines equivalence in terms of implications, conjunctions, disjunctions and/or negations
- r belongs to *deMorganS* if it reduces $nd\ \phi$

[11] In a standard ordering $x \leqslant y$ iff either $length(x) \leqslant length(y)$ or $length(x) = length(y)$ and $x \leqslant_L y$, where $\leqslant_L$ denotes the lexicographic ordering on strings.

– r belongs to $distrAndS$ if it is DISTRAND or GENERALDISTRAND
– r belongs to $extrarules$ if it does not belong to any of the aforementioned sets

For example, a teacher may add $(\phi \wedge \neg\psi) \vee \psi \rightsquigarrow \phi \vee \psi$. This rule is added to $orRulesS$. The dual rule, $(\phi \vee \neg\psi) \wedge \psi \rightsquigarrow \phi \wedge \psi$, belongs to $simplifyStep$.

Using the strategy for DNF, we easily obtain a strategy for rewriting a formula into CNF or NNF. To obtain a strategy for CNF we change $orRulesS$ into a dual $andRulesS$, and $distrAndS$ into $distrOrS$ in the DNF strategy. A strategy for rewriting a formula to NNF is obtained by replacing $distrAndS$ by the empty strategy.

Some textbooks and tools do not use all commutative variants of a rule. We can also adapt our strategy for this purpose by replacing occurrences of a commutative variant of a rule by a substrategy that combines an application of commutativity with the rule. It is harder to accommodate associativity. Associativity does not appear in our rule set, but is implicitly applied. Applications of associativity are ignored, so a student can change parentheses in a conjunction without affecting the feedback services. It is possible to add associativity to our strategy, but then a student would always have to enter all parentheses. This makes formulae rather hard to read, and increases the possibility of syntax errors.

A strategy for proving two formulae equivalent

This subsection discusses a strategy for proving two formulae equivalent. This strategy builds upon the strategy for rewriting a formula into DNF. In particular, we discuss the heuristics used in this strategy.

The basic idea behind our strategy for proving two formulae equivalent is simple: rewrite both formulae into DNF and prove that the two resulting formulae are equivalent by extending the normal forms (Lodder and Heeren, 2011). However, without including heuristics in this strategy, students do not get the necessary strategic insight for this kind of problems, and derivations may become rather long. For instance, a proof of

$$p \vee (q \wedge r) \Leftrightarrow (p \wedge \neg q) \vee (p \wedge \neg r) \vee (q \wedge r)$$

by extending both normal forms takes at least 23 steps[12] plus applications of commutativity. A heuristic which uses absorption reduces this number to 11 steps plus applications of commutativity. Another advantage of using this heuristic, is that the formulae stay much shorter.

The first heuristic we use in our strategy is a general principle that divides a problem in smaller subproblems. In our case this means that if we for example want to prove $\phi \Leftrightarrow \psi$ and ϕ and ψ are both conjunctions: $\phi = \phi_1 \wedge \phi_2, \psi = \psi_1 \wedge \psi_2$, we first check using truth-tables whether or not $\phi_1 \Leftrightarrow \psi_1$ and $\phi_2 \Leftrightarrow \psi_2$ hold, or $\phi_1 \Leftrightarrow \psi_2$ and $\phi_2 \Leftrightarrow \psi_1$. If so, the strategy splits the proof in two subproofs, applying commutativity if necessary. The same steps are performed if ϕ and ψ are both disjunctions, negations, implications or equivalences. For example, a proof of

$$(p \wedge p) \rightarrow (q \wedge (r \vee s)) \Leftrightarrow p \rightarrow ((s \vee r) \wedge q)$$

[12]3 steps to rewrite p into $(p \wedge q) \vee (p \wedge \neg q)$, 6 times 3 steps to rewrite all disjuncts into disjuncts with all three different variables, and 2 applications of idempotence to remove duplicates.

takes only three steps: two applications of commutativity and one of idempotency. The strategy does not rewrite the implication, nor distributes and over or. After ϕ and ψ have been rewritten into simplified DNF, the heuristic rearranges conjuncts and disjuncts to try proving equivalence. Since normal forms are not unique, we need to rewrite these normal forms into complete normal forms in some cases. In a complete normal form, each conjunct contains all the occurring variables, possibly negated. Complete normal forms are unique up to commutativity. Since rewriting into complete normal forms may take quite a number of steps, and the resulting formulae may get very long, we introduce two additional heuristics.

If all the disjuncts of ϕ and ψ contain a common literal, we factor out this common literal by using inverse distribution. Note that when we rewrite a formula into complete normal form, we do not use distribution anymore, and hence prevent a loop. We now have to prove the equivalence of two simpler formulae, which might not make the proof shorter, but at least the formulae are smaller. [13]

A second heuristic exploits the idea that a complete normal form is seldom necessary. In fact, by using splitting rules of the first heuristic during normalization, in combination with applications of the absorption rule, we shorten proofs considerably. Also the subformula we choose to extend to normal form influences the length of the proof. For example, we do not choose subformulae that occur on both sides.

A strategy for proving consequences

Finally, we discuss a strategy that supports solving exercises in which a student proves a consequence. The goal of the exercises is to use the basic rewrite rules from Figure 6, together with the rules for conjunction introduction and conjunction elimination from Figure 7, to prove that a formula is a consequence of a set of formulae. Since conjunction elimination is our only consequence rule, we use a strategy that is similar but dual to the strategy for proving equivalences. A proof starts with an application of conjunction introduction, if necessary, then rewrites both sides into CNF, and derives the consequence from the assumptions by conjunction elimination. The strategy has a special case when the set of assumptions is inconsistent. We then rewrite the assumption set into F, and use FALSEAND to introduce the conclusion. The strategy for proving consequences reuses some of the heuristics of the strategy for proving equivalences. We can only use the splitting rules in case of a conjunction. If the assumption is a conjunction we also use we a new substrategy: if an assumption is equal, up to associativity and commutativity, to a formula $\phi \wedge \psi$, and we can derive the consequence χ from ϕ without using ψ, we do not rewrite ψ.

[13] We also made a small adaptation in the DNF strategy to prevent proofs such as:

$$\begin{array}{ll}
s \wedge (p \vee q) & \Leftrightarrow \{\text{Distribution}\} \\
(s \wedge p) \vee (s \wedge q) & \Leftrightarrow \{\text{Common literal}\} \\
s \wedge (p \vee q) & \Leftrightarrow \\
\ldots & \\
s \wedge (q \vee (p \wedge \neg q)) & \Leftrightarrow \{\text{Common literal}\} \\
(s \wedge q) \vee (s \wedge p \wedge \neg q) &
\end{array}$$

We divide the DNF strategy in the NNF part and applications of distribution. In between we check whether this special case occurs. If so, we factor out the literal on the right-hand side and use a splitting rule.

Our strategies for proving equivalences or consequences can also be used in set-algebra. The strategy to prove an equivalence can be used to prove equality of sets, and the strategy to prove a consequence can be used to prove inclusion. For example, Hartel et al. (Hartel et al., 1995) measure the proof skills of computer science students using, amongst others, an exercise to prove the inclusion $((A \cup B) \cap A^{\complement}) \subseteq B$. The example solution they give is exactly the solution that is calculated by our strategy, up to some applications of commutativity.

EXPERIMENTAL RESULTS

Since the first version of our tool we performed several small-scale experiments with students. A first evaluation took place in 2006 with a web-based learning environment in which we used a first version of our strategy for rewriting a formula into DNF (Lodder et al., 2008). In this experiment we wanted to obtain an answer to the following four questions:

– How good is the feedback provided by the tool?
– How do students use the tool?
– Do students learn by using the tool?
– How do students appreciate the tool?

We used a pretest and posttest, a questionnaire, and we analysed the logs of the learning environment. The students who participated in the experiment were divided into two groups. One group used the complete learning environment, the other group used a version with restricted feed forward: the environment did not provide next steps. Our main findings were that feed forward is essential for completing an exercise: without next steps some of the students could not finish the more difficult exercises, that students do learn by using the tool, and that the feedback was not always helpful. The learning environment did not use buggy rules, but generated feedback automatically based on the difference between the entered formula and all possible correct next steps. We found that students tend to interpret feedback as a hint: if, for example, the learning environment concluded that a student made a mistake in applying DeMorgan, students would try to apply DeMorgan in the next step. This was also the case when the environment wrongly concluded that the student intended to apply DeMorgan, while the student tried, for example, to apply distribution.

The next version of our learning environment added buggy rules to provide feedback. For example, with a carefully developed set of buggy rules it is less likely that the learning environment recognises a wrong application of distribution as a buggy application of DeMorgan.

We performed a small experiment with this new version of our learning environment, in which 5 students filled out a questionnaire. We performed two other experiments with the exercises about proving equivalences in the learning environment. After an experiment with the first prototype we received questionnaires of 3 students, and after an experiment with an improved version we received questionnaires of 4 students. Students also made a pretest and posttest, but the numbers are too low to draw conclusions from the results. The logs were mainly used to improve the set of buggy rules. Figure 8 summarises the answers to some of the questions on a scale from 1 (bad) to 5 (good).

Question	DNF	Equivalence proofs
Quality of feedback on syntax	3.0	3.2
Quality of feedback on rules	2.0	3.7
How helpful are the hints?	4.0	4.0
How helpful are the next steps?	4.3	4.1
How well can you make an exercise without help after practicing for some time?	4.0	4.8
Do you make fewer mistakes after some practicing?	3.4	4.6
Does the tool help you in getting a better understanding?	2.6	3.8
Does the tool help you in improving your skills?	3.2	4.4

Fig.8. Results of the small-scale experiments

Although the number of participants in the experiments are too low to draw conclusions, the results are promising. We cannot compare the results for the learning environments with exercises about rewriting into DNF and proving equivalences, since the latter environment has a significantly improved user interface. The different scores on the quality of rule feedback might be explained by the fact that we used a more complete set of buggy rules for proving equivalences, but also by the fact that a student has to motivate each step with a rule name in equivalence proofs. This makes it clear to a student that it is not allowed to apply more than one rule in a step. Students applying more than one rule in a single step in the environment for DNF get a message that the new formula is equivalent, and that she either made a mistake or applied more than one rule in a single step. This is not always very helpful.

CONCLUSIONS

We have used a framework for describing rules and strategies to develop strategies that support students solving exercises in propositional logic. The framework provides services for analysing student steps, and for giving feed forward such as hints, next steps, examples, and complete solutions. Our approach guarantees that this feedback and feed forward is available at any time when solving an exercise, also if a student diverges from a preferred solution. Our feedback and feed forward services have the following characteristics:

– we offer three exercise types (normal forms, equivalences and consequences)
– a student inputs either just a formula or a formula together with the name of a rule
– a student can work in two directions and receive feedback and feed forward in both directions
– the exercises for rewriting into DNF and CNF are randomly generated; the other sets of exercises are fixed. A student can also enter her own exercises
– an exercise has a difficulty level
– the services offer some type of KCR feedback at the syntax level; rule-level feedback deals with buggy rules. Our services can generate different types of feed forward: hints, next steps, and complete solutions

We have shown how we can adapt our strategy with different rule sets. Since feedback and feed forward are provided by services separate from a user-interface or learning environment, it is easy to adapt the feedback and feed forward for different languages or different notations of logical symbols. Adding a new connective, such as Sheffer's stroke, would take quite a lot of work.

At this moment the only kind of strategic feedback offered by our services consists of diagnosing that a student step does not follow a particular strategy. We do not use this feedback in our learning environment for logic, since we have not yet implemented a procedure to recognise if a student step might be smarter than the strategy. However, in some cases strategic feedback can be very useful, for example if a student arrives at the same expression after a number of steps. Our services do not yet recognise such a situation. Currently, feed forward gives a hint about which next step to take. It would be nice to offer feed forward at a more abstract level, for example by describing a subgoal, and then giving hints about how to reach this subgoal. The services would then not only inform a student which rule to apply, but also why it should be applied.

Our services do not accept syntactically incorrect formulae, including ambiguous formulae. In some cases this implies that a student cannot make the mistakes she might make using pen and paper when using the services. For example:

$$\begin{array}{l} \quad (\neg p \lor q) \land \neg(p \land q) \\ \Leftrightarrow \{\text{DeMorgan}\} \\ \quad (\neg p \lor q) \land \neg p \lor \neg q \\ \Leftrightarrow \{\text{Absorption}\} \\ \quad \neg p \lor \neg q \end{array}$$

In the first step the student forgets the parentheses around $\neg p \lor \neg q$, which makes the formula ambiguous. The application of absorption rewrites the formula into a non-equivalent formula. Our services do not accept the second formula.

In the exercises in which a student proves an equivalence or a consequence, a student applies one rule in a step. When a formula contains two occurrences of an implication, for example, it would be more user friendly to accept two applications of the definition of implication in one step. At this moment, our services do not recognise multiple applications of the same rule. In exercises on rewriting a formula into a normal form, teachers do not always ask students to provide the intermediate steps. In our services a student can apply only one rule in a step. A possible future extension of our learning environment would have a variant of the services for more advanced students where a user can apply more than one rule in a step, or even directly enter the final answer.

We have performed some small-scale experiments with a learning environment built on top of the services described in this paper, and the results are promising. We will perform larger experiments by the end of 2015, in which we will investigate whether our learning environment supports students in developing their skills and understanding of propositional logic.

Acknowledgements. We thank our students for experimenting with the tools and giving feedback. Maarten Hemker and Peter Dol developed the first version of a user interface for our learning envi-

ronment for proving equivalences, and René Dohmen and Renaud Vande Langerijt extended this for exercises about rewriting to a normal form, and made some further improvements.

REFERENCES

Aleven, V., Mclaren, B. M., Sewall, J., and Koedinger, K. R. (2009). A new paradigm for intelligent tutoring systems: Example-tracing tutors. *International Journal on Artificial Intelligence in Education*, 19(2):105–154.

Beeson, M. J. (1998). Design principles of Mathpert: Software to support education in algebra and calculus. In Kajler, N., editor, *Computer-Human Interaction in Symbolic Computation*, pages 89–115. Springer-Verlag.

Ben-Ari, M. (2012). *Mathematical Logic for Computer Science, 3rd Edition*. Springer.

Benthem, J. v., Ditmarsch, H. v., Ketting, J., Lodder, J., and Meyer-Viol, W. (2003). *Logica voor informatica, derde editie*. Pearson Education.

Burris, S. (1998). *Logic for mathematics and computer science*. Prentice Hall.

Copi, I., Cohen, C., and McMahon, K. (2011). *Introduction to logic*. Pearson Education, Limited.

Craig, A. J. (2007). Comparing research into mental calculation strategies in mathematics education and psychology. In *Proceedings of the British Society for Research into Learning Mathematics 29(1)*, pages 37–42.

Dalen, D. v. (2004). *Logic and Structure*. Universitext (1979). Springer.

Dostálová, L. and Lang, J. (2007). Organon – the web tutor for basic logic courses. *Logic Journal of IGPL*.

Dostálová, L. and Lang, J. (2011). Organon: Learning management system for basic logic courses. In Blackburn, P., Ditmarsch, H., Manzano, M., and Soler-Toscano, F., editors, *Tools for Teaching Logic*, volume 6680 of *Lecture Notes in Computer Science*, pages 46–53. Springer Berlin Heidelberg.

Goguadze, G. (May 2011). *ActiveMath - Generation and Reuse of Interactive Exercises using Domain Reasoners and Automated Tutorial Strategies*. PhD thesis, Universität des Saarlandes, Germany.

Grivokostopoulou, F., Perikos, I., and Hatzilygeroudis, I. (2013). An intelligent tutoring system for teaching fol equivalence. In *AIED Workshops*.

Hartel, P. H., Es, B. v., and Tromp, D. (1995). Basic proof skills of computer science students. In *1st Functional Programming languages in Education, FPLE 1995, Nijmegen, The Netherlands. pp. 269-287. Springer-Verlag LNCS 1022*, pages 269–287. Springer-Verlag.

Hattie, J. and Timperley, H. (2007). The power of feedback. *Review of Educational Research*, 77(1):81–112.

Heeren, B. and Jeuring, J. (2014). Feedback services for stepwise exercises. *Science of Computer Programming, Special Issue on Software Development Concerns in the e-Learning Domain*, 88:110–129.

Heeren, B., Jeuring, J., and Gerdes, A. (2010). Specifying rewrite strategies for interactive exercises. *Mathematics in Computer Science*, 3(3):349–370.

Herding, D., Zimmermann, M., Bescherer, C., Schroeder, U., and Ludwigsburg, P. (2010). Entwicklung eines frameworks für semi-automatisches feedback zur unterstützung bei lernprozessen. In *DeLFI*, pages 145–156.

Huertas, A. (2011). Ten years of computer-based tutors for teaching logic 2000-2010: Lessons learned. In *Proceedings of the Third International Congress Conference on Tools for Teaching Logic*, TICTTL'11, pages 131–140, Berlin, Heidelberg. Springer-Verlag.

Hurley, P. (2008). *A Concise Introduction to Logic*. Cengage Learning.

Huth, M. and Ryan, M. (2004). *Logic in Computer Science: Modelling and Reasoning about Systems*. Cambridge University Press.

Kelly, J. (1997). *The essence of logic*. The essence of computing series. Prentice Hall.

Lodder, J. and Heeren, B. (2011). A teaching tool for proving equivalences between logical formulae. In Blackburn, P., Ditmarsch, H., Manzano, M., and Soler-Toscano, F., editors, *Tools for Teaching Logic*, volume 6680 of *Lecture Notes in Computer Science*, pages 154–161. Springer-Verlag.

Lodder, J., Jeuring, J., and Passier, H. (2006). An interactive tool for manipulating logical formulae. In Manzano, M., Pérez Lancho, B., and Gil, A., editors, *Proceedings of the Second International Congress on Tools for Teaching Logic*.

Lodder, J., Passier, H., and Stuurman, S. (2008). Using ideas in teaching logic, lessons learned. *Computer Science and Software Engineering, International Conference on*, 5:553–556.

Narciss, S. (2008). Feedback strategies for interactive learning tasks. In Spector, J., Merrill, M., van Merriënboer, J., and Driscoll, M., editors, *Handbook of Research on Educational Communications and Technology*. Mahaw, NJ: Lawrence Erlbaum Associates.

Nwana, H. S. (1990). Intelligent tutoring systems: an overview. *Artificial Intelligence Review*, 4(4):251–277.

Race, P. (2005). *Making learning happen – A guide for post-compulsory education*. Sage.

Schoenfeld, A. (1987). Cognitive science and mathematics education: An overview. In Schoenfeld, A., editor, *Cognitive Science and Mathematics Education*, chapter 1, pages 1–32. Lawrence Erlbaum Associates.

Stamper, J. C., Barnes, T., and Croy, M. (2011a). Enhancing the automatic generation of hints with expert seeding. *Int. J. Artif. Intell. Ed.*, 21(1-2):153–167.

Stamper, J. C., Eagle, M., Barnes, T., and Croy, M. (2011b). Experimental evaluation of automatic hint generation for a logic tutor. In *Proceedings of the 15th International Conference on Artificial Intelligence in Education*, AIED'11, pages 345–352, Berlin, Heidelberg. Springer-Verlag.

Star, J. R. and Rittle-Johnson, B. (2008). Flexibility in problem solving: The case of equation solving. *Learning and Instruction*, 18(6):565–579.

VanLehn, K. (2006). The behavior of tutoring systems. *International Journal of Artificial Intelligence in Education*, 16(3):227–265.

VanLehn, K. (2011). The relative effectiveness of human tutoring, intelligent tutoring systems, and other tutoring systems. *Educational Psychologist*, 46(4):197–221.

Vrie, E. M. v. d. and Lodder et al, J. S. (2009). *Discrete wiskunde A, Lecture notes (in Dutch)*. Open Universiteit Nederland.

Waalkens, M., Aleven, V., and Taatgen, N. (2013). Does supporting multiple student strategies lead to greater learning and motivation? Investigating a source of complexity in the architecture of intelligent tutoring systems. *Computers & Education*, 60(1):159–171.

Zimmermann, M. and Herding, D. (2010). Entwicklung einer computergestützten lernumgebung für bidirektionale umformungen in der mengenalgebra. *Beiträge zum Mathematikunterricht 2010*.