Journal of Universal Computer Science, vol. 22, no. 8 (2016), 1097-1122
submitted: 6/1/16, accepted: 29/7/16, appeared: 1/8/16

A Domain Reasoner for Propositional Logic

Josje Lodder
(Open University of The Netherlands
Heerlen, The Netherlands
josje.lodder@ou.nl)

Bastiaan Heeren
(Open University of The Netherlands
Heerlen, The Netherlands
bastiaan.heeren@ou.nl)

Johan Jeuring
(Open University of The Netherlands and Utrecht University
Utrecht, The Netherlands
J.T.Jeuring@uu.nl)

Abstract: An important topic in courses in propositional logic is rewriting propositional formulae with standard equivalences. 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.
Key Words: propositional logic, normal forms, learning environment, domain reasoner, feedback, feed forward, intelligent tutoring
Category: L.2.3, L.3.0, L.3.5, L.3.6

1 Introduction

Students learn propositional logic in programs such as mathematics, philosophy, computer science, law, etc. Students learning propositional logic practice by solving exercises about rewriting propositional formulae. Most textbooks for propositional logic [Benthem et al., 2003, Hurley, 2008, v.d. Vrie 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 infeasible to give all of them in a textbook, or provide them online.

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. In these cases, access to an intelligent tutoring system (ITS) [VanLehn, 2006] might be of help.

Feedback is an important aspect of an ITS. Usually an ITS offers various kinds of feedback: a diagnosis of a student step, a hint for the next step to take, in various levels 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. An ITS that follows the steps of a student when solving a task can be almost as effective as a human tutor [VanLehn, 2011].

What kind of feedback do ITSs for propositional logic give? There are many tutoring systems for logic available [Huertas, 2011]. In this paper we look at systems that deal with standard equivalences, in which a student has to learn to rewrite formulae, either to a normal form or to prove an equivalence. We analyse what kind of feedback is offered by the various learning environments for rewriting propositional logic formulae, and what kind of feedback is missing, and we discuss how we can provide these kinds of feedback in a learning environment. To give feedback we define solution strategies (procedures describing how basic steps may be combined to find a solution) for several classes of exercises, and we discuss the role of our strategy language in defining these solution strategies.

Some interesting aspects of solving exercises in propositional logic are:

– Exercises such as proving equivalences can be solved from left to right (or top to bottom), or vice versa. How do we support solving exercises in which a student can take steps at different positions?
– Proving the equivalence of two formulae requires heuristics. These heuristics support effective reasoning and the flexible application of solution strategies in these proofs. How do we formulate heuristics in our solution strategies for solving these kind of exercises? How 'good' are our solutions compared to expert solutions?
– Reuse and adaptivity play an important role in this domain: different teachers 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 logic exercises?

This paper describes the knowledge necessary to support solving propositional logic exercises in a learning environment, including solutions to the above aspects of solving propositional logic exercises.

Most existing systems for propositional logic do not have a student model; this paper calls such systems learning environments (LE). This paper focusses on the components necessary for providing feedback and feed forward in a propositional logic LE. However, we have also developed an LE on top of these compo-

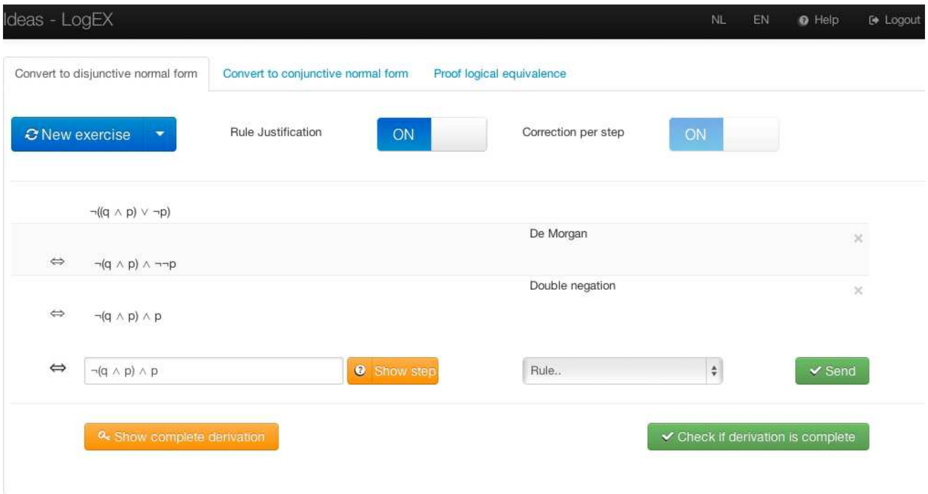

Figure 1: Screenshot of our learning environment for logic

nents [Lodder et al., 2006], see Figure 1.[1]

This paper is organised as follows. Section 2 gives an example of an interaction of a student with a (hypothetical) LE. Section 3 describes the characteristics of LEs for propositional logic, which Section 4 uses to compare existing LEs. We identify a number of aspects that have not been solved satisfactorily, and describe our approach to tutoring propositional logic in Section 5. Until Section 5 we describe a theoretical framework and look at related work. From section 5 on we present our own approach to tutoring propositional logic using so-called feedback services, and the implementation of our approach in the LogEx environment. Section 6 shows how this approach supports solving logic exercises for rewriting logic expressions to disjunctive or conjunctive normal form and for proving the equivalence of two logical formulae. We conclude with briefly describing the results of several small experiments we performed with LogEx.

2 Example interactions in an LE for propositional logic

This section 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

$$\neg((q \to p) \land p) \land q$$

[1] http://ideas.cs.uu.nl/logex/

into disjunctive normal form (DNF). The student might go through the following steps:

$$(\neg(q \rightarrow p) \wedge \neg p \wedge q \tag{1}$$

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

$$(\neg(q \rightarrow p) \wedge \neg p) \wedge q \tag{2}$$

The LE reports that this formula is equivalent to the previous formula, but it cannot determine which rule has been 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:

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

Now the LE recognises the rule applied (DeMorgan), and adds the formula to the derivation. Suppose the student does not know how to proceed here, and asks for a hint. The LE responds with: use Implication elimination. The student asks the LE to perform this step, which results in:

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

The student continues with:

$$(\neg\neg q \vee \neg p \vee \neg p) \wedge q \tag{3}$$

The LE reports that this step is not correct, and mentions that when applying DeMorgan's rule, a disjunction is transformed into a conjunction. 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:

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

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

$$\neg p \wedge q$$

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.

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

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

The LE places the right-hand side formula below the left-hand side formula, and the student has to fill in the steps in between. It is possible to enter steps top-down or bottom-up, or to mix the two directions. The student chooses to enter a bottom-up step and to rewrite

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

into:

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

If she does not know how to proceed, she can ask for a hint. The LE suggests to rewrite this last formula; a first hint for these kind of exercises will always refer to the direction of the proof. Now she can choose to perform this rewriting or she can ask for a second hint. This hint will suggest to use equivalence elimination.

She can continue to finish the exercise, but she can also ask the LE to provide a complete solution.

3 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 Section 4. 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 monitors the interactions between a student and a system when a student is solving a particular task. Important aspects of the inner loop are the analyses performed and the feedback provided. We distinguish feedback consisting of reactions of the system on steps performed by the students, and hints, next steps and complete solutions provided by the system. 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. For the interactions in the inner loop, some aspects are specific for LEs for logic.

3.1 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 2) and proving an equivalence (EQ).

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).

3.2 Interactions in the inner loop

In the inner loop of an LE, a student works on a particular task. 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 LEs require a student to provide both the name of the rule to apply, and 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).

3.3 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. The numbers refer to examples of these mistakes in Section 2.

- A syntactical mistake (1)
- 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 or 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. A student can 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 (2). A second possibility is the rewriting of a formula into a semantically different formula (3).
- 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.

We distinguish three categories of mistakes: syntactic errors, errors in applying a rule, and strategic errors. Narciss characterises classes of feedback depending on how information is presented to a student [Narciss, 2008]. When 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).

3.4 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). 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)?

3.5 Solutions

Some LEs offer worked-out examples (WO), or solutions to all exercises available in the tool (SOL).

3.6 Adaptability

Finally, we look at flexibility and adaptability. Can a teacher or a student change the set of rules or connectives (YES, NO)?

4 A comparison of tools for teaching logic

This section describes some LEs for logic using the characteristics from the previous section. We build upon a previous overview of tools for teaching logic by [Huertas, 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 LogEx: rewriting a formula in normal form and proving an equivalence using standard equivalences as rewrite rules. Quite a few tools for logic support learning natural deduction, which is out of scope for our comparison. We summarise our findings in Figure 2.

4.1 Rewriting a formula in normal form

Using Organon[2] [Dostálová and Lang, 2011, Dostálová and Lang, 2007], a student practices rewriting propositional formulae into DNF or CNF (conjunctive

[2] http://organon.kfi.zcu.cz/organon/

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

Figure 2: Comparison of logic tools and their characteristics

Type	NF: normal form; EQ: equivalence 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
Input	FO: input a formula; RU: input a rule name; RaF: input a rule name and 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; KCR: explanation why a step does not follow a desired strategy
Hint	NE: LE provides next step; GE: list of possible useful rules, subgoal, etc.; ***: not always available; FF1: feed forward only in one direction (top down)
Solution	WO: worked out demos
Adaptation	YES: users may adapt the rule set; NO: users cannot adapt the rule set

normal form). It automatically generates exercises, based on a set of schemas. 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 in (2). 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.

FMA contains exercises on rewriting propositional formulae to complete normal form: a DNF or CNF where each conjunct respectively disjunct contains all the occurring variables, possibly negated. [Prank, 2014]. A student highlights the subformula she wants to change. In input mode, she enters the changed subformula. The tool checks the syntax, and provides syntax error messages if necessary. In rule mode, a student chooses a rule, and FMA applies this rule to a subformula, or it gives an error message if it cannot apply it. In 2013, an analyser was added to FMA. The analyser analyses a complete solution, and provides error messages on steps where a student solution diverges from a solution obtained from a predefined strategy. For example, the analyser might give the feedback: “Distributivity used too early”.

Logicweb[3] 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 selects a rule, and the tool applies it. Thus a student can focus on the strategy to solve an exercise. The only mistake a student can make is choosing 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.

4.2 Proving equivalences

SetSails[4] [Zimmermann and Herding, 2010, Herding et al., 2010] offers two kinds of exercises: prove that two set-algebra expressions denote the same set, or prove

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

that two propositional logic formulae are equivalent. We only look at the last kind of exercises. SetSails contains a (small) set of predefined exercises, but a user can also 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. A student can also enter a formula, in case it is missing in the list of suggested formulae. Further feedback, such as corrections on the applied rules and hints, is given when a student asks the system to check a proof, which can be done at each step. 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 possibly needed, but not how to apply them, and the list of the rules needed is not complete. The system does not provide next steps or complete solutions. After entering an exercise, a user chooses rules from a predefined set or adds 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 student can work both forwards and backwards, but the tool does not give advice about these directions.

Logic Cafe[5] 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. If a student makes a mistake in a step, it is not accepted. In some cases Logic Cafe gives global feedback about a reason, for example that a justification should start with the number of the line on which the rule is applied, or that a justification contains an incorrect rule name. When a student asks for a hint, she gets a list of rules she has to apply. This kind of feed forward is only available for predefined exercises. A student can enter her own exercise. 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.

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

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

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 signals a mistake. 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.

4.3 A comparison

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

The kind and content of the feedback varies a lot, partly depending on the way a student works in the tool. Feedback on the rule-level consists of mentioning that a rule name is incorrect, or that a mistake has been made. SetSails gives a general form of the correct rule to be applied. FOL equivalence is the only tool that gives error-specific feedback. None of the other tools report common mistakes (KM). Logicweb gives feedback on the strategic level when a student uses a wrong distribution rule, and FMA indicates where a student solution diverges from a solution obtained from a predefined strategy.

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 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.

SetSails and Logic Cafe offer the possibility to prove equivalences while working in two directions. However, these tools do not offer hints on whether to perform a forward or a backward step, and it is not possible to receive a next backward step.

In SetSails a user can define her own set of rules. However, it does 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, supported 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 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 provides feed forward not only for a standard way to solve an exercise, but also for alternative ways. It also supports a student that uses both forward and backward steps in her proof.

The feedback provided in 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.

5 Feedback services

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. The expert knowledge module is 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, we use the term *domain reasoner* for this component [Goguadze, 2011]. We discuss how to construct a domain reasoner for propositional logic that has all characteristics introduced in Section 3.

A domain reasoner provides feedback services to an LE. We use a client-server style in which an LE uses stateless feedback services of the domain reasoner by sending JSON or XML requests over HTTP [Heeren and Jeuring, 2014]. We identify three categories of feedback services: 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 are used for many domains, including rewriting to DNF or CNF and proving two formulae equivalent.

5.1 Services for the outer loop

The feedback services supporting 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

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).

5.2 Services for the inner loop

There are two fundamental feedback services for the inner loop. The *diagnose* service generates feedback. It analyses a student step and detects various types of mistakes, such as syntactical mistakes, common misconceptions, strategic errors, etc. The *allfirsts* service calculates feed forward, in the form of a list of all possible next steps based on a (possibly non-deterministic) strategy.

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). In Section 5.5 we present rules for propositional logic.
- A *rewrite strategy* that specifies how an exercise can be solved stepwise by applying rules. Section 6 defines strategies for 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.

Explicitly representing rules and rewrite strategies improves adaptability and reuse of these components. We come back to the issue of adaptability in Section 6.2.

5.3 Alternative approaches

There are different ways to specify feedback or feed forward for logic exercises. Defining feedback separately for every exercise is very laborious, especially since solutions are often not unique. In this approach it is hard to also provide feedback or hints when a student deviates from the intended solution paths. One way to overcome this is to use a database with example solutions [Aleven et al., 2009]; an implementation of this idea for a logic tutor is described by Stamper. In this tutor, Deep Thought, complete solutions and intermediate steps are automatically derived using data mining techniques based on Markov decision processes. These solutions and steps are then hard coded. In this way, Deep Thought [Stamper et al., 2011] can provide a hint in 80% of the cases. Another advantage of using example solutions over using solution strategies, is that it is not always clear how to define such a strategy.

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 similar to the exercises in the database.

5.4 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[6]. In this section we describe how LogEx deals with the characteristics given in Figure 2. 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.

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.

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. In Section 6.2 we will sketch an approach to supporting adaptation.

5.5 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 [v.d. Vrie et al., 2009] (Fig. 3).

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

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

Figure 3: Rules for propositional logic

Variants can be found in other textbooks. For example, Burris defines equivalence in terms of implication [Burris, 1998], and Huth leaves out complement and true-false rules [Huth and Ryan, 2004].

Sometimes derivations get long when strictly adhering to a particular rule set. For this reason we implicitly allow associativity in our solution strategies, so that associativity does not need to be mentioned when it is applied together with another rule. This makes formulae easier to read, and reduces the possibility of syntax errors. Commutativity has to be applied explicitly; but we offer all commutative variants of the complement rules, the false and true rules, and absorption (Fig. 3). For example, rewriting $(q \wedge p) \vee p$ into p is accepted as an application of AbsorpOr. Also a variant of the distribution rule is accepted: students may rewrite $(p \wedge q) \vee r$ in $(p \vee r) \wedge (q \vee r)$ using DistrOr, and the same holds for DistrAnd.

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 $\neg(p \vee q \vee r \vee s)$ into $\neg p \wedge \neg(q \vee r) \wedge \neg s$ as an application of a generalised DeMorgan rule. These generalised rules are more or less implied by allowing implicit associativity.

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 (almost 100) buggy rules we use is based on the experience of teachers, and includes rules obtained from analysing the log files of the diagnose service.

5.6 A strategy language

Although some textbooks give strict procedures for converting a formula into normal form [Huth and Ryan, 2004], most books only give a general description [v.d. Vrie 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, 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]. 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 rewriting strategies we use the strategy language developed by Heeren et al. [Heeren et al., 2010]. This language is used to describe strategies in a broad range of domains. The meaning of the word 'strategy' here slightly deviates from its usual meaning. A rewriting strategy is any combination of steps, which could be used to solve a procedural problem, but which can also be a more or less random combination of steps. 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 rewriting strategy, and in the (inductive) definition of the language, they are considered rewriting strategies by themselves. We use combinators to combine two rewriting strategies, so a rewriting strategy is a logical rule r, or, if s and t are rewriting strategies then:

- $s \mathbin{<\!\star\!>} t$ is the rewriting strategy that consists of s followed by t
- $s \mathbin{<|>} t$ is the rewriting strategy that offers a choice between s and t
- $s \mathbin{>|>} t$ is the rewriting strategy that offers a choice, but prefers s
- $s \triangleright t$ is a left-biased choice: t is only used if s is not applicable
- *repeat* s repeats the rewriting 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 $(p \vee s) \wedge (q \vee r) \wedge (u \vee v)$. To bring this formula into DNF we apply distribution. We can apply distribution top-down (to the first conjunct in $(p \vee s) \wedge ((q \vee r) \wedge (u \vee v))$) 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. We implement this using the preference operator.

6 Strategies for propositional logic exercises

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

6.1 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. If a strategy gives the hint to apply DeMorgan, a student might not understand why this hint is given. 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 an 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, based on the strategy described in the Open University textbook [v.d. Vrie et al., 2009], 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 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 to remove double negations before applying 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* s: check whether a formula is a disjunction and apply s to one of the disjuncts, otherwise apply s 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 disjunction simplification rules that we try to apply at top-level: 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 rewriting 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 by applying *orRulesS*, or the dual strategy *andRulesS*, or one of the three rules DoubleNeg, NotFalse, or NotTrue
- *eliminateImplEquivS*: remove implications by applying DefImpl or equivalences by applying DefEquiv
- *deMorganS*: use DeMorganOr or DeMorganAnd to move negations down

The rewriting strategy *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 rewriting strategy always tries to simplify a formula first. After simplifying it removes implications or equivalences, simplifying in between, and then applies DeMorgan.

After obtaining an NNF we distribute conjunctions over disjunctions to obtain a DNF. This is achieved by the rewriting strategy *distrAndS*, which applies DistrAnd or GeneralDistrAnd, preferring the second rule.

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

$$\begin{aligned}&dnfStrategy = repeat\\&\quad (orRulesS <\!\!|\!\!> somewhereOr\ (nnfStep \triangleright distrAndS))\end{aligned}$$

Using *repeat* at the beginning of the rewriting strategy ensures that at each step we apply the complete rewriting strategy again, and hence simplify whenever possible.

In some cases *dnfStrategy* does not further simplify a formula. It simplifies $p \vee (q \wedge \neg q \wedge r)$, but not $p \vee (q \wedge r \wedge \neg q)$, because q and $\neg q$ are not adjacent formulae in the conjunct. We introduce a rewriting strategy *groupLiterals*, 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 substrategy in our rewriting strategy for proving the equivalence between two formulae later in this section.

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

A third rewriting strategy, *deMorganNot*, checks whether an application of DeMorgan leads to the possibility to simplify.

The improved definition of *nnfStep* looks as follows:

$$\begin{aligned}nnfStep = {}& simplifyStep\\&\triangleright (groupLiterals >\!\!|\!\!> distrNot >\!\!|\!\!> deMorganNot)\\&\triangleright (eliminateImplEquivS >\!\!|\!\!> deMorganS)\end{aligned}$$

The rewriting strategy ends if there are no steps left that can be applied. Possibly the rewriting strategy reaches a normal form before it ends. The finished service is used to check whether a formula is indeed in normal form. We allow a student to simplify a formula even if a normal form is reached.

6.2 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 rewriting strategies or rule sets. Section 6.1 shows how to define variants of the DNF strategy.

Another way in which a teacher can adapt feedback services is by changing the rule set. Our DNF strategy is structured in a way that makes it easy to adapt the rewriting strategy for users who apply more, fewer, or different rules. Our basic strategy contains five substrategies that can be considered as sets of rules: *orRulesS*, *simplifyStep*, *eliminateImplEquivS*, *deMorganS* and *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 rewriting strategy, a teacher can change the content of any of these five sets. To guarantee that a modified strategy still returns a normal form, the modification has to satisfy certain criteria, described in [Lodder et al., 2015a].

6.3 A rewriting strategy for proving two formulae equivalent

This subsection discusses a rewriting 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 rewriting 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.

The first heuristic we use in our rewriting 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 rewriting 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

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

takes only three steps: two applications of commutativity and one of idempotency. The rewriting 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 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.

We use inverse distribution to factor out common literals in the disjuncts of ϕ and ψ. 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.

A complete normal form is seldom necessary. We considerably shorten proofs by using splitting rules of the first heuristic during normalization, together with applications of the absorption rule. The subformula we choose to extend to normal form influences the length of the proof too. For example, we do not choose subformulae that occur on both sides.

To evaluate our rewriting strategy we asked 4 human experts (theoretical computer scientists and logicians) to solve a set of 6 'independent' exercises[7]. The expert solutions to two exercises were almost equal to the solutions of LogEx, up to a few differences in the order of the applied rules. One expert found a quicker solution to one exercise. In another exercise LogEx found a short cut which was overlooked by the experts. The experts found shorter solutions to the other three exercises. LogEx constructs a proof via a DNF while in some cases a proof via CNF is shorter, and using inverse distribution more often than in our strategy also helps. From the 187 steps taken by the experts in their solutions to the exercises 169 (= 90%) were recognised as part of the strategy by LogEx.

7 Experimental results

Since the first version of LogEx we performed several small-scale experiments with students [Lodder et al., 2008]. In this section we describe the results of three experiments carried out in December 2014, December 2015 and February 2016.

In December 2014 we organised a pilot study with the LogEx learning environment [Lodder et al., 2015b]. In this pilot study we used pre- and post-tests together with an analysis of the log files to answer the question whether using

[7] `https://en.wikibooks.org/wiki/Logic_for_Computer_Scientists/Propositional_Logic/Equivalence_and_Normal_Forms##Problems`

	pre-test	post-test
Completion	0,54	1,0
Mistakes	0,84	0,7

Figure 4: Results pre- and post-test December 2015

LogEx helps students to reach the following learning goals: after practicing with the LE, a student can

- recognise applicable rules
- apply rules correctly
- rewrite a formula in normal form
- prove the equivalence of two formulae using standard equivalences
- demonstrate strategic insight in how to rewrite a formula in normal form or prove an equivalence in an efficient way.

The number of the students participating in this experiment (5) was too low to draw firm conclusions. However, the results of the study indicate that indeed LogEx helps students to reach the learning goals, except for the last goal. We hypothesise that the reason that students do not improve in efficiency is because LogEx does not provide strategic feedback.

In December 2015 we repeated the pilot evaluation with a group of 8 students. Again we used pre- and post-tests and analysed the loggings. We graded the tests in two ways: we measured the completion (percentage of completed exercises) and the number of mistakes per completed exercise. The results are given in Figure 4. Although we constructed pre- and post-tests such that the difficulty of both tests is comparable, we have no objective criterion to measure difference in difficulty. Still, the results on the pre- and post-tests indicate that indeed students learn to apply rules correctly, and to rewrite a formula in normal form or prove the equivalence between two formulae. The analysis of the loggings also showed that while working with LogEx, students become more skilled in applying the appropriate rules. In contrast to the results of the 2014 test, here we find that students do learn to work more efficiently. We measure efficiency by dividing the number of steps of a completed exercise by the number of steps of the example solution generated by LogEx. We only took exercises that were completed by at least half of the students into account. Figure 5 shows the results. The exercises are presented in the order in which they were completed: students started with the first exercise on rewriting a formula to DNF, and ended with the fifth exercise on proving an equivalence. Note that the first exercise on deriving a CNF (*cnf0*) was solved much less efficiently than the preceding DNF exercises: students had to find out how to adjust their solution strategy. Exercise

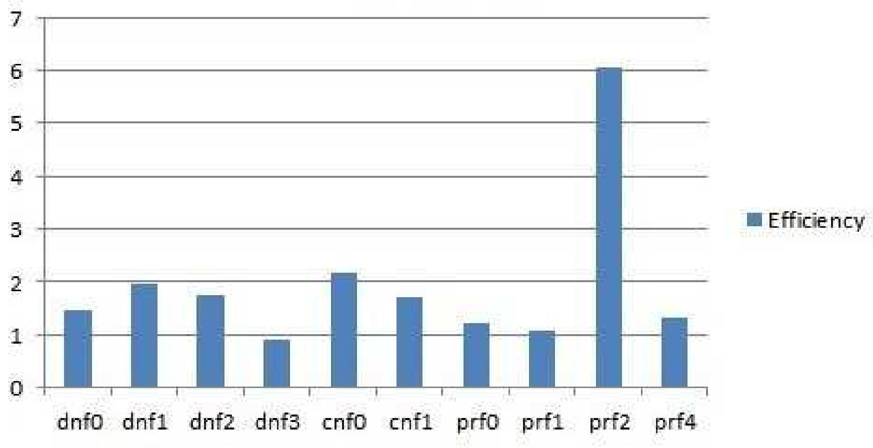

Figure 5: Logging analysis for December 2015 (measuring efficiency)

prf2 has a short solution. If a student does not find this solution, the alternative is long, which explains the high value for this exercise.

In February 2016 we performed a small experiment with students from the Open University of the Netherlands. Our main research question for this experiment was:

- to what extent do feedback and feed forward contribute to the learning of the students?

The Open University is a university for distance education with students living all over the Netherlands. Hence the experiment was performed in the context of distance learning. After a short instruction via an on-line learning environment, students worked on a 20-minute pen and paper pre-test. After sending in the answers, they received access to LogEx, in which they practiced with DNF, CNF and proof exercises. The experiment was concluded with a 20 minute pen and paper post-test. 15 students participated, but only 12 handed in both pre- and post-test. In our analysis we restrict ourselves to these 12 students. They were divided into two groups of 6 students. One group (group A) could use the full functionality of LogEx, the second group (group B) received no hints and next steps. Students in the second group got only feedback after finishing their exercise. Students in both groups could ask for a worked out solution, for example to compare it with their own solution. Both pre- and post-test consisted of three exercises. As in the December 2015 experiment, we graded the tests in two ways: we measured the completion and the number of mistakes per completed exercise. We have no objective criterion for determining the difficulty of both tests, hence we cannot measure overall learning effects, but we can compare the

	Group A	Group B
Completion pre-test	0,44	0,47
Completion post-test	0,60	0, 53
Mistakes pre-test	1,0	1,2
Mistakes post-test	1,1	1,5

Figure 6: Results pre- and post-test, Open University 2016

results of both groups. As shown in Figure 6, both groups made the same amount of mistakes in the pre-test. In the post-test both made more mistakes (maybe due to the difficulty of the post-test or tiredness), but group A made fewer mistakes than group B. Completion of the pre-test was higher in group B than in group A, in the post-test these results were reversed. The results indicate that indeed the presence of immediate feedback and feed forward do increase learning. We also analysed the loggings. This analysis shows that students in group A spent more time working in LogEx than students in group B (on average 57 min. versus 41 min.) The average number of exercises that students worked on in LogEx is for both groups more or less the same: (8,3 versus 8,8), but in group B the deviation was greater: one student worked on only four exercises, another only five. If we omit the results of these two students from the pre- and post-test, the results on completion for group B are somewhat better than those of group A. This experiment suggests that the main reason for the difference in performance between the two groups is a motivational one: students practice more when they get feedback and feed forward, and hence their results are better. This explanation is confirmed by the remark of a student in group B who complained that she could not correct her mistakes and hence "did not learn anything".

8 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, or enters her own exercise.

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 logical symbols.

We have performed 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 more experiments in the academic year 2016-2017, 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 environment 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. Marco Huijben en Wouter Tromp developed a useful tool for analysing the loggings of LogEx. Expert solutions were provided by Nikè van Vugt, Hans van Ditmarsch, Fer-Jan de Vries and José Martín Castro-Manzano. We thank our anonymous referees for their constructive comments.

References

[Aleven et al., 2009] 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.

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

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

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

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

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

[Dostálová and Lang, 2011] 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, 2011] 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 et al., 2013] Grivokostopoulou, F., Perikos, I., and Hatzilygeroudis, I. (2013). An intelligent tutoring system for teaching fol equivalence. In *AIED Workshops*.

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

[Heeren and Jeuring, 2014] 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 et al., 2010] Heeren, B., Jeuring, J., and Gerdes, A. (2010). Specifying rewrite strategies for interactive exercises. *Mathematics in Computer Science*, 3(3):349–370.

[Herding et al., 2010] 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, 2011] 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, 2008] Hurley, P. (2008). *A Concise Introduction to Logic*. Cengage Learning.

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

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

[Lodder and Heeren, 2011] 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 et al., 2015a] Lodder, J., Heeren, B., and Jeuring, J. (2015a). A domain reasoner for propositional logic. Technical Report UU-CS-2015-021, Department of Information and Computing Sciences, Utrecht University.

[Lodder et al., 2015b] Lodder, J., Heeren, B., and Jeuring, J. (2015b). A pilot study of the use of logex, lessons learned. *CoRR*, abs/1507.03671. Proceedings of the Fourth International Conference on Tools for Teaching Logic (TTL2015).

[Lodder et al., 2006] 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 et al., 2008] 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, 2008] 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, 1990] Nwana, H. S. (1990). Intelligent tutoring systems: an overview. *Artificial Intelligence Review*, 4(4):251–277.

[Prank, 2014] Prank, R. (2014). A tool for evaluating solution economy of algebraic transformations. *Journal of Symbolic Computation*, 61:100–115.

[Schoenfeld, 1987] 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 et al., 2011] Stamper, J. C., Eagle, M., Barnes, T., and Croy, M. (2011). 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.

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

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

[v.d. Vrie et al., 2009] v.d. Vrie et al., E. M. (2009). *Discrete wiskunde A, Lecture notes (in Dutch)*. Open Universiteit Nederland.

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