A Teaching Tool for Proving Equivalences between Logical Formulae

Josje Lodder and Bastiaan Heeren

School of Computer Science, Open Universiteit Nederland
P.O.Box 2960, 6401 DL Heerlen, The Netherlands
{josje.lodder, bastiaan.heeren}@ou.nl

Abstract. In this paper we describe a teaching tool for proving equivalences between propositional logic formulae, using rewrite rules such as De Morgan's laws and double negation. This tool is based on an earlier tool for rewriting logical formulae into disjunctive normal form (DNF). Both tools make use of a rewrite strategy, which specifies how an exercise can be solved stepwise. Different types of feedback can be calculated automatically from such a strategy specification. We describe a strategy for constructing expert-like equivalence proofs, and present two techniques for improving the proofs that are generated by the strategy.

Key words: propositional logic, equivalences, e-learning, feedback

1 Introduction

The construction of proofs is an important topic in logic courses. Several tools have been developed in the last decades to help students in acquiring proving skills [3, 9, 10]. Most often, natural deduction is used as a proof system, but also axiom systems are used. When students have to apply logic, they also have to simplify and rewrite logical formulae, and this takes some practice. For instance, a computer science student should be able to simplify a database query, or recognize that two database queries are equivalent. A typical example of applying logic is to simplify the following SQL query (using fewer negations):

```
SELECT s.name
FROM   Students s
WHERE  NOT (NOT (s.subject = math) OR s.startdate = 2010)
         AND s.grade >= 8)
```

Other examples of rewriting logical formulae are the simplification of conditional expressions found in most programming languages, specifying and reasoning with business rules, and turning logical propositions into Prolog clauses.

At the Open Universiteit Nederland, we have been working on several interactive exercise assistants, including a tool to train students in transforming a propositional formula into disjunctive normal form (DNF) [7]. These tools are based on a strategy language [6], in which rewrite strategies for solving exercises can be expressed (e.g., converting a formula into DNF). From such a strategy specification, different types of feedback can be calculated automatically, such as providing hints on how to continue, recognizing an intermediate step submitted by the student, and generating worked-out solutions.

We recognize the importance for computer science students to be able to manipulate logical formulae using rewrite rules. In this paper we discuss how our logic tool can be extended with exercises in proving the equivalence of propositional logic formulae. This paper makes the following contributions:

– We describe a strategy for constructing expert-like equivalence proofs (i.e., proofs that appear non-mechanical). The strategy is illustrated by a number of example proofs that are generated by the strategy. Our approach is general, and therefore applicable to constructing proofs in other areas.
– Our claims are supported by a prototype implementation of the strategy for constructing proofs. We highlight the changes that are needed to the tool for rewriting formulae into DNF.

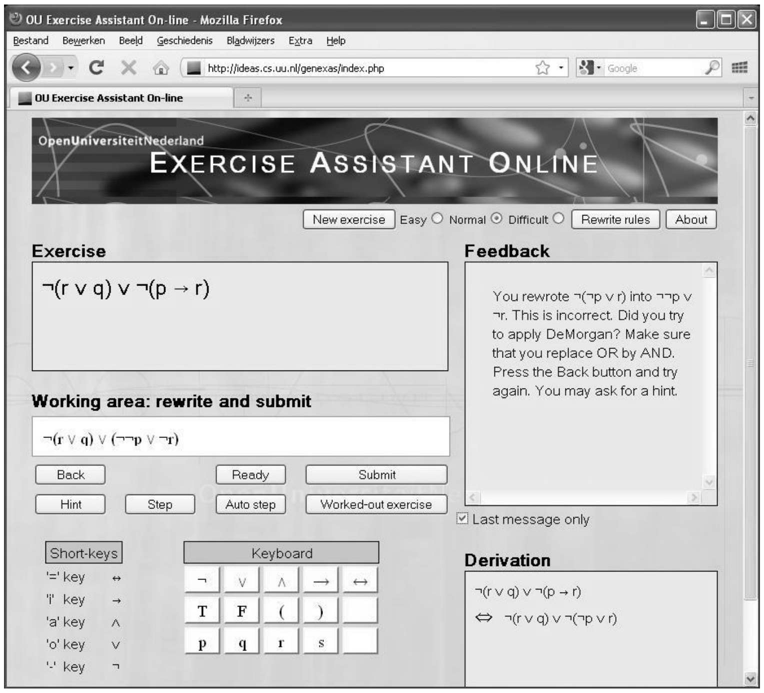
The remainder of this paper is structured as follows. We first introduce our web-based exercise assistant for rewriting logical formulae into DNF in Section 2, where we briefly discuss our approach for developing interactive exercise assistants based on rewrite strategies. Section 3 then presents a strategy for proving equivalences between formulae and two techniques for improving the proofs. Examples of proofs that are generated by our strategy are given in Section 4. The last two sections discuss related work and draw conclusions.

2 Rewriting formulae into DNF

We start with an overview of our tool for rewriting arbitrary formulae into DNF. Most of its functionality can be reused for exercises in proving equivalences. Figure 1 shows a screenshot of the interactive exercise assistant[1]. Students have to rewrite a formula into normal form, using a fixed set of allowed rewrite rules. At each point, hints are available about the next step, or an example solution can be shown. More importantly, students also receive feedback when they submit intermediate answers. The tool identifies the rewrite rule that was used, or it tries to recognize a common misconception (also known as *buggy rule*) in case the answer is incorrect. For example, Figure 1 contains a feedback message about the incorrect application of De Morgan (i.e., the buggy rule $\neg(\phi \vee \psi) \Rightarrow \neg\phi \vee \neg\psi$).

Feedback is derived automatically from a strategy specification. Such a strategy describes the order in which rewrite rules have to be applied to solve a particular type of exercise. Strategies for reaching DNF have been reported in [6]. When a student deviates from the strategy, this can be detected and reported by the tool. We currently allow these deviations, also because they may prove to be clever shortcuts. For practical reasons, associativity of conjunction and disjunction is silently performed by the tool. On the contrary, commutativity requires an explicit step by the student.

[1] The DNF tool is available at: `http://ideas.cs.uu.nl/genexas/`


OU Exercise Assistant On-line - Mozilla Firefox
Bestand Bewerken Beeld Geschiedenis Bladwijzers Extra Help
http://ideas.cs.uu.nl/genexas/index.php
OU Exercise Assistant On-line
OpenUniversiteitNederland
EXERCISE ASSISTANT ONLINE
New exercise
Rewrite rules
About
Back
Ready
Submit
Hint
Step
Auto step
Worked-out exercise
Short-keys
'=' key ↔
'i' key →
'a' key ∧
'o' key ∨
'-' key ¬
Keyboard
¬ ∨ ∧ → ↔
T F ( )
p q r s

Easy ○ [✓ Normal] Difficult ○

Exercise

¬(r ∨ q) ∨ ¬(p → r)

Working area: rewrite and submit

¬(r ∨ q) ∨ (¬¬p ∨ ¬r)

Feedback

You rewrote ¬(¬p ∨ r) into ¬¬p ∨ ¬r. This is incorrect. Did you try to apply DeMorgan? Make sure that you replace OR by AND. Press the Back button and try again. You may ask for a hint.

[✓ Last message only]

Derivation

¬(r ∨ q) ∨ ¬(p → r)

⇔ ¬(r ∨ q) ∨ ¬(¬p ∨ r)

Fig. 1. Screenshot of the Exercise Assistant for rewriting formulae into DNF

The tool has been tested with bachelor's students in a course on discrete mathematics. The results of the test were very promising [8]. The tool helped the students in understanding logical formulae and improving their rewriting skills.

3 Proving equivalences between formulae

We will now discuss the design of a tool for practicing the construction of equivalence proofs: the DNF tool is a good starting point for this. The tools operate on the same type of formula, and the same collection of rewrite rules (e.g., commutativity, absorption, and the buggy rules) can be used. Because we base our approach on rewrite strategies, the full machinery for analyzing steps, recognizing common misconceptions, generating hints, and providing feedback is readily available. Changes to the following components are required:

– Modifications to the user interface are needed. Students should be allowed to perform a forward step or a backward step, at all times. Suppose a proof is asked for the equivalence $\phi \Leftrightarrow \psi$. Then ϕ can be rewritten to ϕ' (forward step, giving the task of proving $\phi' \Leftrightarrow \psi$), or ψ to ψ' (backward step, resulting in $\phi \Leftrightarrow \psi'$). The proof is completed when ϕ and ψ have been rewritten to the same formula. The user interface should accommodate for steps in both directions.

– A strategy is needed for proving equivalences in an expert-like way. This is the most significant extension to our tool. The strategy for reaching DNF will play a prominent role in the strategy specification.
– Lastly, the creation of exercises is different. In the DNF tool, we use a random formula generator. In the case of proving equivalences, we restrict ourselves to a fixed set of exercises, such as the ones in Section 4.

3.1 A strategy for proving equivalences

The general idea for proving the equivalence $\phi \Leftrightarrow \psi$ is rather straightforward: rewrite both ϕ and ψ to DNF, and then rewrite these normal forms to equal forms using a given set of rewrite rules. This approach results in proofs that appear mechanical. For example, consider $\neg(p \wedge (q \rightarrow r)) \Leftrightarrow \neg((q \rightarrow r) \wedge p)$. Suggesting to eliminate the implication, or to apply De Morgan's law at one of the sides, is not very reasonable. Therefore, we make several refinements to the strategy. The strategy consists of two parts (explained in sections 3.2 and 3.3):

– Part 1: rewrite ϕ and ψ into normal forms, but search for parts that do not have to be rewritten at each step.
– Part 2: to finish the proof, rewrite the normal forms into equal forms. This is only needed when the normal forms are different.

The strategy is used by the tool to give hints and to provide sample solutions, and these are available at each moment. The strategy alternates between making forward and backward steps, just like a student. In fact, it rewrites a pair (ϕ, ψ) by applying a rule either to ϕ or ψ, leaving the other formula unchanged. Whenever a hint is asked, the next step is computed from the strategy. Although it might be feasible to calculate the *shortest proof* for simple exercises (i.e., calculate the proof once, before a student starts with the exercise), such an approach becomes impractical as soon as students are allowed to depart from this proof. In such a scenario, a shortest proof has to be calculated repeatedly. It is unclear how this can be done efficiently.

3.2 Towards disjunctive normal form (part 1)

The first part reuses the strategy for rewriting formulae into DNF (found in [6]). Before each step, we first try to perform the following simplifications:

Proof decomposition. Assume we have to prove $\phi \Leftrightarrow \psi$. If ϕ and ψ are both conjunctions, say $\phi = \phi_1 \wedge \phi_2$ and $\psi = \psi_1 \wedge \psi_2$, then check whether $\phi_1 \Leftrightarrow \psi_1$ and $\phi_2 \Leftrightarrow \psi_2$ holds, or $\phi_1 \Leftrightarrow \psi_2$ and $\phi_2 \Leftrightarrow \psi_1$. If so, decompose the proof into two subproofs. Truth tables are used to check the equivalences. In the latter case, use commutativity to exchange ψ_2 and ψ_1. Note that this decomposition is only a mental step, since in the end we are interested in constructing a linearized proof. The effect of this step is that the conjunction will not be rewritten, and that rewriting towards DNF takes place on ϕ_1, ϕ_2, ψ_1, and ψ_2. Follow a similar procedure when ϕ and ψ are both disjunctions, implications, equivalences, or negations. The subproofs can again be decomposed.

Common subformulas. Check whether ϕ and ψ share the same subformula χ (not a proposition letter). Substitute χ in ϕ and ψ by a new proposition letter, and check whether the resulting formulae are still equivalent. If this is the case, treat χ as an atom. No rewriting takes place on such an atom. This substitution is not visible for the student, except that these subformulas will not be transformed by the strategy. Students are still allowed to rewrite it.

3.3 Towards equal forms (part 2)

In many cases the normal forms of ϕ and ψ will be equal up to associativity and commutativity, and a simple reordering (applications of the commutative law) completes the proof. Sometimes the differences are more fundamental. In those cases the following steps are performed:

(a) If the set of proposition letters that occur in the normal forms of ϕ and ψ are different, eliminate the letters that occur in only one of the normal forms (by applying simplification rules).
(b) If ϕ contains (up to commutativity) a subformula of the form $p \vee (\neg p \wedge \chi)$ and ψ contains $p \vee \chi$, rewrite $p \vee (\neg p \wedge \chi)$ into $p \vee \chi$, using distribution and true/false rules.
(c) If no other simplifications are possible, extend the normal forms by using true/false rules and distribution to a complete normal form. Each conjunction corresponds to a row of the truth table. A complete normal form is unique (up to commutativity), and thus guarantees the completion of the equivalence proof.

The main purpose of the tool is to improve the skills in applying rewrite rules, and to learn how to recognize simple equivalences. We do not want students to memorize the rewrite strategy, nor do we make it explicit. Its function is only to provide sensible hints and worked-out examples.

4 Examples

We have tested the rewrite strategy for proving equivalences on a set of exercises. We used exercises from textbooks [11, 4], and added some exercises to test special cases. We discuss some of the proofs generated by the strategy. The order in which the rules are applied is indicated by numbering the steps.

Example 1. Prove $\neg(p \vee \neg(p \vee \neg q)) \Leftrightarrow \neg(p \vee q)$.

$$\begin{array}{cll}
 & \neg(p \vee \neg(p \vee \neg q)) & \\
\overset{1}{\Longleftrightarrow} & \neg(p \vee (\neg p \wedge \neg\neg q)) & \text{De Morgan} \\
\overset{2}{\Longleftrightarrow} & \neg(p \vee (\neg p \wedge q)) & \text{double negation} \\
\overset{3}{\Longleftrightarrow} & \neg((p \vee \neg p) \wedge (p \vee q)) & \text{distribution} \\
\overset{4}{\Longleftrightarrow} & \neg(T \wedge (p \vee q)) & \text{complement} \\
\overset{5}{\Longleftrightarrow} & \neg(p \vee q) & \text{true/false rule}
\end{array}$$

Both starting formulae are negations: because of *proof decomposition*, the top-level negation is kept throughout the proof. The strategy proceeds by rewriting the subformula $p \vee \neg(p \vee \neg q)$ into DNF, resulting in two forward steps (1–2). The right-hand side was already in DNF. The second part of the strategy then rewrites $p \vee (\neg p \wedge q)$ into $p \vee q$. In this case, all proof steps are forward. Note that swapping the starting propositions would give a completely backward proof.

Example 2. Prove $\neg((p \to q) \to (p \wedge q)) \Leftrightarrow (p \to q) \wedge (\neg p \vee \neg q)$.

$$\begin{array}{lll}
 & \neg((p \to q) \to (p \wedge q)) & \\
\overset{1}{\Longleftrightarrow} & & \text{implication elimination} \\
 & \neg(\neg(p \to q) \vee (p \wedge q)) & \\
\overset{2}{\Longleftrightarrow} & & \text{De Morgan} \\
 & \neg\neg(p \to q) \wedge \neg(p \wedge q) & \\
\overset{3}{\Longleftrightarrow} & & \text{double negation} \\
 & (p \to q) \wedge \neg(p \wedge q) & \\
\overset{4}{\Longleftrightarrow} & & \text{De Morgan} \\
 & (p \to q) \wedge (\neg p \vee \neg q) &
\end{array}$$

Both propositions have the subformula $p \to q$, and replacing this formula by a new proposition letter (say a) results in propositions that are still equivalent. Hence, the strategy constructs a proof for $\neg(a \to (p \wedge q)) \Leftrightarrow a \wedge (\neg p \vee \neg q)$. More specifically, the common subformula $p \to q$ should not be rewritten (by the strategy). All steps are forward, and belong to part 1.

Example 3. Prove $p \to (q \to r) \Leftrightarrow (p \to q) \to (p \to r)$.

$$\begin{array}{lll}
 & p \to (q \to r) & \\
\overset{1}{\Longleftrightarrow} & & \text{implication elimination} \\
 & \neg p \vee (q \to r) & \\
\overset{2}{\Longleftrightarrow} & & \text{implication elimination} \\
 & \neg p \vee \neg q \vee r & \\
\hline
 & \neg p \vee \neg q \vee r & \\
\overset{11}{\Longleftrightarrow} & & \text{commutativity} \\
 & \neg q \vee \neg p \vee r & \\
\overset{10}{\Longleftrightarrow} & & \text{true/false rule} \\
 & (T \wedge (\neg q \vee \neg p)) \vee r & \\
\overset{9}{\Longleftrightarrow} & & \text{complement} \\
 & ((p \vee \neg p) \wedge (\neg q \vee \neg p)) \vee r & \\
\overset{8}{\Longleftrightarrow} & & \text{distributivity} \\
 & (p \wedge \neg q) \vee \neg p \vee r & \\
\overset{7}{\Longleftrightarrow} & & \text{double negation} \\
 & (\neg\neg p \wedge \neg q) \vee \neg p \vee r & \\
\overset{6}{\Longleftrightarrow} & & \text{De Morgan} \\
 & \neg(\neg p \vee q) \vee \neg p \vee r & \\
\overset{5}{\Longleftrightarrow} & & \text{implication elimination} \\
 & \neg(\neg p \vee q) \vee (p \to r) & \\
\overset{4}{\Longleftrightarrow} & & \text{implication elimination} \\
 & \neg(p \to q) \vee (p \to r) & \\
\overset{3}{\Longleftrightarrow} & & \text{implication elimination} \\
 & (p \to q) \to (p \to r) &
\end{array}$$

In this particular example, both forward and backward steps are used. The first five steps consist of eliminating the implications. Note that the order of these five elimination steps (two forward steps, and three backward steps) is not fixed by the strategy. In this paper we only show the default order. After 5 steps we have $\neg p \vee \neg q \vee r$ and $\neg(\neg p \vee q) \vee \neg p \vee r$. From this point on, the proof is decomposed into $\neg p \vee \neg q \Leftrightarrow \neg(\neg p \vee q) \vee \neg p$ and $r \Leftrightarrow r$. The latter holds trivially. The remaining steps focus on the former equivalence. After 7 steps we have $\neg p \vee \neg q$ and $(p \wedge \neg q) \vee \neg p$, which are both in DNF. From here on, part 2 of the strategy takes over and completes the proof (steps 8–11).

5 Related work

A nice overview of the differences between theorem provers, proof checkers, proof assistants, and proof editors on the one hand, and tutorial systems on the other hand, is given by Lukins et al. [9]. In this section we restrict ourselves to other tutorial systems, but we also compare our tool with a computer algebra system.

There are several tools for teaching how to prove theorems in propositional logic, but most teach natural deduction. These tools contain strategies for proving propositions, and use these strategies to provide hints or worked-out examples (e.g. Fitch [2], AProS [10], and Pandora [3]). There are also tools for rewriting exercises in DNF and CNF, such as Organon [5]. Organon is a web tutor for basic logic courses, and is used at a Czech university. A very simple tool for checking equivalences is the Equivalency Checker [1] from Texas A&M University: students can enter two formulae, and the tool checks equivalence (yes/no). DC Proof 1.2 (`http://www.dcproof.com/`) allows students to prove equivalences between formulas using a mix of rewrite rules (De Morgan, double negation, implication and equivalence elimination), and a natural deduction style of reasoning. The student has to choose a rule, which is executed by the tool. Predefined hints and worked-out solutions are only available for a fixed set of exercises. The tool does not contain a strategy to provide help.

The propositional theorem prover of the computer algebra system Yacas (`http://yacas.sourceforge.net/`) uses rewrite rules to simplify a negation of a theorem into false. Since Yacas is not meant for teaching logic, there is no need for a sophisticated strategy. Because Yacas is not a specialized theorem prover, this simple strategy is fast enough. In general, theorem provers are designed to find proofs efficiently, and they typically do not use rewrite rules. As far as we know, the implementation of an expert-like strategy for solving equivalences with rewrite rules is new.

6 Conclusions

We have presented a strategy for proving equivalences between logical formulae. This strategy is based on rewriting both formulae into DNF, with two exceptions: decompose the proof and keep common subformulae, whenever possible. This makes the generated proofs shorter. Afterwards, a strategy for rewriting the normal forms into equal forms is used to complete the proof. The strategy that generates the proofs has been implemented, and we have described how the web-based exercise assistant should be changed to support interactive exercises on the construction of equivalence proofs.

At this moment, the proof generator works as a stand alone application. We intend to integrate it in our existing web-based tool. After that we can test the tool with students: we hope to perform a first test during spring 2011. The DNF tool gives feedback when a student takes a step that deviates from the standard strategy. Since it is not our goal to teach the underlying strategy for proving equivalences, this kind of feedback is no longer necessary. However, we could compare the lengths of the student solution with the length of the generated proof. If the student needs more steps, we could add a message explaining that a shorter solution exists. Such a message could even be reported as soon as the (possibly incomplete) solution exceeds the expected number of steps. The tool could suggest to go back to the point where the detour started. Proofs that turn out to be shorter than the generated proof are of special interest to us, since they may suggest short-cuts that can be added to the strategy.

The approach followed is not restricted to proofs for logical formulae. Another direction of future research is to apply our ideas to different domains, such as equivalence proofs for relation algebra or set algebra.

Acknowledgements. The authors wish to thank Johan Jeuring, Alex Gerdes, Sylvia Stuurman, and Harrie Passier for their contributions to the exercise assistant. We thank the anonymous reviewers for their constructive comments. Discussions with Daniel Herding about searching proofs for set algebra are gratefully acknowledged.

References

1. C. Allen and C. Menzel. The Logic Machine: Equivalency Checker, 2006. `http://logic.tamu.edu/cgi-bin/equivalency.pl`.
2. J. Barwise and J. Etchemendy. *Language, Proof and Logic (Book & CD-ROM)*. Center for the Study of Language and Information, 1st edition, April 2002.
3. K. Broda, J. Ma, G. Sinnadurai, and A. Summers. Pandora: A Reasoning Toolbox using Natural Deduction Style. *Logic journal of the IGPL*, 15(4):293–304, 2007. `http://www.doc.ic.ac.uk/pandora/runpandora.html`.
4. S.N. Burris. *Logic for Mathematics and Computer Science*. Pearson Education, 1998.
5. L. Dostálová and J. Lang. Organon - the web tutor for basic logic courses. *Logic journal of the IGPL*, 15(4):305–311, August 2007.
6. B. Heeren, J. Jeuring, and A. Gerdes. Specifying rewrite strategies for interactive exercises. *Mathematics in Computer Science*, 3(3):349–370, 2010.
7. J. Lodder, J. Jeuring, and H. Passier. An interactive tool for manipulating logical formulae. In M. Manzano, B. Pérez Lancho, and A. Gil, editors, *Proceedings of the Second International Congress on Tools for Teaching Logic*, 2006.
8. J. Lodder, H. Passier, and S. Stuurman. Using ideas in teaching logic, lessons learned. *International Conference on Computer Science and Software Engineering*, 5:553–556, 2008.
9. S. Lukins, A. Levicki, and J. Burg. A tutorial program for propositional logic with human/computer interactive learning. *SIGCSE Bull.*, 34(1):381–385, February 2002.
10. W. Sieg. The AProS project: Strategic thinking & computational logic. *Logic journal of the IGPL*, 15(4):359–368, August 2007.
11. E. Thijsse. *Logica in de praktijk (in Dutch)*. Academic Service, April 2000.