MASTER'S THESIS

Logical shortcuts

Heuristic steps in logic tutoring systems

Steins, R.J. (Ruben)

Award date:
2020

Ir. R.(Ruben) J.Steins

Student ID Number: 851942907

LOGICAL SHORTCUTS

HEURISTIC STEPS IN LOGIC TUTORING SYSTEMS

Thesis Presentation: Friday July 10, 2020 at 1:00 PM.

Open Universiteit
de beste! www.ou.nl

LOGICAL SHORTCUTS

HEURISTIC STEPS IN LOGIC TUTORING SYSTEMS

by

Ir. R.(Ruben) J.Steins

in partial fulfillment of the requirements for the degree of

Master of Science

in Software Engineering

at the Open University of the Netherlands, Faculty of Science
Master's Programme in Software Engineering

to be defended publicly on Friday July 10, 2020 at 1:00 PM.

Student number:

Course code:	IM9906	
Thesis committee:	Dr. B. Heeren (chair),	Open University
	Drs. J. Lodder (primary supervisor),	Open University

Contents

SUMMARY

When entering solutions to problems in Interactive Tutoring Systems (ITS) students often skip or combine steps. How can such systems offer support for these heuristic steps? Based on analysis of the log data of an existing logic tutoring system, a classification of different types of heuristic steps has been devised.

A proof-of-concept implementation in the Logic-tutor LogEx has been used in a number of experiments with students to validate its usage.

Research has shown that the strategy-language used in the IDEAS-framework can be used to encode heuristic steps. Experiments show that they are liked and used by students when available.

SAMENVATTING

Bij het invoeren van oplossingen voor problemen in Interactieve tutorsystemen (ITS) blijken studenten vaak stappen over te slaan of te combineren. Hoe kunnen deze systemen dit soort heuristische stappen ondersteunen? Op basis van log-data-analyse van een bestaand systeem is een classificatie opgesteld van de soorten heuristsche stappen.

Een voorbeeldimplementatie hiervan in de Logicatutor LogEx is gebruikt in een aantal experimenten met studenten om het gebruik ervan te valideren.

Uit het onderzoek is gebleken dat de strategietaal die gebruikt wordt in het IDEAS-framework geschikt is om de heuristische stappen in vast te leggen en dat ze veelvuldig door studenten worden gebruikt indien aanwezig.

1

INTRODUCTION

Learning to solve mathematical problems is a bit like learning to ride a bike: while it is valuable to get a verbal explanation, to read about it in a book or watching someone in action, the only way to really become proficient is by trying until you fundamentally grasp the concept. Quick feedback during practice is essential. While the directness of a bump on the pavement is hard to match, and really has little value in mathematics education, there are other ways to provide learners with adequate feedback.

To practice a particular type of task, for instance rewriting a propositional logic formula into a different form, or solving equations for a particular variable, large amounts of exercises are required to allow the learner to practice as much as needed. While human tutors are perfectly capable of producing problem sets, the task to devise them and later correct and provide feedback on the results is laborious, especially when multiple students are involved. Since each student might struggle with different concepts, tailoring the exercises to focus on those problems, is an even bigger challenge.

Intelligent Tutoring Systems (ITS) are tools that can provide training material, such as exercises, and give feedback on submitted solutions. With an ITS students can practice on their own, at their own pace. With the ability to serve an endless stream of exercises without time constraints, it allows students to practice until they feel confident. Studies have shown that ITSs can be a very effective learning tool, when applied correctly [KF15; Mos+03].

There are a number of aspects that make an ITS particularly effective:

- Students learn the most when instructions they receive are individualized. A significantly better performance has been observed over students who “receive[d] classroom instruction” [Ma+14; Van11]. When instructions, in terms of feedback, are tailored to a particular student this leads to better performance.
- ITSs that mimic aspects of human tutors have also been highly successful. For in-

stance, if a tutor notices the student struggling with problems of a certain difficulty she might provide the student with a simpler one [MB17].

The way students enter solutions into an ITS is often done in a step-wise fashion. Each step brings a student closer to the solution. Usually the valid options are encoded in some set of rules, which is what the ITS uses to validate the input and provide feedback to the student: the entered step was correct or incorrect and the erroneous part is pointed out. The system might also provide hints on how to continue or what rules are applicable.

From a didactical standpoint it may be desirable that the student follows a certain order, or uses a specific strategy to solve the exercise. Students, especially those that are more advanced in the topic, may try to deviate from this and by combining certain steps when entering them, effectively skipping intermediate steps.

Take for example this arithmetic expression that is evaluated by working out the operations in the correct order of precedence (multiplication and division precede addition and subtraction). Each line shows a step towards the answer and the "rule" applied:

$4+8*2-5$	
$4+16-5$	Multiplication over addition/subtraction
$20-5$	Addition
15	Subtraction

A student with some more experience might perform the same evaluation as follows, combining several steps:

$4+8*2-5$	
$-1+16$	Multiplication AND subtraction
15	Addition

When the ITS rejects steps that are correct but do not follow the order or step-size expected by it, this may lead to confusion and frustration with students. Alternatively, the ITS might see the step is invalid, but is unable to give specific feedback. A human tutor would recognize these "shortcuts" taken by students and accepts the solution or is able to point out the error.

Analysis of the logfiles from an existing ITS, the Logic tutor LogEx, developed at the Open University, shows quite a few student attempt to use these kind of "heuristic steps" in their solutions. A small exploratory survey into Logic tutoring systems like LogEx shows a lack of support for them across the board.

The main focus of this thesis is to find out how support for heuristic steps can be added to ITSs. A classification of the different categories of heuristics steps is made, after which a proof-of-concept is developed by implementing a subset of heuristic steps into the LogEx

ITS. The implementation is tested in multiple experiments with students to see if and how the heuristic steps were used.

The main research question answered in this thesis is:

Research Question. *How can heuristic steps in solutions for proposition logic rewriting exercises in the LogEx tutoring system be detected?*

The contribution of this research is that a way has been found to describe Heuristic Steps in a usable way in an ITS. This addition leads to an ITS that behaves more like its human counterparts and allows students more freedom when entering solutions. A proof-of-concept of this has been produced and tested on students, who were mostly positive about the additions to the system.

The remainder of this thesis will go into details on how the above mentioned was achieved. First, all concepts will be defined and the context will be described in Chapter 2. Next, the methods used to answer the research question and validate the results is outlined in Chapter 3. Following this, the actual results are shown, including details on the implementation and experiments in Chapter 4. Finally, any threats to validity are discussed in Chapter 6, after which the conclusions and recommendations are given in Chapter 7.

2

CONTEXT

This chapter provides some more context, gives a definition for the most important concepts and shows an overview of related work.

2.1. INTELLIGENT TUTORING SYSTEMS

An Intelligent Tutoring System (ITS) can take many shapes and forms, but is defined here as a computer system that help students to learn a topic by:

- giving information about the topic
- offering questions, challenges, or assignments to practice the topic
- provide some kind of feedback on the quality of the answer
- give hints (feed forward) on how to proceed

Some ITSs include a representation of the student and their abilities to further tailor the amount and level of exercises presented, called a student model [Ma+14].

Usually an ITS consists of several conceptual units: the domain knowledge module, the student model module, the tutoring module and the user interface (UI) module [HJ14].

2.2. THE IDEAS FRAMEWORK

IDEAS (Interactive Domain-specific Exercise Assistants) is a "... generic Haskell framework for constructing the expert knowledge module [..] for an ITS or learning environment" [1].

[1] Taken from https://hackage.haskell.org/package/ideas on March 1, 2020

It provides a lot of Haskell types and functions to set up a domain knowledge base. The knowledge is captured in Rules which are combined into Strategies. An example of such a Rule would be, bringing back the arithmetic exercise from the Introduction, that multiplication precedes addition. A Strategy is defined a being either a Rule, or a set of Rules in a particular order or combination.

For example, and over-simplified, a Strategy to work out any arithmetic expression could be: apply the rule 'multiplication over addition' as many times as possible after which you apply the 'solve addition' rule as many times as possible.

The IDEAS framework uses a domain specific language (DSL) to define those strategies. Multiple combinators can be used to chain Rules together to indicate sequence (<*>) , choice (<|>, >|>, ▷) or repetition (*repeat*) [HJ17].

Several different tutoring systems have been developed as part of the IDEAS research effort: an interactive Haskell tutor [Ger12], a Java refactoring tutor [KHJ17] and a Logic tutor called LogEx [LHJ16].

2.3. LogEx

LogEx is an ITS to teach several concepts from propositional logic: rewriting into normal forms and proving logical equivalence. It does this by presenting students with exercises. Solutions are entered in very granular steps, which are compared to a set of Rules. After each step, the tool will give feedback to the student and may give hints on the next step.

When learning propositional logic, as part of a computer science curriculum for instance, students have to acquire proficiency in rewriting formulae into particular normal forms, notably the Conjunctive Normal Form (CNF) and Disjunctive Normal Form (DNF). This process, which involves applying multiple transformations to the original formula until the desired form has been reached, is non-deterministic in nature. Many different sets of steps lead to the desired outcome (although there are heuristics that help to get the conversion done quickly). Figure 2.1 shows a partially solved exercise in LogEx [LHJ16].

2.3.1. Normal Forms in propositional logic

The focus of this thesis is on the rewriting exercises to learn Normal Forms in propositional logic.

Normal Forms are special syntactical forms a formula can have. A formula is in DNF if it only consists of disjuncts of conjuncts, which in turn only consist of literals (atoms or negated atoms), as shown in Definition 2.1:

$$(\phi_1 \wedge \ldots \wedge \phi_n) \vee \ldots \vee (\chi_1 \wedge \ldots \wedge \chi_m), \text{in which } \phi_1, \ldots, \phi_n, \chi_1, \ldots, \chi_m \text{ are literals.} \tag{2.1}$$

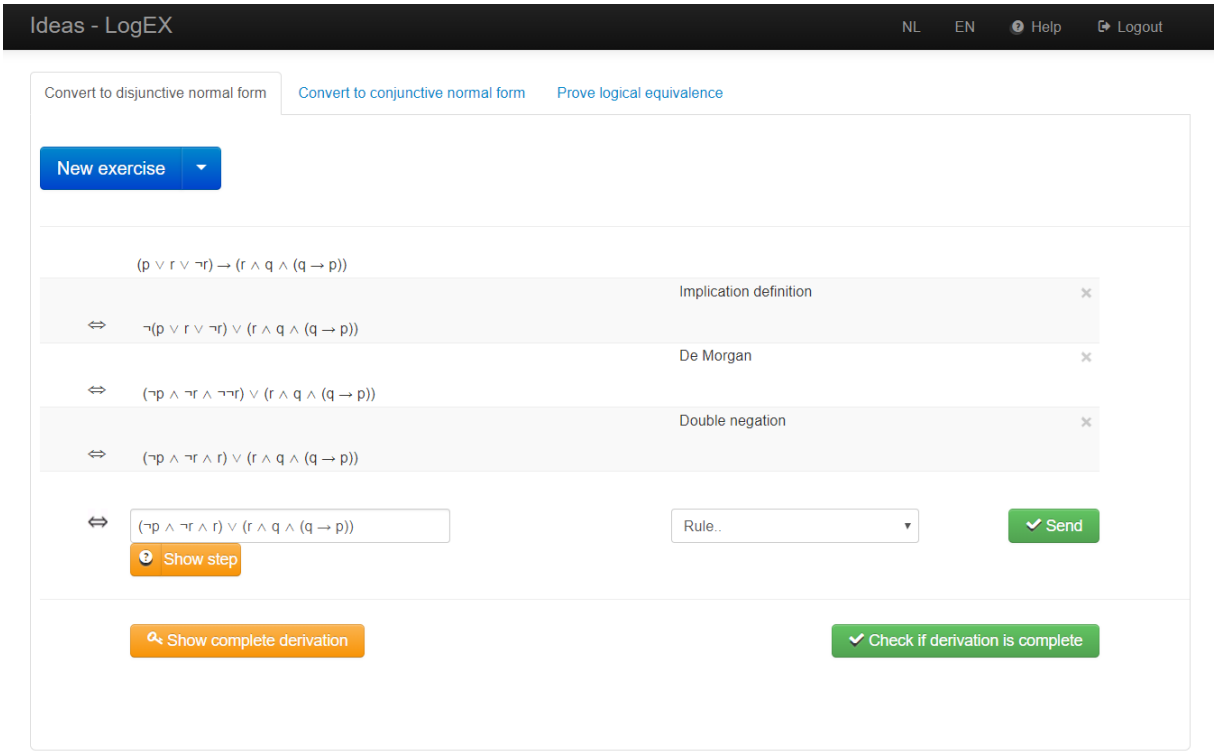

Figure 2.1: Screenshot of LogEx showing a DNF exercise.

The CNF is similar to the DNF, but formulae in this form consist of a conjunction of disjunctions of literals, as shown in Definition 2.2

$$(\phi_1 \vee \ldots \vee \phi_n) \wedge \ldots \wedge (\chi_1 \vee \ldots \vee \chi_m), \text{in which } \phi_1, \ldots, \phi_n, \chi_1, \ldots, \chi_m \text{ are literals.} \tag{2.2}$$

A more general form of both DNF and CNF is the Negation Normal Form (NNF), in which both conjunctions and disjunctions are allowed.

It can be proven that for each formula ϕ there is a logically equivalent formula ϕ' in CNF and a logically equivalent formula ϕ'' in DNF. This proof is omitted here, but can be found in textbooks on logic [Ben+14]. The CNF and DNF have several applications in mathematics and computer science, such as automated theorem proving and circuit theory. Being able to rewrite logical formulae into their CNF and DNF equivalents is a convenient skill to have as a student or practitioner in one of those fields.

2.3.2. Rewriting Rules

The set of supported rewriting rules for Propositional Logic in LogEx is included in Table 2.1. Each of these rules is recognized by the tool as a valid step which may lead towards a potential solution. The system can provide students feedback in the form of hints and suggestions for possible next steps. It can also detect that the student has made a common error or has applied a rule incorrectly [LHJ15].

Rule	Example
Absorption	$(p \wedge q) \vee q \Leftrightarrow q$ or $p \wedge (p \vee q) \Leftrightarrow p$
Commutativity	$(p \vee q) \Leftrightarrow (p \vee q)$ or $p \wedge q \Leftrightarrow q \wedge p$
De Morgan	$\neg(p \vee q) \Leftrightarrow \neg p \wedge \neg q$ or $\neg(p \wedge q) \Leftrightarrow \neg p \vee \neg q$
Distribution	$r \wedge (p \vee q) \Leftrightarrow (r \wedge p) \vee (r \wedge q)$ or $r \vee (p \wedge q) \Leftrightarrow (r \vee p) \wedge (r \vee q)$
Double negation	$\neg\neg q \Leftrightarrow q$
Equivalence Definition	$p \leftrightarrow q \Leftrightarrow (p \wedge q) \vee (\neg p \wedge \neg q)$
F-Rule Conjunction	$p \wedge \mathrm{F} \Leftrightarrow \mathrm{F}$
F-Rule Complement	$p \wedge \neg p \Leftrightarrow \mathrm{F}$
F-Rule Not T	$\neg\mathrm{T} \Leftrightarrow \mathrm{F}$
F-Rule Disjunction	$p \vee \mathrm{F} \Leftrightarrow p$
Idempotency	$q \vee q \Leftrightarrow q$
Implication Definition	$p \rightarrow q \Leftrightarrow \neg p \vee q$
T-Rule Conjunction	$q \wedge \mathrm{T} \Leftrightarrow q$
T-Rule Complement	$p \vee \neg p \Leftrightarrow \mathrm{T}$
T-Rule Not T	$\neg\mathrm{F} \Leftrightarrow \mathrm{T}$
T-Rule Disjunction	$p \vee \mathrm{T} \Leftrightarrow \mathrm{T}$

Table 2.1: Set of supported rewriting rules in the current version of LogEx

2.3.3. STRATEGY

While there are many possible series of rewriting steps that will convert any formula into a Normal Form, not all are equally efficient. Default strategies exist which, when applied consistently, will lead a student to a correct solution in a reasonable number of steps. For instance, the rewriting of any formula to the DNF or CNF can be done using the following strategy, which is also used in LogEx:

1. Remove Implication and Equivalences by using elimination rules;
2. Push negations inward using DeMorgan (after this step the formula is in NNF);
3. Distribute $\wedge$ over $\vee$ (for DNF) or $\vee$ over $\wedge$ (for CNF).

Although there are perhaps strategies that might lead to a DNF of CNF quicker or in less steps (at least for certain formulae), the one provided here will always lead to a correct solution. [2]) In this case the reliability and ease-of-use make it a sensible default Strategy.

2.3.4. SERVICES

The architecture of LogEx is service-based. The web front end ,the UI module, is only used to present information to the user and process input. All logic is performed in the Common

[2]https://people.eecs.berkeley.edu/~daw/teaching/cs70-f03/Notes/lecture07.pdf, retrieved 03-04-2020

Gateway Interface (CGI) back end. Each operation is available to the front-end application as a different service-call. The list of services is quite extensive, as can be seen in Figure 2.2. The ones used in this thesis are shown in Table 2.2.

Service	Purpose
`basic.allfirsts`	Returns a list of steps that are suitable, according to the strategy.
`basic.apply`	Applies a particular rule to the current expression, or an error, if the chosen rule is not applicable
`basic.diagnose-string`	Evaluates the expression. It can detect equivalence, or an incorrectly applied rule (Buggy) or a deviation from the strategy (Detour).
`basic.onefirst`	Gives a possible next step, according to the strategy.
`basic.ready`	Checks if the exercise is complete.
`basic.derivation`	Gives back the entire solution

Table 2.2: Most relevant Service end-points in LogEx and their purpose

2.3.5. Feedback

The feedback provided by LogEx on student input is based on the Feedback Strategies described by Narciss and consists of four categories [Nar13; LHJ16]:

1. whether the answer is correct or incorrect, using the `basic.diagnose-string`-service
2. what the correct result is, using the `basic.derivation`-service
3. the location of the mistakes and explanation of the error
4. hints on how to proceed, using the `basic.onefirst`-service

While giving specific feedback on Heuristic Steps is not part of this thesis, the feedback features are an integral part of LogEx and any alteration we make to LogEx should take this mechanism into account.

2.4. Granularity and Heuristic Steps

As said, students enter solutions to exercises in LogEx, and many other ITSs, step-by-step. Consider for example the following (partial) rewrite attempt in Equations 2.3, 2.4 and 2.5:

$$\neg((r \wedge p) \vee (q \wedge r)) \tag{2.3}$$

$$(\neg(r \wedge p) \wedge \neg(q \wedge r)) \tag{2.4}$$

$$((\neg r \vee \neg p) \wedge (\neg q \vee \neg r)) \tag{2.5}$$

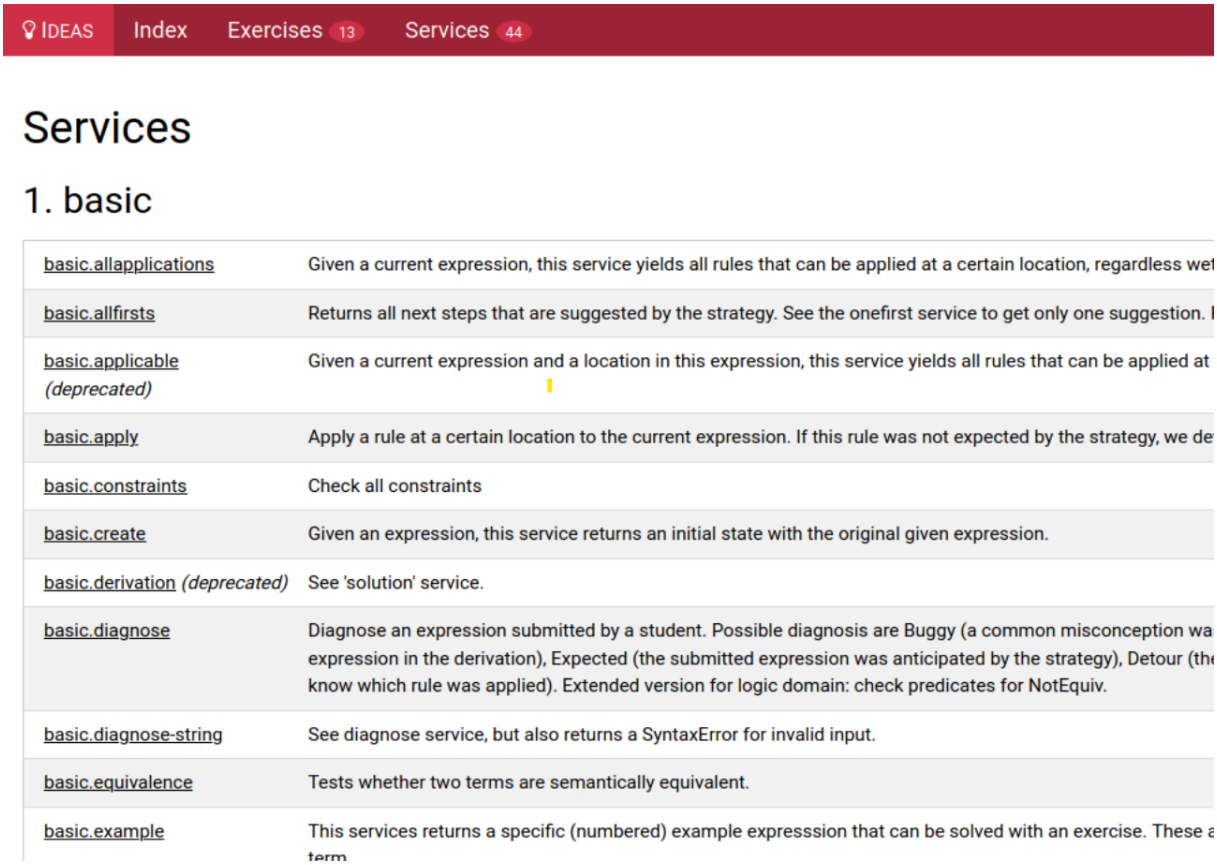

Figure 2.2: Screenshot of a partial LogEx services-list. Note the total number available at the top of the page.

The first step is done using an application of De Morgan's law (step 2.4). The second step applies it once again, but the student has observed a symmetry in the equation and applied De Morgan to both disjuncts in one go (step 2.5).

What happens here can be classified as a shift in Granularity. As a concept in computability this is defined as "a means of constructing simple theories out of more complex ones" [Hob90]. In the context of an ITS it can be seen as the "level of detail" used in solving a particular exercise, or more specifically, the size of steps taken by a student in solving a particular exercise.

When speaking of higher and lower levels of granularity, confusion about the meaning of the adjectives often arises. Does a higher granularity mean that the grains are smaller (more granular) or that the level of abstraction is higher (and thus the "grains" are bigger). To avoid this, the terms "fine-grained" and "course-grained" will be used instead.

When rewriting Equation 2.6 to DNF, a student could apply the "Implication Definition"-strategy (which essentially uses the equivalency: $p \rightarrow q \Leftrightarrow \neg p \vee q$) in two separate steps, resulting in steps 2.7 and 2.8, or she could apply the strategy twice in one go and jump straight to Equation 2.8. This latter approach is an indication the student is thinking at a more course-grained granularity.

$$((r \rightarrow p) \wedge \neg p) \rightarrow (\neg q \wedge \neg p) \tag{2.6}$$

$$((\neg r \vee p) \wedge \neg p) \rightarrow (\neg q \wedge \neg p) \tag{2.7}$$

$$\neg((\neg r \vee p) \wedge \neg p) \vee (\neg q \wedge \neg p) \tag{2.8}$$

Most people can easily switch between those different grain sizes and, when their knowledge advances, they can see patterns emerge [MG94]. Moreover, as Hobbs puts it, "they can have both deep and shallow knowledge at the same time" [Hob90].

These "organic granularity shifts" used by students when solving exercises are often recognizable by human tutors, who, due to their higher level of understanding, are capable of seeing the exercises at an even lower level of granularity (bigger grain-size). This knowledge can be used to guide students by giving appropriate feedback on a more general, strategic level [GM89].

The term Heuristic Step has been chosen to mean any deviation from the most fine-grained stepwise solution within the Strategy for a particular problem in an Intelligent Tutoring System.

2.5. Performance

The students' input is parsed into an Abstract Syntax Tree (AST) representation in LogEx. The AST is used by the different feedback services to decided which reply to give. To put is simply, any Rule that can be applied to a member of the AST is considered a possible next step. Not all of those are in the Strategy, so not all of them are returned by the `basic.onefirst`-service or `basic.allfirst`-service.

The amount of Rules in the system determine how many options have to be considered. The more rules there are, the more potential candidates should be checked. When adding Heuristic Steps to the LogEx system, care has to be taken to keep feature and performance parity (no loss of functionality or loss of performance speed).

2.6. Related Work

A useful comparison between several existing Logic tutors has been done by Lodder e.a., but unfortunately, the Heuristic Step behaviour was not part of their original comparison matrix [LHJ16]. Therefore a small explorative investigation of a number of other ITSs developed as part of ongoing or earlier research has been performed in order to find evidence of support for Heuristic Steps or similar mechanisms.

Figure 2.3: Screenshot of Deep Thought with an almost solved problem

2.6.1. ASK-ELLE

The Haskell tutor Ask-Elle has a system to dynamically promote or demote certain strategies from major to minor to allow differentiating step size. Minor rules are "...used to perform administrative tasks, such as moving down into a term, updating an environment, or automatically simplifying a term ..." [Ger12]. Because Ask-Elle is based on the Ideas-framework as well, further investigations into the mechanism applied here might prove fruitful.

2.6.2. DEEP THOUGHT

Deep Thought, developed at NC State University is a Logic Tutor that focusses on "the practice of solving deductive logic proof problems in graphical representation" [3] (see also Figure 2.3) [MB17]. This tool uses a data-driven approach to enhance the tutors behaviour with regards to hints provided and problem selection. Step-size and granularity are not mentioned in the research at all. Several small tests have not been able to detect this. In fact, feedback seems limited to "Incorrect rule application" in most cases. This might be because "proof problems" require each step to be explicitly motivated and thus do not leave much room for Heuristic Steps.

[3] http://eliza.csc.ncsu.edu/DeepThought

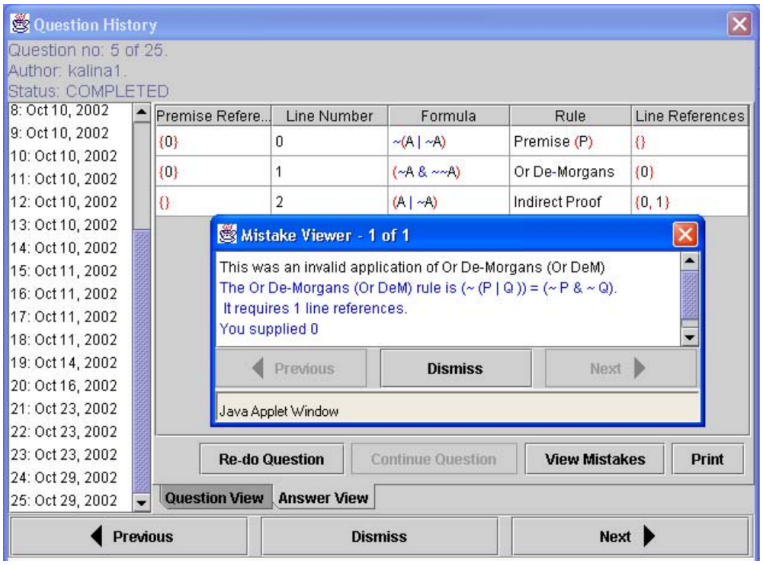

Figure 2.4: Screenshot of the original Logic-ITA taken from [Yac03]

2.6.3. LOGIC-ITA

The Logic ITA developed at the University of Sydney validates individual steps on their own merit, without checking its appropriateness [Yac05]. Afterwards the tool can give an indication whether or not a particular step was useful in solving the exercise.

When evaluating steps entered, the tool first analyses the validity of the submitted formula (both syntactically and logically). When valid, all parts of the entry must be valid (i.e. the specified rule, the referenced lines and premises). If the entered step is invalid, the system will check if alteration of one of these components will lead to a valid solution. This information is used to supply the student with an appropriate hint [Yac03]. Figure 2.4 shows an example of this.

2.6.4. ORGANON

ORGANON is a logic tutor developed by the University of West Bohemia in Pilsen, which supports a number of different logic exercises among which are rewriting to CNF and DNF [DL07]. It was impossible to access a running version of the tool. The authors indicate the tool is capable of giving feedback on individual steps, but do not mention step size at all.

2.6.5. FMA/CAL

Prank e.a. allow multiple input modes for their tutor. In “free input mode” students can enter steps that consist of arbitrarily long strings [Pra14]. No stepwise feedback is given when in this mode, because the bigger steps “make it harder to recognize the reasons for non-equivalence”. In other words, students can make unpredictable leaps of thoughts that are harder to diagnose. The alternative is using a rule-based approach, which “[..] allows

the user to ignore low-level details". This latter approach leaves no room for interpretation since the student is not allowed to enter free text.

2.6.6. SETSAILS!

Zimmerman and Herding developed SetSails!, a German tutor containing set-theoretical questions [ZH10]. It is unclear from their research if dynamic setup sizes are supported in any form. However, the apparent lack of the possibility to enter free-form solution steps and the focus on a single correct solution (*"Anwendungen hingegen haben im Hintergrund meistens nur einen korrekten Lösungsweg"*), seems to imply that this is not the case. While the software was available for download, I was unable to get it to run.

2.6.7. CTAT

The Cognitive Tutor Authoring Tools (CTAT) developed at Carnegie Mellon is a suite of authoring tools for tutors [Ale+06]. An example logic tutor has been developed [4] of which a screenshot is shown in Figure 2.5. The tutor (which trains conversion to the Negation Normal Form) lets the student select a particular part of the formula and a strategy ("rewrite rule") to apply to that selection. Selecting a too small or too large part of the expression will result in an error. The tutor does not seem to support application of the same rule to multiple parts of the formula in one step. Whether or not this is a limitation of this specific tutor or the authoring tools is unclear and requires more study.

2.6.8. LOGIC/SELL

At the Open University of Catalonia a logic tutor (Logic) was developed, but no details pertaining its implementation were described [Hue+11]. Unfortunately there is no longer a version available online to test with.

2.6.9. FINDINGS

It seems most logic ITSs currently used do not have any support for Heuristic Steps. The rule-based approach used in FMA/CAL seems to be a more strict form of the strategy selection used in LogEx [Pra14]. Including explicit 'Heuristic Step-strategies' has been . The "upgrading" of strategies used in Ask-Elle is worth looking into, although the type of problems in that ITS are not entirely comparable to those in LogEx [Ger12]. The possibilities offered by other (as opposed to Ideas) authoring tools, such as CTAT have not been investigated [Ale+06].

[4] Cdn.ctat.cs.cmu.edu/logic-tutor/html

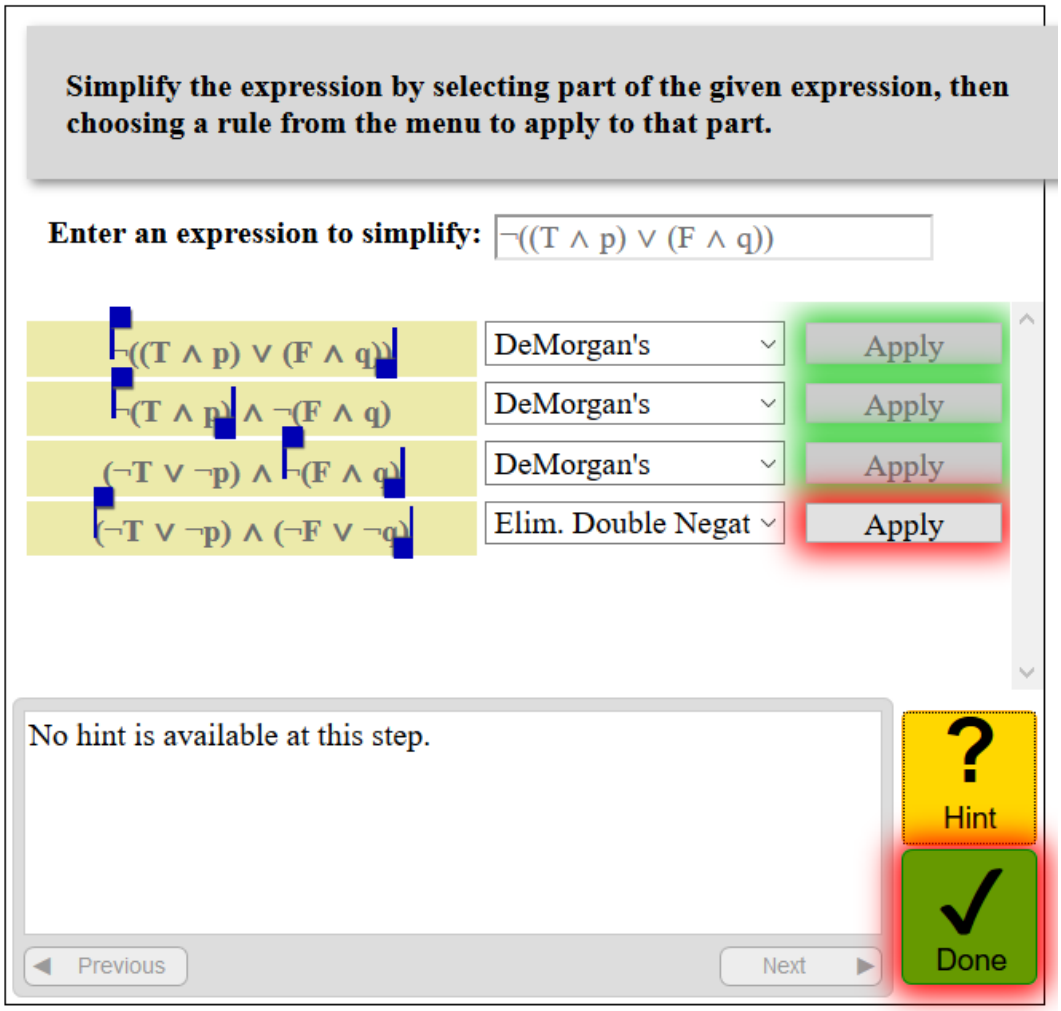

Figure 2.5: Screenshot of a Logic Tutor built with CTAT

3

RESEARCH METHOD

3.1. MAIN RESEARCH GOAL

Cursory examination of the LogEx logfiles indicated that many students attempt to use Heuristic Steps when doing rewriting exercises. LogEx had no support for those, so students were confronted with unexpected error messages, such as the one in Figure 3.1. Allowing Heuristic Steps will make LogEx more closely mimic human tutor behaviour.

Therefore, the following main research question in this thesis was formulated:

Research Question. *How can heuristic steps in solutions for proposition logic rewriting exercises in the LogEx tutoring system be detected?*

3.1.1. RESEARCH QUESTIONS

In order to answer the main Research Question, three sub-questions have been formulated, which will be elaborated upon in the following sections:

- **RQ1:** What are common Heuristic Steps taken by students and how can they be classified?
- **RQ2:** How can we implement Heuristic Steps in LogEx?
- **RQ3:** How can we detect Heuristic Steps in entered solutions?

This chapter describes, for each of the three research questions, how they are answered and how the results are validated. If necessary, some additional context is also given.

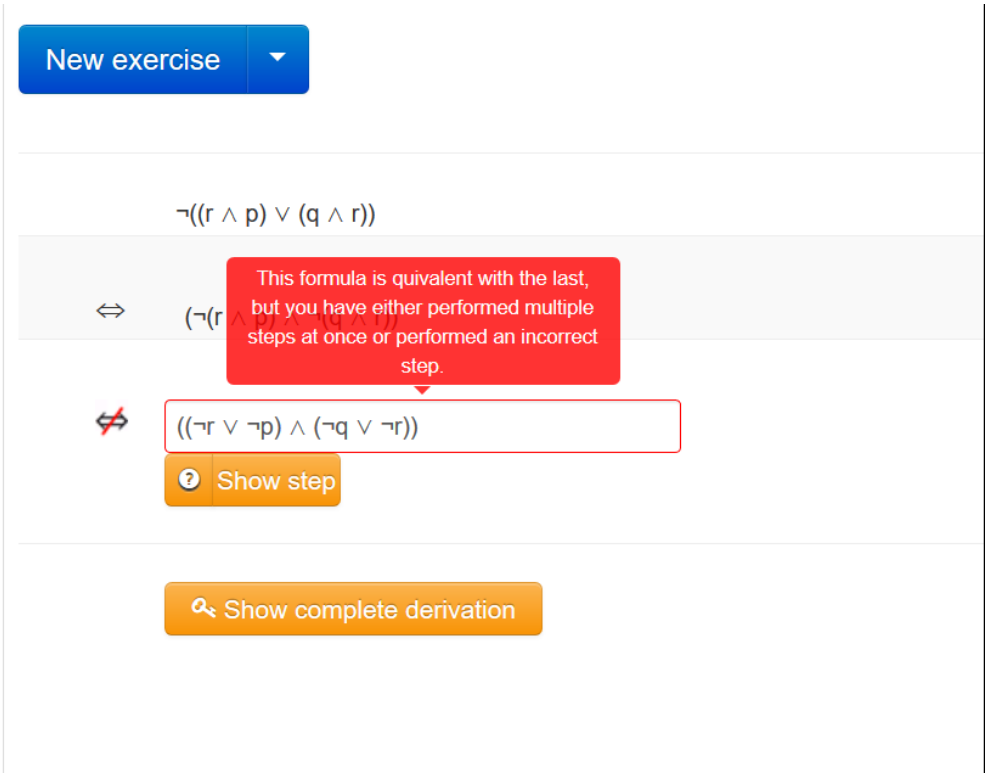

Figure 3.1: Screenshot of LogEx showing an error due to a student applying a Heuristic Step.

3.2. RQ1: What are common Heuristic Steps taken by students and how can they be classified?

The answer to this research question will be a typology of the different types of Heuristic Steps taken by students in the context of solving exercises in the LogEx ITS.

3.2.1. Sources of information

A number of sources were used to gather information on the usage of Heuristic Steps, both in theory and in practice.

1. **LogEx logfiles:** LogEx logs every step a student has entered as well as the system response in great detail. Analysis of the logfiles of several experiments, some of which was conducted before this graduation project started will provide insight in how the system is used. Some automation to help the analysis may be possible.

2. **Homework Submissions on Paper:** A second source of information are the submissions (on paper) of mathematics homework done by students in a Discrete Mathematics course. These paper submissions have to be studied by hand. Since these results represent the students' train of thought most accurately, this source might prove to be the most valuable.

3. **Textbooks:** A third source of potential Heuristic Steps are mathematics textbooks, which may contain examples or solutions that qualify as Heuristic Step.

3.2.2. Validation

The validity and usefulness of the typology is tested using experiments with students solving LogEx exercises. From the logfiles generated during experiments and survey-data gathered afterwards, the usage of Heuristic Steps is determined.

3.3. RQ2: How can we describe Heuristic Steps in an ITS?

Specifically: how can Heuristic Steps be described in such a way that LogEx is able to use them.

3.3.1. Approach

In order to answer this question, a subset of the identified Heuristic Steps is implemented in an experimental version of LogEx. The implementation adheres to the IDEAS standards and works without disrupting the "normal" program flow.

The implementation is written in Haskell and is accessible online for testing purposes. [1]

3.3.2. Backwards compatibility and performance

Besides working correctly, the solution also has to perform reasonably well (no exact demands have been specified) and solution entered without Heuristic Steps still need to be supported.

The potential explosion of state-space is taken into account when choosing a solution. Every addition to the rule-set potentially increases the amount of work the LogEx-services have to perform.

3.3.3. Validation of results

The results will be tested in multiple experiments with students and by running manual and automated tests against the application.

[1] https://ideastest.science.uu.nl/logic-step/, retrieved 2020-07-01

3.4. RQ3: How can we detect Heuristic Steps in entered solutions?

This research question has been answered with a prototype of LogEx that is able to detect (a subset of) the identified Heuristic Step-types.

3.4.1. Approach

An extension to LogEx has been be written in Haskell that allows the ITS to detect (a number of) Heuristic Steps in stepwise solutions entered in the tool.

3.4.2. Validation

A set of test cases has been devised based on the different Heuristic Step-categories defined in RQ1. These are used to determine if and how easy LogEx is able to detect the Heuristic Steps. Performance levels for the solution should also be acceptable (the thresholds for which have to be determined as well) and impact will be measured.

3.4.3. Experiments with students

A real-life test of the prototype has been conducted twice with different groups of students. These were heterogeneous in terms of knowledge level, but for quite a few students conversion to DNF or CNF was most likely a relatively new topic and a very brief explanation on the DNF and CNF.

The used were offered a chance to use LogEx in preparation for their exams. The students were informed beforehand that certain experimental features were being tested as well, but the specifics were not disclosed. They received a quick explanation on LogEx.

Predefined exercises

While students were free to enter any exercise they liked, or work on an auto-generated one, a list of pre-defined exercises was supplied. This list was chosen in such a way that the Heuristic Steps implemented in the prototype could be used in solving them.

Experiment 1, 10-2019

In October 2019 an experiment was conducted with about fifteen volunteers, all students following the course “Premaster formele technieken 1: discrete wiskunde en logica” (“Pre Master formal techniques 1: discrete mathematics and logic”) as part of their studies at the Open University. The students were asked to use LogEx to rewrite several expressions into DNF. After the explanation students used the tool for about one hour. A list of exercises was shared with the students at the start of the experiment. It contained thirteen exercises of

different difficulty level as shown in Table 3.1. The qualification ("Easy", "Normal" etc. are those used in LogEx as well).

Nr	Exercise	Level
1	$\neg(p \rightarrow \neg q)$	Easy
2	$\neg(\neg p \wedge \neg(q \vee r))$	Easy
3	$\neg\neg p \leftrightarrow \neg\neg q$	Easy
4	$\neg\neg(q \vee p) \wedge \neg\neg(q \vee r)$	Medium
5	$\neg((q \rightarrow r) \rightarrow \neg q)$	Medium
6	$q \wedge p \wedge q \wedge (\neg q \leftrightarrow \neg r)$	Medium
7	$(\neg(p \rightarrow q) \vee \neg(r \rightarrow s)) \rightarrow \neg s$	Medium
8	$\neg(p \wedge q) \leftrightarrow \neg p$	Medium
9	$r \leftrightarrow ((p \wedge q) \vee (p \wedge r))$	Medium
10	$(p \rightarrow q) \leftrightarrow (r \rightarrow s)$	Difficult
11	$\neg(\neg p \vee (r \leftrightarrow s))$	Difficult
12	$((r \rightarrow s) \leftrightarrow (p \vee s)) \wedge ((p \wedge r) \vee (q \wedge p))$	Very Difficult
13	$((s \leftrightarrow q) \wedge \neg r) \leftrightarrow p$	Very Difficult

Table 3.1: Predefined exercises for the first experiment

At the time of this first experiment, the only heuristic step that was supported was the homomorphic double negations. Afterwards most students (thirteen) filled out a survey.

EXPERIMENT 2, 02-2020

The second experiment was conducted in January of 2020 with a group of 23 students, all following the course "Logica, verzamelingen en relaties" ("Logic, Sets and Relations") as part of their studies at the Open University. Again, the students were asked to use LogEx to rewrite several expressions into DNF. After the explanation students used the tool for about one hour.

At this point in time, the Homomorphic Subformula Heuristic Steps: multiple double negation, multiple implication and multiple equivalence, as well as the Granularity-based DeMorgan/Double-negation combination were supported. Again, a large portion of the students (nineteen) filled out a survey on their experience with the tool.

This time, students could choose from the list of pre-defined exercises from within the application. The options to manually enter formulae or have the system auto-generate one were also available. The list of included exercises is shown in Table 3.2.

3.4.4. SURVEY QUESTIONS

The survey used for the different experiments contains the following questions:

1. What was your account number?
2. How would you rate your level of proficiency in propositional logic?

Nr	Exercise	Level
1	$\neg(p \rightarrow \neg q)$	Easy
2	$\neg(\neg p \wedge \neg(q \vee r))$	Easy
3	$\neg\neg p \leftrightarrow \neg\neg q$	Easy
4	$\neg((q \rightarrow r) \vee q \vee r$	Easy
5	$(p \rightarrow r) \vee (q \rightarrow r)$	Easy
6	$\neg(\neg p \vee \neg q)$	Medium
7	$\neg(p \wedge q) \leftrightarrow \neg p$	Medium
8	$(p \rightarrow q) \leftrightarrow (r \rightarrow s)$	Difficult
9	$\neg(\neg p \vee (r \leftrightarrow s))$	Difficult

Table 3.2: Predefined exercises for the first experiment

(a) Fundamental Awareness (basic knowledge)

(b) Novice (limited experience)

(c) Intermediate (practical application)

(d) Advanced (applied theory)

(e) Expert (recognized authority)

3. How many problems did you attempt to solve using the LogEx Logic Tutor?

(a) <5

(b) 6-10

(c) >10

(d) Unspecified

4. How easy was entering solution steps in the LogEx Logic Tutor?

(a) Very easy

(b) Easy

(c) Not particularly hard or easy

(d) Hard

(e) Very Hard

5. Did any of your solution steps get rejected even though you knew they were correct?

(a) No, not at all

(b) Yes, once or twice

(c) Yes, on multiple occasions

6. Did you make use of the option to rewrite two double negations in a single step (e.g. $\neg\neg p \vee \neg\neg q \Leftrightarrow p \vee q$)?

(a) Yes

(b) No

(c) Not sure

7. Did you make use of the ability to rewrite using De Morgan and Double Negation in a single step (e.g. $\neg(\neg p \vee \neg q) \Leftrightarrow p \wedge q$)? [2]

   (a) Yes

   (b) No

   (c) Not sure

8. How much does using the LogEx Logic Tutor contribute to your understanding of propositional logic?

   (a) Not at all

   (b) A little bit

   (c) A lot

9. How satisfied (on a scale of 1-10) were you about your overall experience with the LogEx Logic Tutor?

10. Do you have any remarks (open question)?

[2] This question appeared only in the survey for the second experiment

4

RESULTS

This chapter provides an overview of the most relevant findings for each research question.

4.1. CLASSIFICATION OF HEURISTIC STEPS

To establish this classification the LogEx log-files of experiments with students were studied to determine if common Heuristic Steps could be identified. Four different experiments were conducted with groups of students on four occasions, two of which as part of this graduation project and homework of twelve students was analysed.

4.1.1. IDENTIFIED CATEGORIES OF HEURISTIC STEPS

After analysing the logs and homework assignments, the following categories have been defined. The names initially assigned to them during analysis were altered to better reflect the nature of the type of Heuristic Step.

1. Homomorphic Sub-formulae
2. Housekeeping
3. Granularity Based

HOMOMORPHIC SUB-FORMULAE

Many students apply the same rewriting rule on multiple sub-formulae in a particular formula at the same time if these sub-formulae are homomorphic, that is, only the propositional letters are different, but the structure is the same. The original name for this category "Symmetry" was too narrow a description.

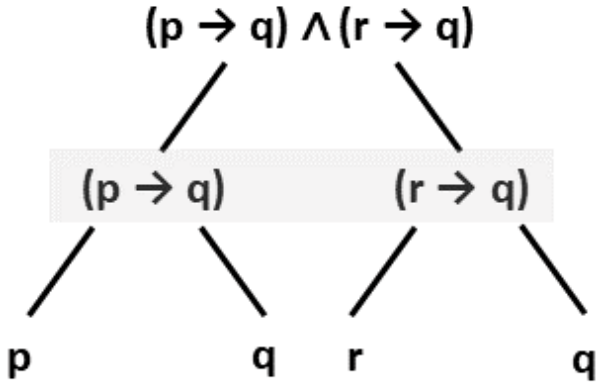

Figure 4.1: Parse tree for $((r \rightarrow q) \vee (p \rightarrow q))$

Formally this means that the sub-formulae follow the same schema. For instance $p \rightarrow q$, $r \rightarrow s$ and $(r \vee p) \rightarrow (s \wedge q)$ are all instances of the formula schema $\phi \rightarrow \psi$.

If students recognize that multiple parts of a formula follow the same schema, they might attempt to apply the same rule to all of them in a single step. The most frequent application of this has been observed for exercises that contained multiple double negations.

If say, a formulae contains the sub-formula $\neg\neg p \vee \neg\neg q$, a number of student will try to apply the "DoubleNeg"-rule to both instances of the schema $\neg\neg\phi$ in one go and submit $p \vee q$

Not all multiple applications of the same rule fall into this category. For instance in $\neg\neg(\neg\neg p)$, although the double negation $(\neg\neg)$ occurs twice in the formula, the approach is different: there is overlap between the two expressions.

If we would make parse trees for Formula 4.1, and Formula 4.2, as shown in in Figure 4.1 and Figure 4.2 the difference would become more clear.

The homomorphic sub-formulae need to be either the same level in the hierarchy of the tree or there should be no overlap. If this is the case, the same rewrite can be applied without problems. If not, they fall into a different category.

$$((r \rightarrow q) \vee (p \rightarrow q)) \tag{4.1}$$

$$(p \rightarrow (q \rightarrow t)) \tag{4.2}$$

Granularity Based

As can be seen in the second parse tree, the homomorphism is found on different levels. This "nested homomorphism" is in fact a shift in grain size: the student is using a more coarse-grained approach. This requires her to reason about the formula on several levels

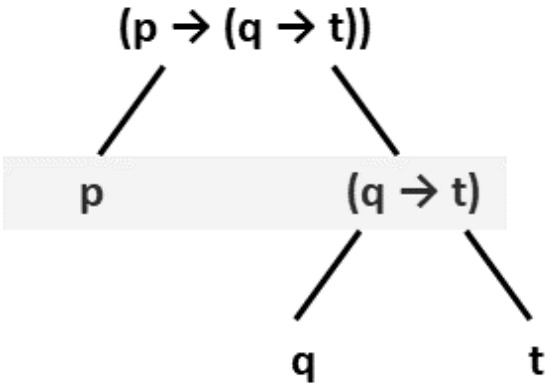

Figure 4.2: Parse tree for $(p \rightarrow (q \rightarrow t))$

of granularity at the same time, whereas in the first category all homomorphisms are of the same grain size. This warrants introduction of a second category, that of the Granularity Based Heuristic Steps.

Another shift in grain size is seen in the following scenario. While the strategy described in Section 2.3.3 will always lead to the correct normal form, it is not guaranteed this is the most efficient way.

Take as an example the formula $\neg((p \rightarrow q) \wedge p) \vee p)$. If we ask LogEx to show the complete derivation we get the following steps:

$\neg((p \rightarrow q) \wedge p) \vee p)$	
$\neg((\neg p \vee q) \wedge p) \vee p)$	Implication definition
$\neg((\neg p \wedge p) \vee (q \wedge p)) \vee p$	Distribution
$\neg(F \vee (q \wedge p)) \vee p$	F-rule complement
$\neg(q \wedge p) \vee p$	F-rule disjunction
$\neg q \vee \neg p \vee p$	De Morgan
$\neg q \vee T$	T-rule complement
$\neg T$	T-rule disjunction

If we postpone applying the "Implication Definition" we get a much simpler solution:

$\neg((p \rightarrow q) \wedge p) \vee p)$	
$\neg(p \rightarrow q) \vee \neg p \vee p)$	De Morgan
$\neg(p \rightarrow q) \vee T$	T-Rule Complement
T	T-rule Disjunction

Furthermore, some steps can be considered less important than others, depending on the level of the student. For these students all the "hoops" they have to jump through could

be frustrating. The learning gain would presumably be lower in this case. Consider the following rewrite steps:

$\neg(\neg p \vee \neg q)$	
$\neg\neg p \wedge \neg\neg q$	De Morgan
$p \wedge q$	Double Negation (2x)

The double negation, once mastered, is trivial to apply. Conceivably, a student can work at a level that requires less detailed "in-between" steps, and write this instead:

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

The grain size can be coarser allowing the student to rewrite in bigger chunks.

Another example would be:

$(p \rightarrow (q \rightarrow (s \rightarrow r)))$	
$\neg p \vee \neg q \vee \neg s \vee r$	Implication Definition

HOUSEKEEPING

The third category is about cleaning up and reordering the formulae while performing another rewrite operation. When rewriting a formula by meticulously following the strategy and all rules to the letter, a lot of extra steps have to be taken to "clean up" the formula. In the following (partial) derivation to DNF we see that in order to use "Idempotency" in step 4, we need to first explicitly reorder the first disjunct by applying "Commutativity":

$(p \wedge q) \leftrightarrow p$	
$(p \wedge q \wedge p) \vee (\neg(p \wedge q) \wedge \neg p)$	Equivalence Definition
$(p \wedge p \wedge q) \vee (\neg(p \wedge q) \wedge \neg p)$	Commutativity
$(p \wedge q) \vee (\neg(p \wedge q) \wedge \neg p)$	Idempotency

Much more intuitive would be to either allow the student to apply the "Commutativity"-rule at the same time as the "Equivalence Definition" or allow the "Idempotency"-rule a bigger scope, taking more siblings into account:

$(p \wedge q) \leftrightarrow p$	
$(p \wedge q \wedge p) \vee (\neg(p \wedge q) \wedge \neg p)$	Equivalence Definition
$(p \wedge q) \vee (\neg(p \wedge q) \wedge \neg p)$	Idempotency

The "Housekeeping"-category also includes the removal of unnecessary parentheses:

$(p \vee q) \rightarrow (q \vee s)$	
$\neg(p \vee q) \vee (q \vee s)$	Implication Definition
$\neg(p \vee q) \vee q \vee s$	Remove Parentheses

Because LogEx works with abstract syntax trees, parentheses are removed and added automatically, this latter feature is already supported by the current version.

4.1.2. Log Analysis Results

Logs from earlier experiments

The first two sets of logs that were analysed, were taken from two experiments done with a large group of computer science students at a University of Applied Science ("HBO" in Dutch) in The Netherlands. One group used the complete tool with all functionality available at that time, including hints about how to proceed and possible next steps. This set is identified by "logex-hints" from now on. The second group of students used the tool but could not use hints and the next step functionality. This set is identified with "logex-nohints". The experiments were done to test the value of these features and are described in [LHJ19].

The assumption was that the logfiles of the "logex-hints" would have limited evidence of the usage of Heuristic Steps. Since LogEx provided stepwise feedback, most students quickly learn that the Heuristic Steps they want to take are not supported by the tool. They will adapt their input and no longer utilize the Heuristic Steps. It turned out however that both sets contained evidence of Heuristic Steps, the "logex-hints"-set even more than "logex-nohints"-set.

Each of these sets of log data was analysed by utilizing the tool describes in Section 5.1. Table 4.1 shows a quantitative breakdown of each dataset. Each log-entry consists of a single interaction with the user and is either a requests for a new exercise, for feedback or a submission of a single step of the solution. The system contained a small set of predefined exercises from which students could choose, but each exercise could be attempted any number of time by any student, so there are many more solutions than there are exercises.

Table 4.2 shows for each of these categories how many solutions were found containing an (attempted) heuristic step. Some solutions contained evidence of multiple types, so they are included in more than one category.

4.1.3. Homework analysis

Homework assignments submitted by students, were another source of potential usage of heuristic steps. Twelve students were allowed their homework to be analysed. They all submitted solutions to the following two exercises:

	logex-hints	logex-nohints
Total number of log entries	6734	3760
Number of distinct users	45	32
Number of distinct exercises	13	13
Number of solutions	555	373
Entries with status Irrelevant	552	617
Entries with status LookedAt	5344	2506
Entries with status Relevant	838	637
Solutions with HS Evidence	41	43
% of solutions with HS Evidence	11,5 %	7,4 %

Table 4.1: Numerical analysis of log data of earlier experiments

	logex-hints	logex-nohints
Homomorphic Subformulae	19	14
Housekeeping	5	18
Granularity Based	19	20

Table 4.2: Different types of heuristic steps

Exercise 1
Let ϕ be the formula $\neg p \vee (q \rightarrow p)$
Prove in two different ways ϕ is a tautology:

a using the truth table for ϕ.

b using standard equivalences.

Exercise 2
Let ϕ be the formula $((p \wedge q) \rightarrow r) \rightarrow (\neg p \wedge q)$.
Give both the disjunctive normal form and conjunctive normal form of ϕ .

Of the twelve submissions, ten contained evidence of Heuristic Step usage. About half of the students applied double negation twice in one step for instance, an example of which is shown in Figure 4.3

4.1.4. Literature Study

No extensive literature study of mathematics and logic textbooks has been done, but a cursory glance through a number of textbooks, showed little to no results. Most descriptions found explained the “proper” way to rewrite formulae and provided as much detail and intermediate steps as possible.

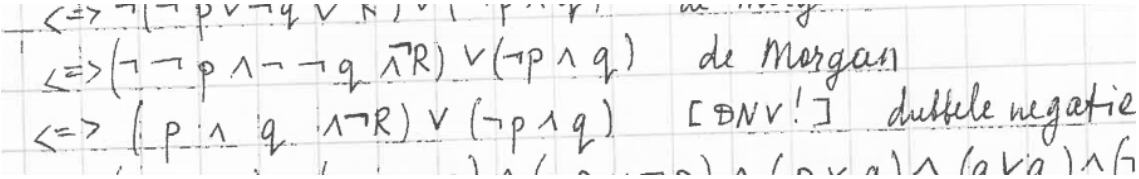

Figure 4.3: Part of homework submission with evidence of Heuristic Step

4.1.5. VALIDATION OF CATEGORIES

Two experiments were conducted to validate the identified categories. At the time of the first experiment, the only supported category was Homomorphic Subformulae, and only for double negation. At the time of the second experimen, the Granularity-based combination of De Morgan and double negation was also supported.

The logs of both experiments were analysed in a similar fashion as the "historical" logs therefore the results can be compared.

LOGS FROM EXPERIMENTS

The log files from both experiments (see Sections 3.4.3 and 3.4.3 for details on the participants and setup) were analysed in the same way as the older experiments. The statistics are found in Table 4.3.

COMPARISON

As can be seen from the data, the availability of Heuristic Steps leads to a much higher usage percentage, from about 10 % of the exercises to more than a quarter of the solutions in the first experiment and more than half of the exercises in the second one.

	exp1	**exp2**
Total number of lines	1499	2141
Number of users	13	20
Number of different exercises	78	96
Number of tasks	167	309
Lines with status Irrelevant	93	107
Lines with status LookedAt	930	831
Lines with status Relevant	476	1159
Tasks with HS Evidence	46	175
% of tasks with HS Evidence	27,5 %	56,6 %

Table 4.3: Numerical analysis of log data from new experiments

A more detailed analysis has also been (partially) conducted to determine the type of Heuristic Step actually used. The results are shown in Table 4.4.

The data shows that when Heuristic Steps are available, they tend to be used frequently by

	epx1	exp2
Homomorphic Subformulae	44	158
Housekeeping	-	-
Granularity-based	2	23

Table 4.4: Different types of heuristic steps in experimental versions

students. This is also a clear indication that the identified types of steps are relevant. No evidence of new categories was found during analysis of the data.

4.2. Implementing Heuristic Steps in LogEx

In order to make use of all the work already done, an effort was made to stay within the current structure of how LogEx is setup. For example, the main strategy for rewriting to DNF is coded (in Haskell) as follows:

```
dnfStrategy :: LabeledStrategy (Context SLogic)
dnfStrategy = label "DNF" $ repeatS $
orRules <|> somewhere (nnfStep |> distrAnd)
```

The Choice combinator (<|>) [1] indicates that there is no preference for either side. So either one of the *orRules* is applied, or the second option can be used. This is again a choice, but one with a bias for the left option (|>). So preferably a "nnfStep" is applied, *somewhere* to a subformula (or the entire formula), but if this is not applicable, "distrAnd" is also acceptable.

Since the *disrtAnd* will convert a Negation Normal Form (NNF) formula into DNF form, our new rules should really be part of the *nnfStep*-strategy. A number of implementation options were considered, which are described here very briefly. More implementation details can be found in Chapter 5.

1. **Client-Only Implementation:** change only the UI-module of the ITS and use the underlying back-end as-is. This would be a breach in the architecture of LogEx, would move complex application logic to the frontend and would cause performance challenges. There are currently 44 rules in LogEx. Using this approach would lead to $44^2 = 1936$ additional checks which might lead to performance issues. If the number of steps we want to check increases, the state-space will grow exponentially. Because of this, this approach was rejected.

2. **Heuristic Steps as Rules:** add every Heuristic Step as a rewriting Rule to the set already in LogEx. While this option requires very little modification in the current LogEx code, the number of different Rules that have to be defined would be very large,

[1] Note that LogEx uses the *legacy* version of this combinator. In the current version of IDEAS this combinator is .|.

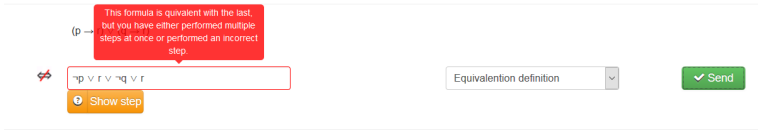

Figure 4.4: Showing an attempt at using Multiple Implication Elimination in original LogEx

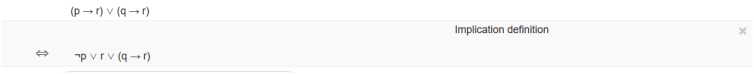

Figure 4.5: Multiple Implication Elimination in the experimental version of LogEx

which would a) be a lot of work and b) cause the same performance issues as in the Client-Only approach. This solution was also rejected in the end, though the prototype used in the October 2019 experiment was built using this technique.

3. **Partial Strategies as Rules:** define partial strategies that describe the Heuristic Steps and collapse them into Rules. This allows us to describe the Heuristic Step in a more abstract fashion. For example: "apply the rule "double.negation" one or more times in a single step." The definition of the different Heuristic Steps is more complex than with the other two approaches, but this solution follows the IDEAS framework best and does not lead to a large amount of new Rules.

SEEING IT IN ACTION

These Heuristic Steps were all implemented and in use during the second experiment (see Section 3.4.3).

Figure 4.4 shows how an attempt to apply the rule *Implication Elimination* multiple times will give an error. In Figure 4.5 we can see that the experimental version will accept this. When showing an example derivation it will however still favour the single application of the rule, as can be seen in Figure 4.6. Finally, in Figure 4.7 we see a combination of DeMorgan with two double negations being accepted in one step.

4.2.1. UNIT TEST AND TEST RESULTS

For each of the implemented Heuristic Steps a set of tests were written, which can be run (automatically or manually) to validate their functioning. More details on these tests and the full output of the test-runner has been included in Appendix A.

These test use the IDEAS-framework test facilities which have no out-of-the-box support for calculating code coverage no attempt has been made to calculate this by hand or using external tooling, such as Haskell Program Coverage (HPC). [2] Using `Haskell Program Coverage` In this regard, the test report does not provide a formal proof of any kind. Nev-

[2]https://wiki.haskell.org/Haskell_program_coverage, retrieved 2020-07-01

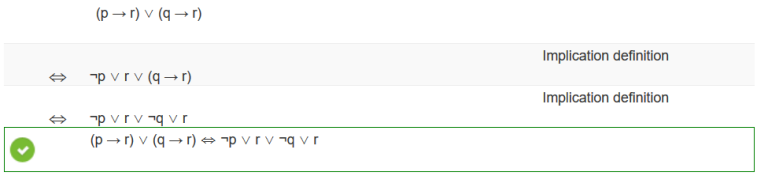

Figure 4.6: The “Show Complete Derivation” option prefers single Implication Elimination

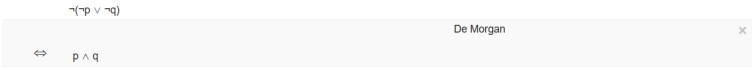

Figure 4.7: DeMorgan combined with (multiple) Double Negation

ertheless, the fact that all these test pass gives some confidence in the correct workings of the implemented rules.

4.3. Putting Heuristic Steps to the test

After a number of Heuristic Steps were implemented they were put to the test using two experiments with students. The response rates were]high: 87% of participants in the first experiment and 83% of those in the second filled out the survey after the experiment.

4.3.1. Survey Questions and Answers

A summary of the results from survey held after the two experiments conducted, is shown here. Some results were redacted to ensure participant anonymity. [3]

Proficiency in propositional logic

We asked all participants to self-assess their level of proficiency in rewriting propositional logic. The results are shown in Table 4.5. Most students considered themselves novices or lower for both sessions.

Proficiency Level	**10-2019**	**01-2020**
Fundamental Awareness (basic knowledge)	23 %	32 %
Novice (limited experience)	46 %	53 %
Intermediate (practical application)	31 %	16 %
Advanced (applied theory)	0 %	0 %
Expert (recognized authority)	0 %	0 %

Table 4.5: Proficiency levels of participants

[3]The complete, unabridged and un-redacted answers are available on request.

Number of problems attempted

As can be seen from Table 4.6 most participants solved between 6 and 10 problems during the 1 hour session.

Number of exercises	10-2019	01-2020
<5	8 %	10 %
6-10	77 %	74 %
>10	0 %	10 %
Unspecified	15 %	6 %

Table 4.6: Number of exercises attempted by participants

Ease of entering steps

The ease of entering steps may influence the desire or participants to take "shortcuts" and apply Heuristic Steps. However, most participants did not feel like this posed much of a challenge and considered it easy as Table 4.7 shows. The keyboard shortcuts were perceived as helpful.

Level of ease	10-2019	01-2020
Very easy	8 %	5 %
Easy	46 %	63 %
Not particularly hard or easy	31 %	32 %
Hard	8 %	0 %
Very Hard	8 %	0 %

Table 4.7: Level of ease of entering solution steps

Unexpected rejected steps

The first experiment was done with a version of LogEx in which the Multiple Double Negation Heuristic Step was implemented, but none of the others. This lead to false expectations by the participants with many rejected steps as a result. During the second session a lot more Heuristic Steps were supported, so less solution steps were (unexpectedly) rejected, as can be seen in Table 4.8.

Level of ease	10-2019	01-2020
No, not at all	25 %	63 %
Yes, once or twice	50 %	26 %
Yes, on multiple occasions	25 %	11 %

Table 4.8: Rejected valid steps

Usage of Multiple Double Negation, Implication and Equivalence

Table 4.9 shows that most participants made use of the Homomorphic Sub-formulae Heuristic Step. This is especially true in the second experiment, where more variants of that Heuristic Step were implemented. These results are supported by the data in the log files. These show that in the first experiment nine out of thirteen participants entered one or more steps using the "multiple double negation"-rules. The second experiment nineteen out of 23 participants did.

Used heuristic step	**10-2019**	**01-2020**
Yes	58 %	78 %
No	25 %	11 %
Not sure	17 %	11 %

Table 4.9: Usage of multiple double negations heuristic step

Usage of DeMorgan en Double Negation

Since the Granularity Based Heuristic Step "DeMorgan and Double Negation" was not implemented in the version of LogEx used in the first experiment, this question was only asked in the second survey, shown in Table 4.10. A small number of students made use of this Heuristic Step, which is explainable since it is more advanced in nature than the Homomorphic Sub-formulae are. Data shows that nine out of 23 participants entered one or more steps containing the `demorgan.doubleneg` rule.

Used heuristic step	**01-2020**
Yes	21 %
No	63 %
Not sure	16 %

Table 4.10: Usage of DeMorgan and double negation in one go (only in second experiment).

Usefulness of the ITS

Table 4.11 show that overall, the participants considered LogEx a valuable tool in aiding their understanding of propositional logic. In the second experiment the value of the tool was considered higher than in the first experiment. This could be due to the fact that more Heuristic Steps were available.

Overall satisfaction

As can be seen in Table 4.12 most participants are satisfied with their overall experience using the tool. Though the average score is somewhat higher, and there are no negative outliers, there does not seem to be a clear indication the students in the second experiment

Level of ease	10-2019	01-2020
Not at all	8 %	5 %
A little bit	54 %	63 %
A lot	38 %	42 %

Table 4.11: Contribution to understanding of propositional logic

	10-2019	01-2020
Average	7,3	8,0
Mean	8	8
Mode	8	8

Table 4.12: Average, mean and mode for the given grades

were more satisfied than those in the first. The distribution of the scores is similar as can be seen in Tables 4.13 and 4.14.

The answers to the open questions have been omitted here.

4.3.2. On Performance

To assess the performance impact of the newly implemented rules, a rudimentary statistical analysis was done using the log-data from the four experiments conducted. The "before"-situation is comprised of the "logex-hints" and "logex-nohints" sets. None of the changes were made in LogEx to support Heuristic Steps. The "exp1"-set contained some (see also Section 5.2.2), the "exp2"-set was from the latest version. All values are in Seconds [4].

Median, Mode, Max and Min

The Median response time for each set of logs was calculated by taking the average of the middle two records (when the number of records was even), or the value of the middle one (when odd) in a sorted list of response times. The following query does this calculation [5]:

```
SELECT AVG(responsetime)
FROM (
SELECT responsetime
FROM requests
ORDER BY responsetime
WHERE source=<log-set-name>
LIMIT 2 - (
SELECT COUNT(*)
FROM requests
```

[4] https://hackage.haskell.org/package/time-1.9.2/docs/Data-Time-Clock.html#:NominalDiffTime retrieved April 20,2020

[5] From: https://stackoverflow.com/a/15766121/1280810 retrieved April 20, 2020

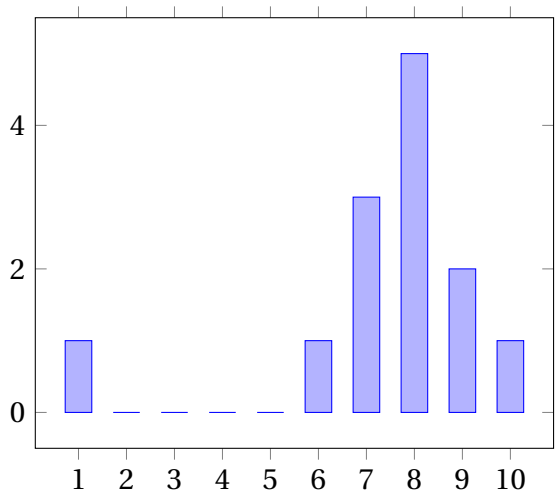

Table 4.13: Overall satisfaction scores for the LogEx tooling, 1st session

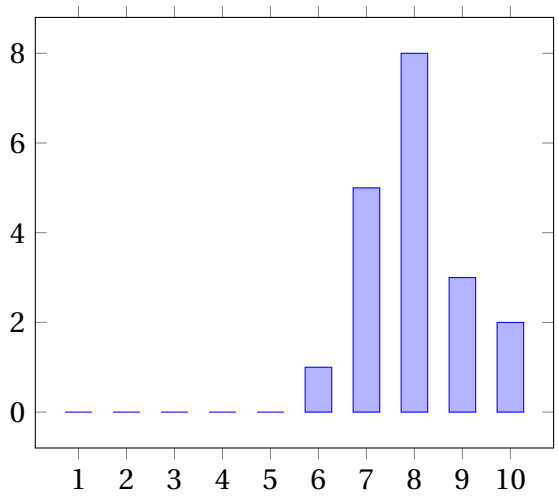

Table 4.14: Overall satisfaction scores for the LogEx tooling, 2nd session

```
WHERE source=<log-set-name>
) % 2    -- odd 1, even 2
OFFSET (
SELECT (COUNT(*) - 1) / 2
FROM requests
WHERE source=<log-set-name>
)
)
```

Since most response times are unique, they were rounded to four decimals before calculating the Modes:

```
select round(responsetime,4), count(*)
from requests
group by round(responsetime,4)
order by count(*) desc
```

The minimal and maximal log values were determined by using the `MAX` and `MIN` functions in SQLite. It has to be noted that the log set of "exp1" contains a number of corrupted records, in which the response time was either `NULL` or garbage data. These records where filtered out before doing the calculations.

Mean response time and standard deviation

The mean response time gives an indication for the general performance of the system. It was calculated using the `AVG`-function of SQLite. The Variance was also calculated, using the Mean-value and the following query:

```
select ( sum( (responsetime-<average>) * (responsetime-<average>))
/ (count(*)-1) )
from requests
```

The Standard Deviation (SD) was calculated by taking $\sqrt{\text{Variance}}$. Table 4.15 shows these values for each of the different log sets.

	logex-hints	**logex-nohints**	**exp1**	**exp2**
Max	0.120186	0.122776	0.339796	0.847603
Min	0.004494	0.004823	0.001350	0.001993
Mean	0.013632	0.016933	0.014189	0.016144
Median	0.010104	0.011517	0.009213	0.012167
Mode	0.0049	0.0053	0.0021	0.0022
Variance	0.000132	0.000186	0.000739	0.000823
SD	0.011481	0.013656	0.027176	0.028689

Table 4.15: Response times in different log-sets of LogEx (rounded)

PERFORMANCE IMPACT

Based on the numbers in Table 4.15, it seems that the addition of Heuristic Steps does not have a large impact on the average response time for LogEx. The lower mode seems to indicate that most requests perform a bit faster although the difference is very small. The data shows that the Standard Deviation for “exp1” and “exp2” is roughly twice as high as those of the earlier experiments, so the performance behaviour is perhaps a little bit more erratic. There could be any number of explanations for this, ranging from code changes in the IDEAS framework, server load, or other infrastructural differences. No effort was made to create comparable circumstances for the different experiments.

Since the first two experiments were conducted using an older version of the Ideas-framework and development work on LogEx has also continued, it is hard to say the differences in performance are directly linked to the implementation of Heuristic Steps. The numbers show however that their inclusion does not seriously hamper the average response time of LogEx.

5

Implementation Details

This chapter contains technical details on the cleaning up and processing of the Logdata as well as details on the different implementation considerations made in the prototype. The chapter can be skipped (or skimmed) by those with less interest in the technological underpinnings.

5.1. Log data Processing

Logfiles Encoding issues

The logs gathered are stored in a SQLite database, which is an open-source file-based database system and library. [1] Unfortunately the data contained in the log file was encoded incorrectly, resulting in poorly readable results, as shown in Figure 5.1

Before any attempt was made to process the logs, the data had to be converted into a properly encoded version. It turned out the data was encoded using ISO 8859-1 (also known less formerly as Latin-1) which does not allow Unicode characters used for the connectives and negations in the equations. As can be seen in Figures 5.2 and 5.3, the tooling used can correctly display those characters after the conversion.

Log Files Process and tooling

In order to find the heuristic steps occurring in the log files, each line, representing a single step in a solution, had to be examined for evidence of heuristic steps. Next, we could assign a category to each line as an indication of the presence or absence thereof. An overview of the different categories assigned and their meaning is shown in Table 5.1.

[1] https://www.sqlite.org/index.html, retrieved October 14, 2019

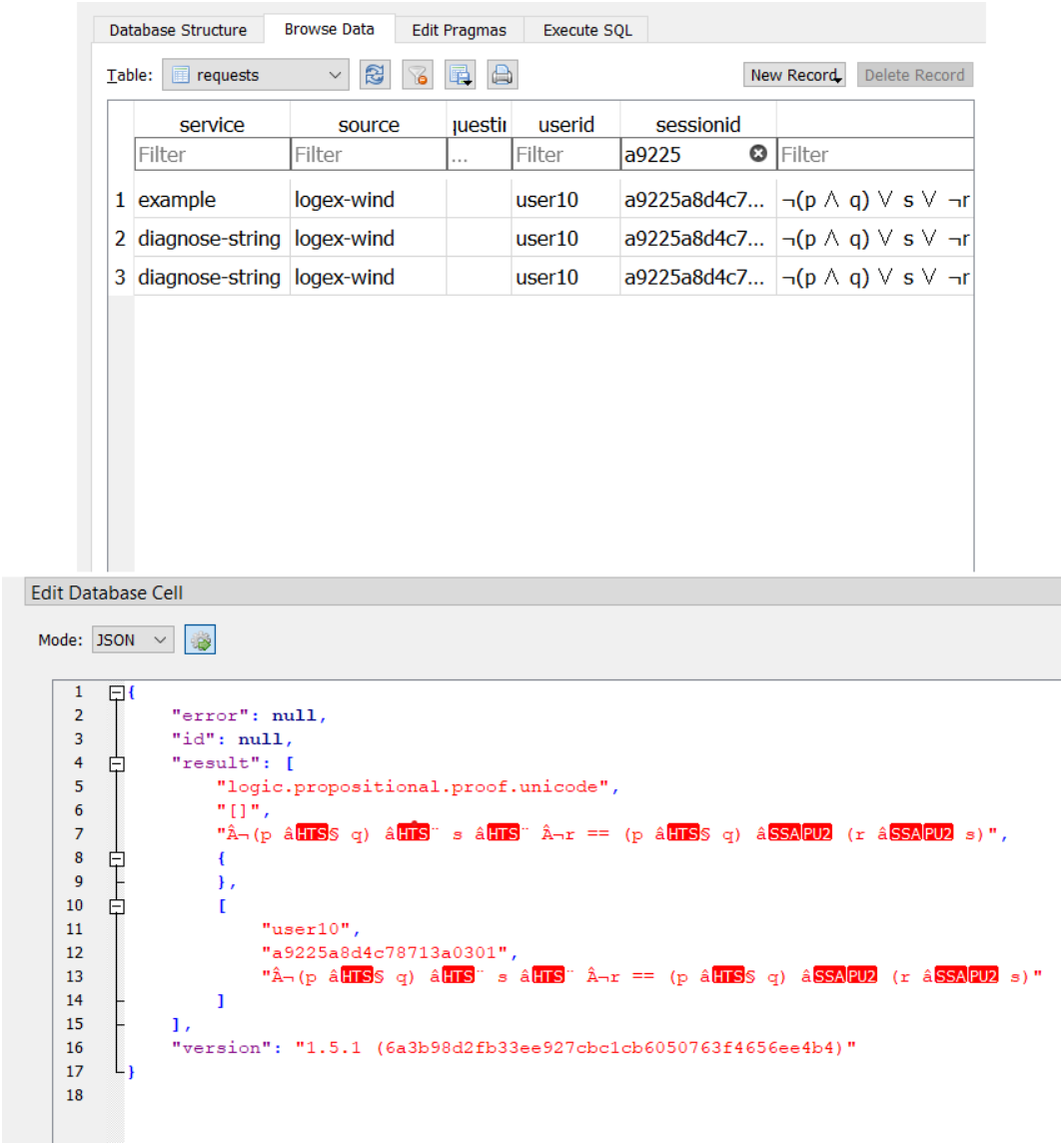

Figure 5.1: Screenshot DB Browser for SQLite showing a sample of the LogEx log

Category	Meaning
Irrelevant	This entry in the log is invalid because it contains only errors, single-line entries, or steps supplied by the NextStep-service
LookedAt	This is a valid log entry, but no evidence of heuristics steps is found in it
Relevant	This log entry contains evidence of a heuristic step being used.

Table 5.1: Different categories for log entries.

Some of these classifications were done automatically using SQL, such as the single-line entries, which were filtered out using the following query:

```
update requests set status = 'Irrelevant' where sessionid in
(
select sessionid s from
```

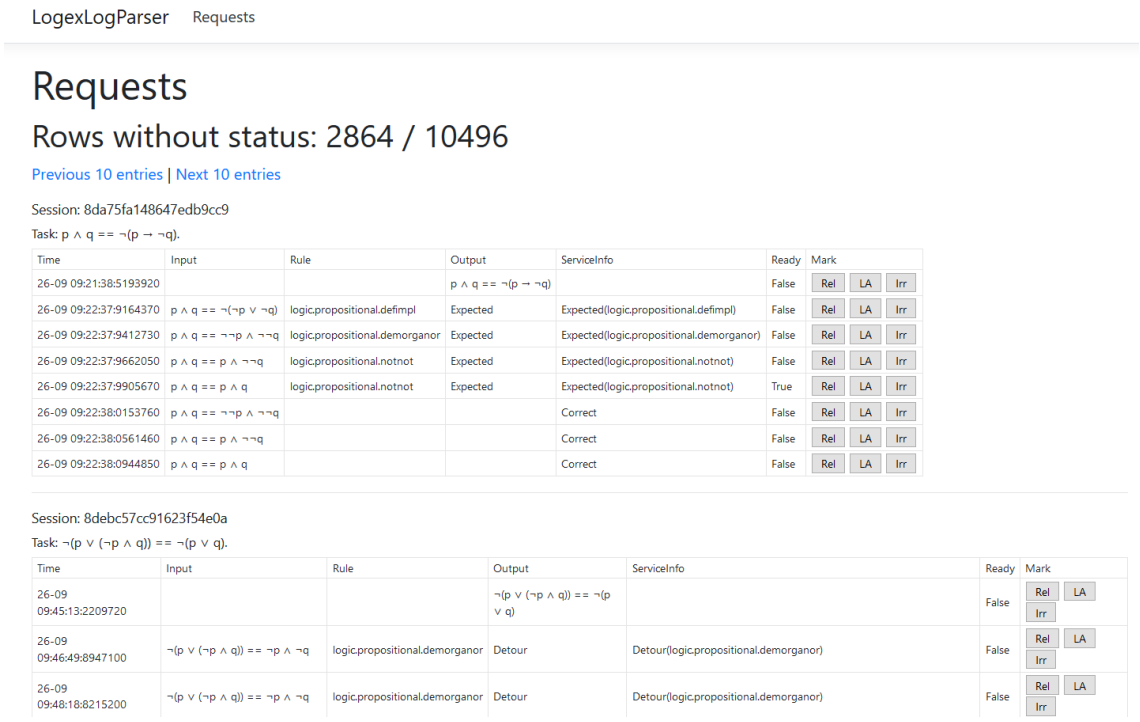

Figure 5.2: Screenshot of the LogParser tool showing two Tasks

```
(
select sessionid, count(*) as cntRows
from requests
where status is null
group by sessionid
)
where cntRows = 1
)
```

The other entries had to be manually processed. In order to facilitate this process, a tool was developed using .NET-Core Razor Pages [2]. The web based tool, which is available online [3] made it easier to mark entries in the log. The Log Parser retrieves its information from the detailed JSON-results that are extracted from the log file.

The interface of the tool is shown in Figure 5.2. As can be seen, the buttons on the right side of the application allow the reviewer to classify a step.

After evidence of a heuristic step was found, the step was marked as “Relevant”, as can be seen in Figure 5.3. This screenshot also shows that this particular user has attempted to resolve the two implications in the original exercise (the first line) in a single step (resulting in the second line). Although the LogEx tool detected the equivalence of both lines (as can be seen from the “Correct” message), it was unable to detect a correctly applied rewriting

[2] https://docs.microsoft.com/en-us/aspnet/core/razor-pages/?view=aspnetcore-3.0&tabs=visual-studio, retrieved October 14, 2019

[3] https://logexlogparser.azurewebsites.net/requests

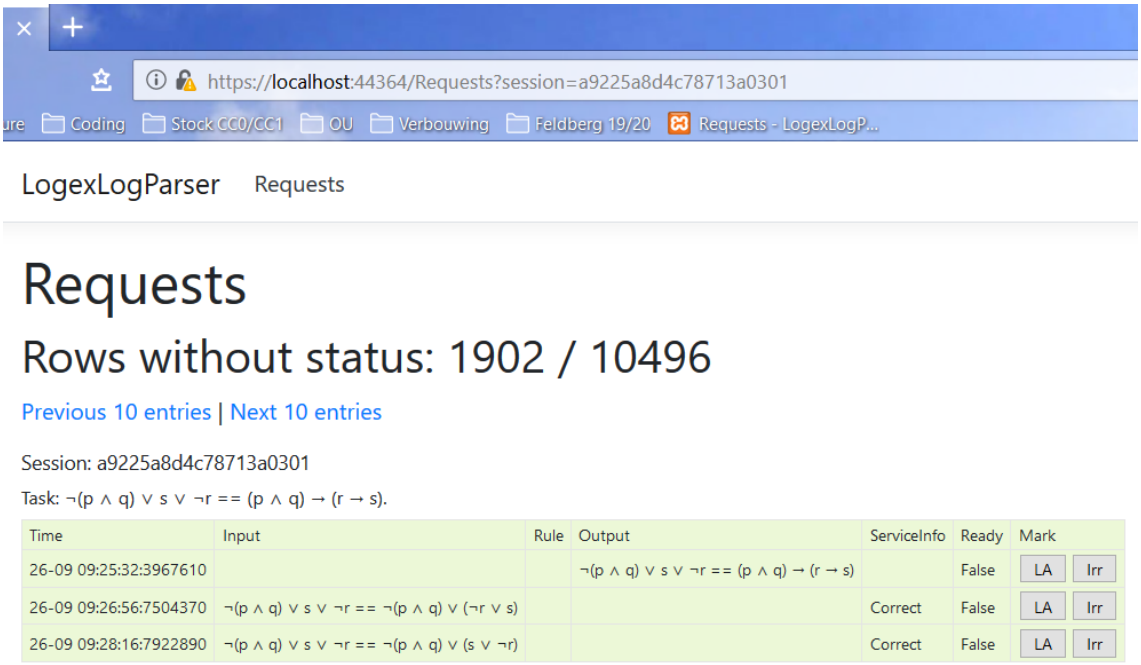

Figure 5.3: Screenshot of the LogParser with evidence showing

rule (as can be seen from the empty “Rule” column).

5.2. Implementation Details

The following section contains additional, mostly technical, details on the different implementation approaches.

5.2.1. Client-Only Implementation

The first approach that was considered was to change only the web-application that calls the LogEx services.

- **logic.allfirst:** We could simply build support into the LogEx web client by allowing it to call the “allfirst” service twice in a row to see if two subsequent steps match the input the student has given. This will only allow us to detect Heuristic Steps that are part of the ideal solution strategy in the first place.
- **logic.allapplications:** To allow any two subsequent steps to be matched, we could also use the “allapplications”-service, which tests the result of all rules in the system on the current formula. There are currently 44 rules in LogEx. Using this approach would lead to $44^2 = 1936$ additional checks and might lead to performance issues. If the number of steps we want to check increases, the state-space will grow exponentially.

A major downside to this approach is that all logic moves to the frontend of the system. This would be a breach of the service-oriented architecture used by IDEAS/LogEx. Furthermore, the number of calls made by the frontend to the different services would increase, as would presumably, latency and loading times. Finally, there would not be any usage of the flexible model the IDEAS-framework offers, nor of the testing facilities already available.

5.2.2. Writing rules for Heuristic Steps

A second solution that was attempted was to capture all Heuristic Steps into "regular" rewrite Rules, thus requiring very few adjustments to LogEx code.

Rules in LogEx are defined as Haskell lambda expressions:

```
ruleDoubleNeg :: Rule SLogic
ruleDoubleNeg = ruleFor groupDoubleNegation "NotNot" $
\x -> Not (Not x)  :~>  x
```

The rule is labelled ("NotNot" in this case) and made part of a group of related rules. Finally the actual rewrite is given. In this expression x is a meta-variable and can be any formula.

In the first implementation used for the first experiment (*exp1* in Section 3.4.3) a number of Heuristic Steps were implemented as simple rewrite rules:

```
ruleDoubleDoubleNegAnd :: Rule SLogic
ruleDoubleDoubleNegAnd =
ruleFor groupDoubleNegation "DoubleNotNotAnd" $
\x y -> Not (Not x) :&&: Not (Not y) :~> x :&&: y
```

This allows student to use the *multi.doubleneg.and* rule in their solutions. By adding it to the *groupDoubleNegation* the rule falls under the *Double Negation* umbrella and student don not even have to indicate whether or not they use the singular of multiple version of the rule. During the experiment students were actually able to use this Heuristic Step and made frequent use of it in their solutions.

For many Heuristic Steps, especially those in the Homomorphic Sub-formulae category we can easily set up additional rules, as shown in Equations 5.1 and 5.2.

$$\text{DoubleNotNot:} \quad \neg\neg\phi \square \neg\neg\psi \Longleftrightarrow \phi \square \psi \tag{5.1}$$

$$\text{DoubleDefImpl:} \quad \phi \rightarrow \psi \square \chi \rightarrow \omega \Longleftrightarrow \neg\phi \vee \psi \square \neg\chi \vee \omega \tag{5.2}$$

The major downside to this approach is that for every possible variant of the Heuristic Step that needs to be supported, a new Rule has to be defined. Not only for each support binary connective, but also for the number of double negations that could occur in a given formula.

SIDESTEP: TEMPLATEHASKELL

One of the more interesting possible solution to this downside was to generate different permutations of expressions dynamically at compile time using TemplateHaskell (for two occurrences of the same pattern, for three etc.) [SP02].

```
myFunc :: Q Exp
myFunc = do
x <- newName "x" -- generate a unique variable name, we'll cover names
    later
y <- newName "y"
return $ LamE              -- lambda expression
[VarP x, VarP y]        -- pattern matching on 'x'
(InfixE (Just (VarE x)) (VarE '(:&&:)) (Just (VarE y)))
```

Eventually this route is considered needlessly complicated and unfeasible within the time frame of the project.

5.2.3. PARTIAL STRATEGIES FOR HEURISTIC STEPS

The approach that best seems to follow the framework is to define partial strategies for the different Heuristic Steps. In order to allow them to be slotted in between the "normal" rules, the Strategies are collapsed into Rules.

HOMOMORPHIC SUB-FORMULAE

First we define a strategy that simply consists of applying a particular rule one or more times, using `repeat1` and `somewhere`. We're using a Haskell idiom called *Zipper*, which allows us to bring part of some data structure into focus, or *context*, to operate on it [Hue97]. This is what `liftToContext` does in the following code:

```
multipleApply :: Rule SLogic -> Strategy (Context SLogic)
multipleApply r = repeat1 $ somewhere $ liftToContext r
```

Now we can use this generic strategy to implement all Heuristic Steps in the Homomorphic Sub-formulae category. Again using the application of Multiple Double Negations as an example, we could formulate this as follows:

```
multipleDoubleNeg :: Strategy (Context SLogic)
multipleDoubleNeg =  liftToContext ruleDoubleNeg
>|> multipleApply ruleDoubleNeg
```

Here we make a Strategy that either uses the original `ruleDoubleNeg` or (`>|>`) [4] the multiple application of this Rule. The left-preference choice is used here so in example derivations and next-step hints LogEx will provide the single application of the Rule by default.

[4] This is written as .\. in the current version of IDEAS.

All we have to do now is to collapse this Strategy into a Rule so it can be used as a solution-step:

```
ruleMultiDoubleNeg :: Rule (Context SLogic)
ruleMultiDoubleNeg = siblingOf groupDoubleNegation $
collapseToRule "multi.doubleneg" multipleDoubleNeg
```

Here, the *siblingOf* function is used to add the Rule to the *Double Negation*-group. We then simply call *collapseToRule* providing a name and our strategy:

```
collapseToRule :: String -> Strategy a -> Rule a
collapseToRule name strategy = makeRule name (applyAll strategy)
```

Here, `applyAll` is again a Zipper that will attempt to apply the specified *Strategy* to every part of the formula. In this way, Multiple Implication Elimination and Multiple Equivalence Elimination have been defined.

Granularity Based Rules

To implement the Heuristic Steps of the Granularity Based type, multiple rules have to be combined into a strategy, which then has to be collapsed to a Rule once again. For this, the same mechanisms used for the Homomorphic Sub-formulae can be used.

```
deMorganAndDoubleNegStrategy :: Rule SLogic -> Strategy (Context SLogic
    )
deMorganAndDoubleNegStrategy dm =
(somewhere (liftToContext dm))
<*>
(multipleApply ruleDoubleNeg)
```

First we make a Strategy from one of the DeMorgan-rules where we apply a sequence (<*>) [5] DeMorgan *somewhere* followed by multiple applications of the *ruleDoubleNeg*. After that we simply use *collapseToRule* function to turn our strategy into a rule, specifying the *ruleDeMorganAnd* and *ruleDeMorganOr*:

```
ruleDeMorganAndDoubleNeg :: Rule (Context SLogic)
ruleDeMorganAndDoubleNeg = siblingOf groupDeMorgan $
collapseToRule "demorganand.doubleneg" (deMorganAndDoubleNegStrategy
    ruleDeMorganAnd)
```

Housekeeping

A number of Housekeeping rules was already available in LogEx, such as the removal of excessive parentheses. Both of the following partial rewrites are accepted:

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

[5] This is written as .*. in the current version of IDEAS.

This examples shows De Morgan being applied very strictly. The outer parentheses can be removed without any problem:

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

No other housekeeping Heuristic Steps were part of this prototype implementation.

6

THREATS TO VALIDITY

While the findings are promising, there are potential threats to their validity.

6.1. EXTREME BIAS IN THE TEST GROUP.

The group of students that participated in the experiments were not chosen at random. They volunteered for the experimental sessions. These were students with an interest in the subject matter or at least a desire to improve their skills.

How this influences the results is hard to predict, but the bias is unmistakeably there. If true and reliable data is needed, a full Randomized Control Trial (RCT) should be conducted, preferably with at least two sets of students, one of which has to solve the problems having Heuristic Steps at their disposal and one group without. That way the effects of the newly implemented features can be assessed much better.

6.2. BIAS IN EXERCISES

In both experiments the students were given a predefined list of exercises they could solve. These exercises were chosen in a way that would make the usage of Heuristic Steps more attractive. An experiment with completely random exercises would provide a much more realistic usage percentage for Heuristic Steps and might potentially yield new types that were not present in the log sets analysed in this project.

6.3. SMALL SAMPLE SIZE

The survey data gathered has not been analysed using statistical methods in any way due to very small sample size. This makes any finding potentially unreliable. An experiment

with a much larger sample size could improve this.

6.4. Current prototype is not formally proven correct

No formal proof of the correctness and completeness of the current implementation of Heuristic Steps is done. Not even the informal checks that are in place (i.e. the test-cases) have been checked with regards to code coverage. There might well be edge cases that give erroneous or unexpected results.

6.5. Performance statistics are to general

The performance statistics used in this thesis have been taken from all service calls. This might lead to a skewed view on actual performance, since many of services do not use the Heuristic Step code, such as the one to generate a new exercise. Performance was only measured from a usability standpoint and not quantified beforehand, so the current numbers can be seen as an indication that the addition of Heuristic Steps cause no performance problems, in general.

6.6. Lack of feedback might lead to under-use

No feedback specifically on Heuristic Steps is generated by the prototype. This may result in being underused. The current approach to not use the Heuristic Steps in the “Next Step”-hint might lead to students being unaware of the existence of them. Since no “Buggy rules” have been implemented for Heuristic Steps, it might also be the case that students attempted to apply a Heuristic Steps but made an error. They only got back generic feedback and altered their solution strategy to exclude heuristic steps.

7

CONCLUSIONS AND RECOMMENDATIONS

7.1. CONCLUSIONS

The main research question "How can heuristic steps in solutions for proposition logic rewriting exercises in the LogEx tutoring system be detected?" was subdivided into three sub-questions. A brief summary of the answer to each question is listed here:

1. **What are common Heuristic Step taken by students and how can they be classified?** Log and homework analysis have indicated that there are at least three distinct types of Heuristic Steps taken by students: those based on Homomorphic Sub-formulae, Granularity-Based and Housekeeping. We have focussed most on the first two categories since the data shows those are used frequently by novices using the LogEx tutoring system.

2. **How can we describe Heuristic Steps in an ITS?** We have described a number of Heuristic Steps using the Domain Specific Language for Strategies available in the IDEAS-framework. Using this DSL, we were able to efficiently add a number of new possibilities to the current set of solution steps available to students. The new additions have been proven to work and are used frequently in experimental sessions.

3. **How can we detect Heuristic Steps in entered solution?** Since we're using the mechanics built-in to the IDEAS-framework, detection comes "out-of-the-box". The existing categories (e.g. DeMorgan-rules or Double Negation-rules) have simply been extended with our new rules.

Heuristic Steps can be detected in the LogEx tutoring system by implementing them as Partial Strategies which are collapsed into Rules. The main types of Heuristic Steps have been identified and when available will be used by users of the ITS.

7.2. Recommended Further Research

7.2.1. Heuristic Steps may promote sloppy work

Analysis on logs seems to suggest that when the Heuristic Steps are available they are used a lot. How will this affect student performance? Having Heuristic Steps at their disposal, a student might skip over steps they do not fully comprehend. Fact is that in the current version of LogEx, each and every step entered simply contains a single rewrite action. This makes it easy for a student to learn them at a very basic level (the smallest grain size possible in terms of granularity).

Being able to skip over certain steps without full comprehension might lower the learning effect of the tutoring system. The RCT mentioned earlier in Chapter 6 might be provide insight into what the effect on the students' learning is of Heuristic Steps.

A randomized trial to compare student's performance with and without Heuristic Steps should be performed analogous to the one performed by Lodder e.a. for previous versions of LogEx [LHJ19].

7.2.2. Student Model

Incorporating a Student Model into the LogEx tutoring system would be a considerable improvement. The inner feedback loop is covered well, and each exercise in itself can be solved using hints and appropriate feedback [Van11]. Taking the student's achievements and performance into account when generating new exercises might increase the value of the tutor. Knowing how far a long a student is, can make the system even more dynamic by allowing access to additional Heuristic Steps, once a certain skill has been adequately proven. For Granularity Based Heuristic Steps this would be especially important, since students are in effect skipping over certain steps they may not have completely understood. The research into and implementation of such a Student Model would be a strong recommendation.

7.2.3. Feedback on heuristic steps

Providing feedback on Heuristic Steps has not been investigated in the context of this project, but is very important. Most notably, hints could be given about existing Heuristic Steps, or they could be used as potential "next steps" by LogEx itself. In the current version the non-heuristic variants or rules have been given precedence over the heuristic ones, but with proper feedback, this need not be the case. Looking into potential "buggy" Heuristic Steps could also be a valuable addition in this regard.

7.2.4. Different types of Heuristic Steps in different domains

The categories for Heuristic Steps found in this thesis might only apply to propositional logic and can not be extrapolated to other domains. While this does not invalidate the findings, it diminishes their applicability. In order to find out if the categories translate into other domains, different types of ITSs can be analysed in a manner similar to the one described in this thesis. A more universally applicable categorisation of Heuristic Steps may be found.

7.2.5. Performance Analysis

The new additions made to LogEx do not seem to have an adverse effect on the performance of the tool. No real evidence of any decrease in responsiveness of the system has been witnessed, although the analysis was not very extensive. Adding rules to the system increases the solution space, but by what amount? A rigorous analysis of the performance impact of the new Strategy-as-rules has to be done, taking into account traversal strategies in the IDEAS framework [1].

[1] Bastiaan Heeren, "Traversals with class". In *Een Lawine van Ontwortelde Bomen*. Universiteit Utrecht, 2013

8

ACKNOWLEDGEMENT

While doing a research project and writing a thesis is often lonely work, especially during a pandemic with mandatory stay-at-home regulations in place, it would be unfair to take all the credit for it. This endeavour couldn't have been completed without guidance and help from others:

1. First and foremost I want to thank Josje Lodder for being my supervisor. Our bi-weekly meeting were short and to the point, but enjoyable nevertheless. Your feedback and knowledge about LogEx were invaluable and I hope my work is of value to you when you finish up your doctoral;

2. Bastiaan Heeren proved to be a true fountain of knowledge on all things Haskell and IDEAS. His keen eye for detail and utter thoroughness in reviewing my work greatly improved its quality;

3. The students who were kind enough to let me use their homework submissions contributed to the validity of the typology;

4. My employer Fontys Hogescholen was kind enough to provide time and funding which made it possible to finish the project in the first place;

5. And last, but most certainly not least, my family and friends, who were kind enough to tolerate my moaning, obsessive late-night LaTeX-sessions and missed quality time. They have supported me throughout the entire process.

Thank you all so very, very much.

Nuenen, July 2020

BIBLIOGRAPHY

[Ale+06] Vincent Aleven, Bruce M. McLaren, Jonathan Sewall, and Kenneth R. Koedinger. "The Cognitive Tutor Authoring Tools (CTAT): Preliminary Evaluation of Efficiency Gains". In: *Intelligent Tutoring Systems*. Ed. by Mitsuru Ikeda, Kevin D. Ashley, and Tak-Wai Chan. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 61–70. ISBN: 978-3-540-35160-3.

[Ben+14] JFAK van Benthem, HP van Ditmarsch, JS Lodder, J Ketting, and WPM Meyer-Viol. *Logica voor informatica*. Open Univeristeit, Heerlen, 2014.

[BO11] Edward C. Benzel and R. Douglas Orr. "A steep learning curve is a good thing!" In: *The Spine Journal* 11.2 (2011), pp. 131–132. ISSN: 1529-9430. DOI: https://doi.org/10.1016/j.spinee.2010.12.012. URL: http://www.sciencedirect.com/science/article/pii/S152994301001449X.

[DL07] Ludmila Dostálová and Jaroslav Lang. "ORGANON — The Web Tutor for Basic Logic Courses". In: *Logic Journal of the IGPL* 15.4 (Aug. 2007), pp. 305–311. ISSN: 1367-0751. DOI: 10.1093/jigpal/jzm021. eprint: http://oup.prod.sis.lan/jigpal/article-pdf/15/4/305/1709324/jzm021.pdf. URL: https://doi.org/10.1093/jigpal/jzm021.

[Ger12] Alex Gerdes. "AskElle: a Haskell tutor". PhD thesis. Open Universiteit Nederland, 2012.

[GM89] Jim E Greer and Gordon I McCalla. "A Computational Framework for Granularity and its Application to Educational Diagnosis." In: *IJCAI*. 1989, pp. 477–482.

[HJ14] Bastiaan Heeren and Johan Jeuring. "Feedback services for stepwise exercises". In: *Science of Computer Programming* 88 (2014), pp. 110–129. ISSN: 01676423. DOI: 10.1016/j.scico.2014.02.021. URL: http://dx.doi.org/10.1016/j.scico.2014.02.021.

[HJ17] Bastiaan Heeren and Johan Jeuring. "An Extensible Domain-Specific Language for Describing Problem-Solving Procedures". In: *Artificial Intelligence in Education*. Ed. by Elisabeth André, Ryan Baker, Xiangen Hu, Ma. Mercedes T. Rodrigo, and Benedict du Boulay. Cham: Springer International Publishing, 2017, pp. 77–89. ISBN: 978-3-319-61425-0.

[Hob90] Jerry Hobbs. "Granularity". In: *Readings in qualitative reasoning about physical systems*. Elsevier, 1990, pp. 542–545.

[Hue+11] Antonia Huertas, Josep M Humet, Laura López, and Enric Mor. “The SELL project: a learning tool for e-learning logic”. In: *International Congress on Tools for Teaching Logic*. Springer. 2011, pp. 123–130.

[Hue97] Gérard Huet. “Functional pearl”. In: *J. functional programming* 7.5 (1997), pp. 549–554.

[KF15] James A. Kulik and J. D. Fletcher. “Effectiveness of Intelligent Tutoring Systems”. In: *Review of Educational Research* 86.1 (2015), pp. 42–78. ISSN: 0034-6543. DOI: 10.3102/0034654315581420.

[KHJ17] Hieke Keuning, Bastiaan Heeren, and Johan Jeuring. “Code Quality Issues in Student Programs”. In: *Proceedings of the 2017 ACM Conference on Innovation and Technology in Computer Science Education*. ITiCSE ’17. Bologna, Italy: Association for Computing Machinery, 2017, pp. 110–115. ISBN: 9781450347044. DOI: 10.1145/3059009.3059061. URL: https://doi.org/10.1145/3059009.3059061.

[LHJ15] Josje Lodder, Bastiaan Heeren, and Johan Jeuring. “A pilot study of the use of LogEx, lessons learned”. In: *Technical Report* (2015), pp. 1–8. ISSN: 18688969. DOI: 10.4230/LIPIcs.xxx.yyy.p. arXiv: 1507.03671v1. URL: http://arxiv.org/abs/1507.03671v1.

[LHJ16] Josje Lodder, Bastiaan Heeren, and Johan Jeuring. “A domain reasoner for propositional logic”. In: *Journal of Universal Computer Science* 22.8 (2016), pp. 1097–1122. ISSN: 09486968.

[LHJ19] Josje Lodder, Bastiaan Heeren, and Johan Jeuring. “A comparison of elaborated and restricted feedback in LogEx, a tool for teaching rewriting logical formulae”. In: *Journal of Computer Assisted Learning* (June 2019). DOI: 10.1111/jcal.12365.

[Ma+14] Wenting Ma, Olusola O. Adesope, John C. Nesbit, and Qing Liu. “Intelligent tutoring systems and learning outcomes: A meta-analysis.” In: *Journal of Educational Psychology* 106.4 (2014), pp. 901–918. ISSN: 0022-0663. URL: http://search.ebscohost.com.ezproxy.elib11.ub.unimaas.nl/login.aspx?direct=true&db=pdh&AN=2014-25074-001&site=ehost-live.

[MB17] Behrooz Mostafavi and Tiffany Barnes. “Evolution of an Intelligent Deductive Logic Tutor Using Data-Driven Elements”. In: *International Journal of Artificial Intelligence in Education* 27.1 (Mar. 2017), pp. 5–36. ISSN: 1560-4306. DOI: 10.1007/s40593-016-0112-1. URL: https://doi.org/10.1007/s40593-016-0112-1.

[MG94] Gordon I. McCalla and Jim Greer. “Granularity-Based Reasoning and Belief Revision in Student Models”. In: *Student Modelling: The Key to Individualized Knowledge Based Instruction*. Ed. by Jim Greer and Gordon McCalla. Berlin, Heidelberg: Springer Berlin Heidelberg, 1994, pp. 39–62. ISBN: 978-3-662-03037-0.

[Mos+03] Jack Mostow, Greg Aist, Paul Burkhead, Albert Corbett, Andrew Cuneo, Susan Eitelman, Cathy Huang, Brian Junker, Mary Beth Sklar, and Brian Tobin. “Evaluation of an automated reading tutor that listens: Comparison to human tutoring and classroom instruction”. In: *Journal of Educational Computing Research - J EDUC COMPUT RES* 29 (Oct. 2003), pp. 61–117. DOI: 10.2190/06AX-QW99-EQ5G-RDCF.

[Nar13] Susanne Narciss. “Designing and Evaluating Tutoring Feedback Strategies for digital learning envi”. In: 23 (2013), pp. 7–26. ISSN: 20139144.

[Pra14] Rein Prank. “A tool for evaluating solution economy of algebraic transformations”. In: *Journal of Symbolic Computation* 61-62 (2014), pp. 100–115. ISSN: 0747-7171. DOI: https://doi.org/10.1016/j.jsc.2013.10.014. URL: http://www.sciencedirect.com/science/article/pii/S0747717113001338.

[SP02] Tim Sheard and Simon Peyton Jones. “Template meta-programming for Haskell”. In: *Proceedings of the 2002 Haskell Workshop, Pittsburgh*. Oct. 2002, pp. 1–16. URL: https://www.microsoft.com/en-us/research/publication/template-meta-programming-for-haskell/.

[Van11] Kurt VanLehn. “The Relative Effectiveness of Human Tutoring, Intelligent Tutoring Systems, and Other Tutoring Systems”. In: *Educational Psychologist* 46.4 (2011), pp. 197–221. DOI: 10.1080/00461520.2011.611369. eprint: https://doi.org/10.1080/00461520.2011.611369. URL: https://doi.org/10.1080/00461520.2011.611369.

[Yac03] Kalina Yacef. “Experiment and evaluation results of the Logic-ITA”. In: *Technical report / University of Sydney. School of Information Technologies* 542 (2003).

[Yac05] Kalina Yacef. “The Logic-ITA in the classroom: a medium scale experiment”. In: *International Journal of Artificial Intelligence in Education* 15.1 (2005), pp. 41–62.

[ZH10] Marc Zimmermann and Daniel Herding. *Entwicklung einer computergestützten Lernumgebung für bidirektionale Umformungen in der Mengenalgebra*. Universitätsbibliothek Dortmund, 2010.

Appendices

A

Complete Source Code, Tests and Testresults

A.1. Source Code Added to LogEx

A.1.1. Rules.hs

Most of the rules and strategies-as-rules are defined here:

```
1
2  -- Make a rule out of a strategy
3  collapseToRule :: String -> Strategy a -> Rule a
4  collapseToRule n s = makeRule n (applyAll s)
5
6  -- MultiRules
7
8  -- Create a Strategy to apply a particular Rule multiple times
9  multipleApply :: Rule SLogic -> Strategy (Context SLogic)
10 multipleApply r = repeat1 $ somewhere $ liftToContext r
11
12
13 -- Make the multiple application strategy so that at single application
14 -- is preferred above a multiple application.
15 multipleDoubleNeg :: Strategy (Context SLogic)
16 multipleDoubleNeg =  (liftToContext ruleDoubleNeg)
17            >|> multipleApply ruleDoubleNeg
18
19 multipleImplication :: Strategy (Context SLogic)
20 multipleImplication = (liftToContext ruleDefImpl)
21                     >|> multipleApply ruleDefImpl
22
23 multipleEquivalence :: Strategy (Context SLogic)
24 multipleEquivalence = (liftToContext ruleDefEquiv)
```

```
                    >|>  multipleApply ruleDefEquiv


-- Make actual nameds rule out of the multiple application strategies
ruleMultiDoubleNeg :: Rule (Context SLogic)
ruleMultiDoubleNeg = siblingOf groupDoubleNegation $
  collapseToRule "multi.doubleneg" multipleDoubleNeg

ruleMultiEquivalences :: Rule (Context SLogic)
ruleMultiEquivalences = siblingOf groupEquivalence $
  collapseToRule "multi.equivalence" multipleEquivalence

ruleMultiImplications :: Rule (Context SLogic)
ruleMultiImplications = siblingOf groupImplication $
  collapseToRule "multi.implication" multipleImplication

-- CombinationRules

-- Make Rules out of the application of DeMorgan and double negation in
    one go.
ruleDeMorganAndDoubleNeg :: Rule (Context SLogic)
ruleDeMorganAndDoubleNeg = siblingOf groupDeMorgan $
  collapseToRule "demorganand.doubleneg" (deMorganAndDoubleNegStrategy
    ruleDeMorganAnd)

ruleDeMorganOrDoubleNeg :: Rule (Context SLogic)
ruleDeMorganOrDoubleNeg = siblingOf groupDeMorgan $
  collapseToRule "demorganor.doubleneg" (deMorganAndDoubleNegStrategy
    ruleDeMorganOr)

ruleDeMorganAndCommunativity :: Rule (Context SLogic)
ruleDeMorganAndCommunativity = siblingOf groupDeMorgan $
  collapseToRule "demorganor.comm" deMorganAndCommunativityStrategy

-- Formulate the strategy that applies (a) DeMorgan rule followed by
    multiple applications of ruleDoublNeg
deMorganAndDoubleNegStrategy :: Rule SLogic -> Strategy (Context SLogic
    )
deMorganAndDoubleNegStrategy dm =
  (somewhere (liftToContext dm))
  <*>
  (multipleApply ruleMultiDoubleNeg)

deMorganAndCommunativityStrategy :: Strategy (Context SLogic)
deMorganAndCommunativityStrategy =
  (somewhere (liftToContext ruleDeMorganAnd))
  <*>
  (somewhere (liftToContext ruleCommOr))
```

A.1.2. STRATGIES.HS

The set of Simplification steps has been extended to include “ruleMultiDoubleNeg”

```
simplifyStep :: LabeledStrategy (Context SLogic)
simplifyStep = label "Simplify" $ oncetdPref $
        orRules <|> andRules <|>
        useRules [ruleNotTrue, ruleNotFalse]
        <|>
        ruleMultiDoubleNeg
```

Both DeMorgan-strategies have been extend to include the combination of DeMorgan and Double Negation:

```
deMorganOr :: Strategy (Context SLogic)
deMorganOr = liftToContext generalRuleDeMorganOr
        >|> liftToContext ruleDeMorganOr
        >|> ruleDeMorganOrDoubleNeg


deMorganAnd :: Strategy (Context SLogic)
deMorganAnd = liftToContext generalRuleDeMorganAnd
        >|> liftToContext ruleDeMorganAnd
        >|> ruleDeMorganAndDoubleNeg
        >|> ruleDeMorganAndCommunativity
```

Finally, the Implication- and Equivalence-elimination strategy has been expanded with out newly defined ones:

```
eliminateImplEquiv :: LabeledStrategy (Context SLogic)
eliminateImplEquiv = label "EliminateImplEquiv" $
      somewhere (liftToContext ruleDefImpl)
      >|>
      ruleMultiImplications
      >|>
      oncebuPref (liftToContext ruleDefEquiv)
      >|>
      ruleMultiEquivalences
```

All newly added rules were added to the Module-description in both files. Those changes are not included here.

A.2. TESTCASES DEFINED

A.2.1. TESTCASES

A set of testcases was defined. They are listed in Tables A.1 and A.2. This mechanism is built-in into the LogEx CGI-service. Before implementation started, there already was a large set of testcases for LogEx. Because quite a few of those were in a failing state, this test set could unfortunately not be used to do regression testing.

Input	Service	Selected Rule	Expected
$\neg(\neg p \wedge \neg q)$	apply	demorganand.doubleneg	$p \vee q$
$\neg(\neg p \vee \neg q)$	apply	demorganor.doubleneg	$p \wedge q$
$\neg(\neg p \vee \neg q)$	diagnose-string	demorganor.doubleneg	$p \wedge q$
$\neg\neg p \wedge \neg\neg q \wedge \neg\neg s$	apply	multi.doubleneg	$p \wedge q \wedge s$
$\neg\neg p \vee \neg\neg q \vee \neg\neg s$	apply	multi.doubleneg	$p \vee q \vee s$
$(\neg\neg p \leftrightarrow \neg\neg q) \vee (\neg\neg s \leftrightarrow \neg\neg r)$	apply	multi.doubleneg	$(p \leftrightarrow q) \vee (s \leftrightarrow r)$
$(\neg\neg p \rightarrow \neg\neg q) \vee (\neg\neg s \rightarrow \neg\neg r)$	apply	multi.doubleneg	$(p \rightarrow q) \vee (s \rightarrow r)$
$(p \leftrightarrow q) \vee (\neg\neg s \leftrightarrow r)$	apply	multi.equivalence	$(p \wedge q) \vee (\neg p \wedge \neg q) \vee (\neg\neg s \wedge r)\|\|(\neg\neg\neg\neg s \wedge \neg r)$
$(p \rightarrow q) \vee (\neg\neg s \rightarrow r)$	diagnose-string	demorganor.doubleneg	$\neg p \vee q \vee \neg\neg\neg s \vee r$

Table A.1: Testcases for different rules

Input	Service	Expected
$\neg\neg p \vee (\neg\neg q -> \neg\neg r)$	allfirsts	$x \vee y$ (multi.doubleneg) $x \vee \neg\neg y$ (multi.doubleneg) $\neg\neg x \vee y$ (multi.doubleneg)
$\neg\neg x \vee \neg\neg y$	allfirsts	$p \vee (q \rightarrow r)$ (multi.doubleneg) $p \vee (\neg\neg q \rightarrow \neg\neg r)$ (multi.doubleneg) $\neg\neg p \vee (q \rightarrow r)$ (multi.doubleneg) $\neg\neg p \vee (q \rightarrow \neg\neg r)$ (multi.doubleneg) $\neg\neg p \vee (\neg\neg q \rightarrow r)$ (multi.doubleneg)

Table A.2: Testcases for “allfirst” service

A.2.2. TESTREPORT

The testcases can be run from the command-line using the following command:

```
./logic.cgi --test=test/bigsteps
```

The output usually only contains details if there are errors during the test-run. Running the testset above returns the following output, showing that all testcases succeeded:

```
Directory bigsteps
.
Directory bigsteps/homomorphicSF
.........
Directory bigsteps/granularity
...
---------------------------------------
Tests :  13
Errors :  0
Warnings :  0

Time :  0.296194843s
```

```
Suites:
Directory bigsteps (tests:  13, errors:  0, warnings:  0, 0.2926422s)
--------------------------------------
```