Equivalence Teaching Tool

Anna Thomas

May 19, 2014

Contents

1 Introduction **4**
- 1.1 Motivation . . . 4
- 1.2 Approach . . . 4
- 1.3 Objectives . . . 5

2 Background **7**
- 2.1 Propositional Logic . . . 7
- 2.2 Truth tables . . . 8

3 Android **9**
- 3.1 Android vs Web app . . . 9
  - 3.1.1 Intuitive to use . . . 9
  - 3.1.2 Learning experience . . . 9
  - 3.1.3 None currently available for Android . . . 9
- 3.2 Tablet vs Phone app . . . 9
  - 3.2.1 Screensize . . . 9
  - 3.2.2 Formation Tree would not feasible on phone app . . . 9
- 3.3 Features . . . 9
  - 3.3.1 Multi-Touch Gestures . . . 9
- 3.4 Market Share . . . 10

4 Related Work **11**
- 4.1 Logic Daemon . . . 11
- 4.2 Pandora . . . 12
- 4.3 Logic Solver . . . 13
- 4.4 Truth Tables . . . 14
- 4.5 LogicCalc . . . 14
- 4.6 Propositional Logic Calculator . . . 15
- 4.7 Previous Year Projects . . . 16

5 Parser **18**
- 5.1 Parser Generation . . . 18
  - 5.1.1 ANTLR4 and alternatives . . . 18
- 5.2 Grammar . . . 18

5.2.1 Tokens . . . 18
5.2.2 Order of Precedence . . . 18
5.2.3 Error Handling - Incorrect grammar syntax to handle errors semantically . . . 18
5.2.4 Extension to First Order Logic . . . 18
5.3 Parser . . . 18
5.3.1 Parser . . . 18
5.3.2 Walker . . . 18

6 Internal Tree Representation 19
6.1 Compiler . . . 19
6.2 Formation Tree . . . 19
6.2.1 Tree operations, eg. adding a new node . . . 19
6.3 Nodes . . . 19
6.3.1 Binary Operators . . . 19
6.3.2 Unary Operators . . . 19
6.3.3 Atoms . . . 19

7 Tree Manipulation 20
7.1 The Rule Engine . . . 20
7.2 Rule Selection - which rules can be applied to a node . . . 20
7.3 Rule Application - how rules are applied to a node . . . 20
7.4 Random Rule Application . . . 20
7.4.1 Generating equivalent formulae . . . 20

8 The App 21
8.1 Truth Tables . . . 22
8.1.1 Ensure equivalences are equal before beginning . . . 22
8.2 Random Generation of Equivalences . . . 22
8.3 Custom Keyboard . . . 22
8.3.1 Logic symbols and variables . . . 22
8.4 Undo/Redo Functionality . . . 22
8.4.1 Stack of trees maintained . . . 22
8.4.2 OnClick listeners . . . 22
8.5 Formation Tree Representation . . . 22
8.5.1 Tree Layout . . . 22
8.5.2 Walker's Algorithm . . . 22
8.5.3 Alternatives . . . 22
8.6 Rule Application . . . 22
8.6.1 How rules are displayed within the app . . . 22
8.7 Difficulty . . . 22
8.7.1 Length of generated equivalences . . . 22
8.7.2 Number of variables/operators . . . 22
8.7.3 Number of rules different . . . 22
8.8 Help . . . 22
8.8.1 Detecting cycles . . . 22

9 Evaluation **23**
9.1 Testing . . . 23
9.1.1 JUnit tests . . . 23
9.1.2 Survey - People who do/don't understand logic . . . 23
9.2 Performance . . . 23

10 Conclusions **24**
10.1 Completed objectives . . . 24
10.2 Comparisons to related work . . . 24

11 Future Work **25**
11.1 Improvements to completed objectives . . . 25
11.2 Device support . . . 25
11.2.1 Reduced app without trees for phones . . . 25
11.3 More complex help . . . 25
11.3.1 Calculating the optimal route . . . 25
11.3.2 Suggested next move . . . 25
11.3.3 Future dead end detection . . . 25
11.4 First Order Logic . . . 25
11.4.1 Extend grammar . . . 25
11.4.2 Add and handle new rules . . . 25

12 User Guide **27**
12.1 How to use the app . . . 27

Introduction

We have been challenged with developing an android application to help new students learn propositional logic and to complete logical equivalences. In this section we will discuss our motivation, approach and objectives.

1.1 Motivation

All logics are based on propositional logic in some form, so it is important that new students learn how to use it. Propositional logic consists of syntax, semantics and proof theory; syntax is the formal language which is used to express concepts, semantics provide meaning for the language and proof theory provides a way to convert one formula into another using a defined set of rules.

We know that new students learning propositional logic can struggle to understand the rules and how they should be applied to formulae. To help with this our idea is to create an equivalence teaching tool; this will be a tablet application which will allow a user to apply rules to a formula until they have reached the desired equivalence.

1.2 Approach

We decided early on in the project that the tool should be an Android tablet application as opposed to a web or mobile one; this allows for an intuitive, interactive design while still having a large screen space.

The tool can be divided up into its main component parts: the parser, tree constructor, tree processing and tablet interface. The parser will be generated by the ANTLR4 parser generator, the tree constructor, processing code and the tablet interface will be written in Java with the interface using the Android SDK.

The parser requires a grammar to generate the relevant parser components. This will be used to parse the initial equivalence and return the ANTLR4 tree

representation of the string.

The tree constructor is required to take the ANTLR4 representation of a tree and convert it into a more useful data structure which can be modified and displayed easily. The operators and atoms of the formula will be represented as nodes and leaves respectively.

Tree processing will be used to internally calculate which rules are applicable to each node and will allow us to subsequently manipulate the tree by applying these rules.

The interface will have an intuitive design displaying the current formula's formation tree (generated by the tree constructor) and allowing a user to click on the operators to apply a rule.

1.3 Objectives

The application should have some key features. These are outlined below:

1. Graphical tree representation of formulae

   Representing the formula as a tree structure allows a user to see exactly how the formula should be read and can help them understand the order of operations. It also allows the user to click on an operator to select a rule for it; this is much more intuitive than just clicking on the whole formula and not knowing which section the rule would be applied to.

2. Undo/Redo functionality

   Previous equivalent formulae will be displayed above the current formula. When an old formula is selected it will expand into tree form and the formulae below it are faded out *(Undo)*. This will allow a user to perform rules on the old tree or select one of the later faded trees *(Redo)*. When a rule is applied, the faded formulae will be removed from the history allowing the user to continue on from that point (Figure 1.1).

3. Generated equivalences

   We want to allow the user to have equivalences automatically generated for them to solve. This would be implemented by running the tool on a generated formula to give a significantly different equivalent one. The key advantage of having this feature as well as allowing manual entry of equivalences would be that the user will have a continuous supply of new equivalent logic formulae after completing all those set by their lecturer. This also requires generating the initial formula for the tool to be run upon or allowing the user to manually enter the first formula and have the second one generated for them.

4. Difficulty setting

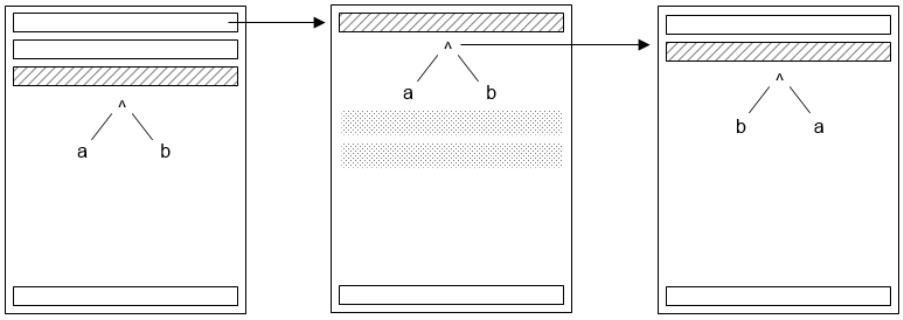

Figure 1.1: Undo functionality - Clicking a previous equivalence shows the formula tree for that equivalence and sets the undone history as translucent. Applying a rule then overwrites the undone history.

Extending the idea of generated equivalences, we could allow the user to select a difficulty. This would be calculated by the length of the generated formula and how many rules were applied to get its equivalent formula. We could also provide a recommended difficulty based on how many previous equivalences they have completed and how far from optimal their solutions were.

5. Help

   Providing the user with help is key to their improvement. Once an equivalence has been set up, the tool should calculate the optimal route from the starting formula to the desired end point. Upon finishing the equivalence the user will receive a message telling them how far from the optimal solution they were and give them the option to try again or to view the optimal solution. Throughout there will also be a help button available that will suggest the next step to the user on request; this includes the recommendation to undo certain steps if the user reaches a cycle. Pointing out mistakes could also be enabled, so if the user has completed a cycle or is heading towards a dead end they will be prompted to consider a different strategy.

Background

We are assuming a basic understanding of propositional logic, including the operators and rules that are defined in the system. For more information on these please visit the Wikipedia article[1].

2.1 Propositional Logic

Propositional logic is a branch of logic that studies ways of combining and modifying whole sentences, statements or propositions to form more complex propositions. It is a formal system containing logical relations and properties which are derived from these methods of joining or altering statements.

A logical system contains three major parts:

1. Syntax - the formal language that is used to express concepts.
2. Semantics - provide meaning for the language.
3. Proof theory - provides a way to convert one formula into another using a defined set of rules.

Logical definitions:

- *Atomic* - A formula whose logical form is $\top$, $\bot$ or p for an atom p.
- *Negated atomic* - A formula of the form $\neg p$.
- *Negated formula* - A formula of the form $\neg A$ for a formula A.
- *Conjunction* - A formula of the form $A \wedge B$.
- *Disjunction* - A formula of the form $A \vee B$.
- *Implication* - A formula of the form $A \rightarrow B$, where A is the *antecedent* and B is the *consequent*.
- *Literal* - A formula that is either atomic or negated atomic.
- *Clause* - A disjunction of one or more literals.

A *statement* is defined as a meaningful declarative sentence that is either true or false. For example, a statement could be:

- 'Socrates is a man.'
- 'All men live on Earth.'

A statement can be constructed of multiple parts, for example, the above statements can be combined into:

- 'Socrates is a man and all men live on Earth.'

Each part of this statement can be considered a proposition. Propositional logic involves studying the connectives that join these such as *'and'* and *'or'* (to form conjunctions and disjunctions), the rules that determine the truth values of the propositions and what that means for the validity of the statement.

2.2 Truth tables

It is necessary to understand the meanings of the symbols used in a language. Truth tables are mathematical tables used in logic to compute the functional values of logical expressions. They can be used to determine whether or not a propositional logic statement is logically valid.

A *situation* determines whether each propositional atom is true or false. A truth table shows all the situations the input variables can be in. We write 1 for true and 0 for false as shown below:

A	B	A∧B	A∨B
1	1	1	1
1	0	0	1
0	1	0	1
0	0	0	0

Truth tables can be used to define any operators, including any new ones which might be desired.

Android

3.1 Android vs Web app

3.1.1 Intuitive to use

3.1.2 Learning experience

3.1.3 None currently available for Android

3.2 Tablet vs Phone app

3.2.1 Screensize

3.2.2 Formation Tree would not feasible on phone app

3.3 Features

3.3.1 Multi-Touch Gestures

Android is an operating system based on the Linux kernel which was designed primarily for touchscreen mobile devices such as smartphones and tablet computers[3].

The user interface is based around direct manipulation. This means using touch inputs that loosely correspond to real-world actions, for example swiping, tapping and pinching to control items on screen. The response to user actions is designed to be immediate and provides a fluid touch interface. It often provides haptic feedback (forces, vibrations or motions) to the user using the vibration capabilities.

Android takes advantage of internal hardware such as accelerometers, gyroscopes and proximity sensors in some applications to respond to other user actions, such as re-orientating the screen from portrait to landscape.

3.4 Market Share

Android's main competitor in the mobile platform market is Apple's iOS. In research conducted in the fourth quarter of 2012, Kantar Worldpanel Comtech showed sales of all Android smartphones worldwide outpacing the iPhone by a huge margin: 70 percent to 21 percent of the smartphone market[4].

In the tablet market the iPad dominated Android's 7" tablets with 53.8 percent to 42.7, which is lower than Android's smartphone market but steadily increasing.

Related Work

4.1 Logic Daemon

Created in Texas A&M University, the Logic Daemon[5] is an online logic proof checker. It comprises a simple web page with two small text input boxes for the premises and conclusion and then one large text box for applying primitive rules (Figure: 4.1).

While this tool does allow us to apply rules to prove equivalences we do not find it very intuitive to use at all. The interface itself looks quite unclear and isn't user friendly so it would not be suited to new students. Applying the rules to the premises is also quite confusing as it shows all the rules and does not alert the user to the fact they cannot be applied until they have already clicked on it. We found this a frustrating way of attempting a proof; as such, we intend to only display the rules that can be applied in the current situation.

However, it is a useful tool for those more knowledgeable about logic for checking proofs once they have got to grips with the interface. It also has a *'Get Help'* button to suggest which rule to use next. We hope to use the idea of a help button in our project but to build upon it in order to provide more extensive help such as showing when an action has been repeated (i.e. a cycle has been reached).

We hope that our tool will be more intuitive to use because we will display formulae as formation trees with interactive nodes to apply rules with. This should provide a more straight forward way of presenting the rules to new students.

The website also provides a simple equivalence checker, well-formed formula checker and countermodel checker. These are all similar tools useful in logic but again the interface is not particularly intuitive or user friendly.

The Logic Daemon tool also provides support for first order logic which we are not planning on covering in our main tool. However, this could be implemented as an extension at the end of our project.

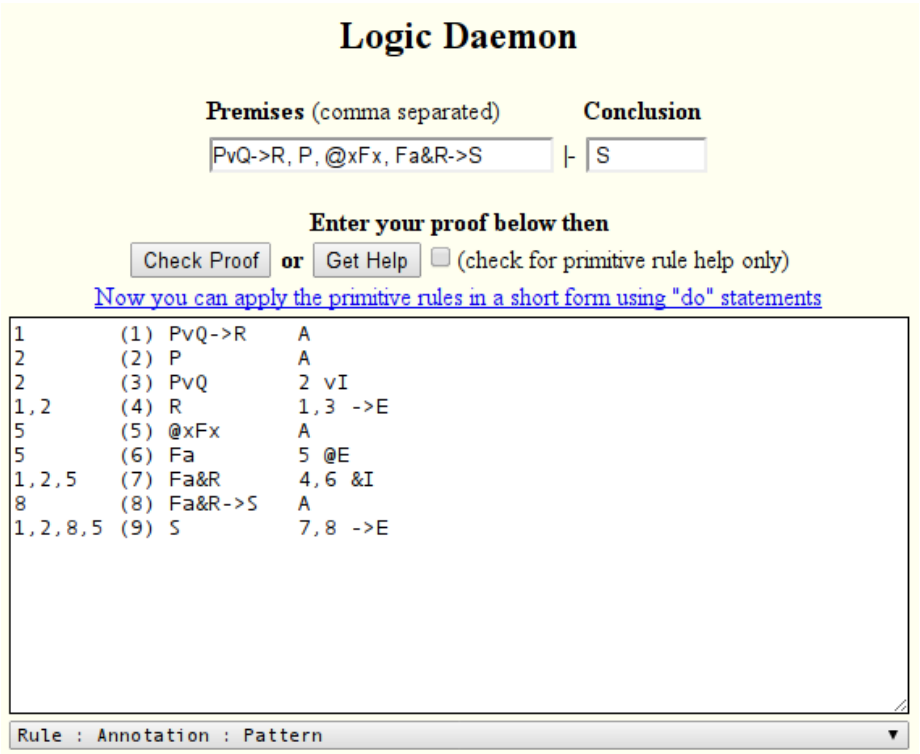

Figure 4.1: Logic Daemon

4.2 Pandora

Pandora[6] is a tool created by Imperial College London; it stands for 'Proof Assistant for Natural Deduction using Organised Rectangular Areas'. It can be used to prove that a goal formula follows from the given formulae. Pandora allows the user to repeatedly apply natural deduction rules (Figure: 4.2).

We do not intend to include natural deduction in our tool as we want to focus more on solving propositional logic equivalences. Pandora is a much nicer tool than the Logic Daemon and we find it much more intuitive and easy to use.

Pandora allows a user to apply rules forwards or backwards. We know from using Pandora that some proofs are easier to complete working backwards; as such, this is an idea we hope to incorporate into our tool by allowing the user to expand either the top or bottom formula to show its formation tree and to apply rules.

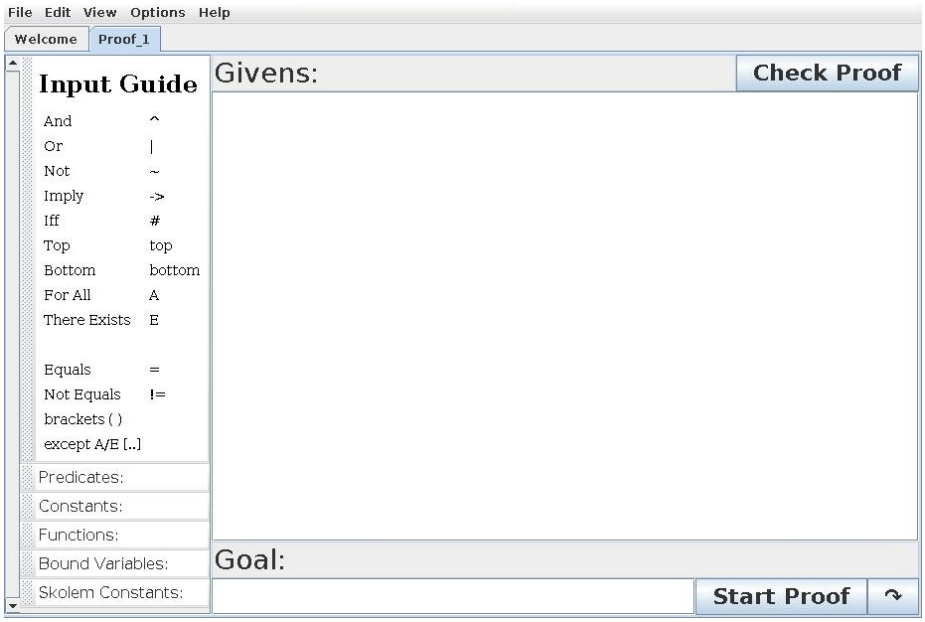

Figure 4.2: Pandora

4.3 Logic Solver

This is an Android application to show truth tables and possible logical equivalences. It looks like it has been made as a phone application as on a tablet it fills very little of the screen and it is difficult to click on some of the links (Figure 4.3).

The user can apply rules to the formula they enter into the application; this is done by selecting the rule they want out of a list which is then applied to the formula. However they cannot use this application to solve an equivalence as they can only enter one formula and then apply rules to that. We want to improve upon this in our own application by allowing the user to enter two formulae and work from either end to solve it.

The list of rules that is offered to the user only shows the rules that can be applied to the current formula. We want to also offer this behaviour as there are far too many rules to display them all to the user and expect them to sort through which could be applied.

Formulae are displayed simply in the application; we think this could be improved upon and want to display our current formula as its formation tree.

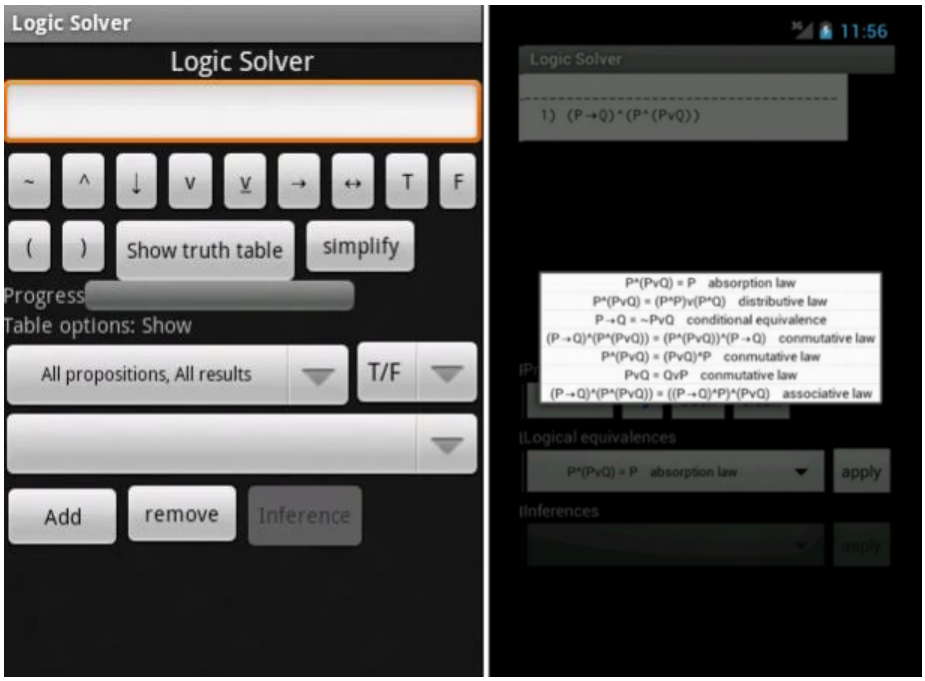

Figure 4.3: Logic Solver Android Application

4.4 Truth Tables

Truth Tables is also an Android application mainly for use on a phone. It simply generates and displays truth tables based on a logical formula (Figure 4.4).

We currently do not plan on adding truth tables to our application as we do not believe it is as useful for learning and understanding propositional logic as our other ideas. However, this could be added as an extension at the end of the project depending on how much time is left.

4.5 LogicCalc

LogicCalc is an Android application for solving problems with propositional logic. It allows the user to create workbooks to save and print their proofs (Figure 4.5). However we had trouble running it on our Nexus tablet.

We like the idea of saving proofs in workbooks for future reference. This is something we had not considered before and are curious to explore as an extension. This functionality would be useful for students completing exercises that needed a paper hand in as they would be able to save and print them off.

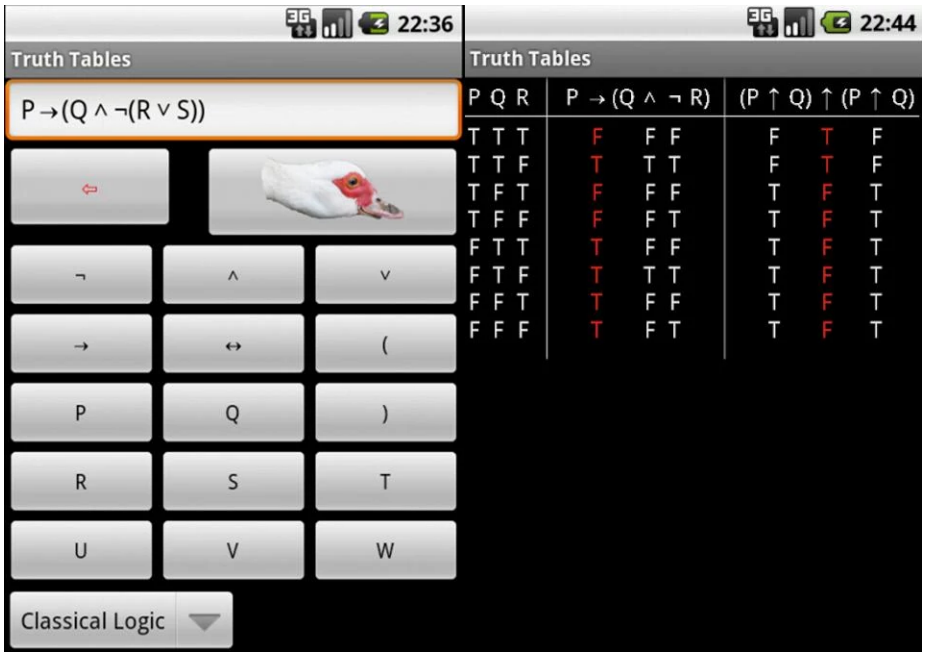

Figure 4.4: Truth Tables Android Application

4.6 Propositional Logic Calculator

The Propositional Logic Calculator[7] finds all of the models of a given propositional formula. The website tells us that the only limitation for this calculator is that we have only three atomic propositions to choose from: p,q and r (Figure: 4.6).

Propositions are entered using the keyboard they provide on the calculator and the reasoning process is initiated by clicking 'ENTER'. It then calculates the truth value assignments that will make the formula true in the 'MODELS' section and the truth value assignments making the formula false in the 'COUNTERMODELS' section.

This tool is used simply for calculating truth values of a formula. We will not be implementing this in our equivalences tool because we do not think that knowing the truth values for a formula is as useful for learning and understanding propositional logic as our other ideas.

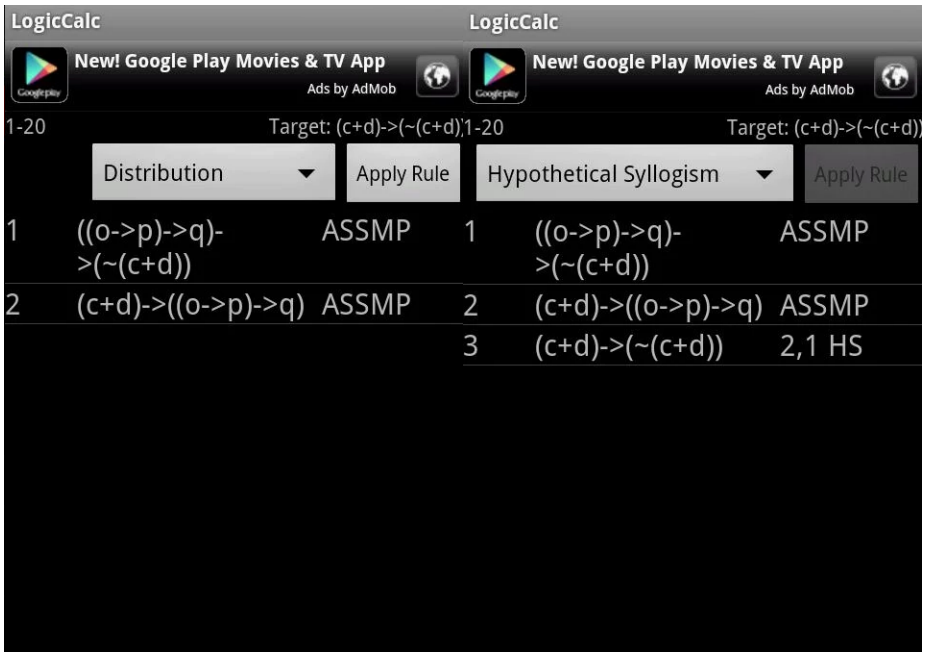

Figure 4.5: LogicCalc Android Application

4.7 Previous Year Projects

Once we have made more progress on our tool we will look into finding out what previous years did when they were assigned similar projects. We do not want to look into these this early in the project so we can generate our own implementation ideas as these projects will be very similar to ours.

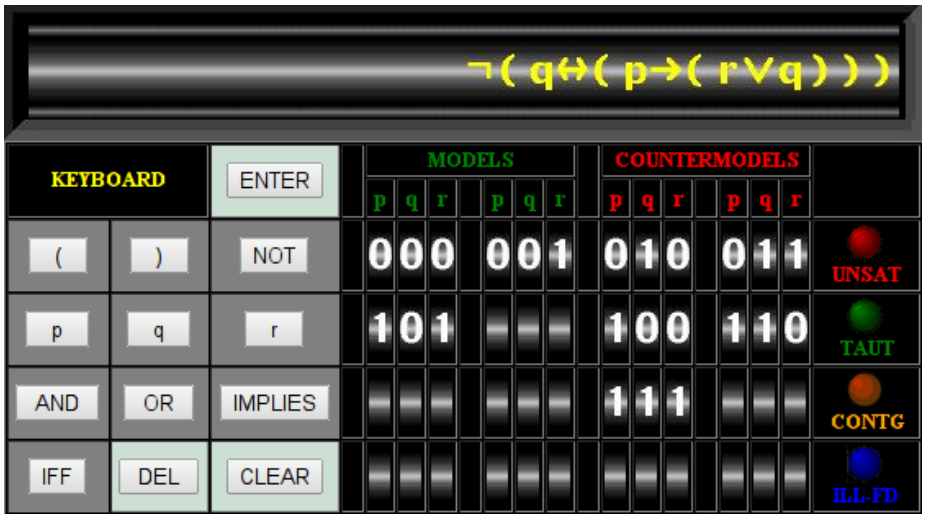

Figure 4.6: Propositional Logic Calculator

Parser

5.1 Parser Generation

5.1.1 ANTLR4 and alternatives

5.2 Grammar

5.2.1 Tokens

5.2.2 Order of Precedence

5.2.3 Error Handling - Incorrect grammar syntax to handle errors semantically

5.2.4 Extension to First Order Logic

5.3 Parser

5.3.1 Parser

5.3.2 Walker

Internal Tree Representation

6.1 Compiler

6.2 Formation Tree

6.2.1 Tree operations, eg. adding a new node

6.3 Nodes

6.3.1 Binary Operators

6.3.2 Unary Operators

6.3.3 Atoms

Tree Manipulation

7.1 The Rule Engine

7.2 Rule Selection - which rules can be applied to a node

7.3 Rule Application - how rules are applied to a node

7.4 Random Rule Application

7.4.1 Generating equivalent formulae

The App

8.1 Truth Tables

8.1.1 Ensure equivalences are equal before beginning

8.2 Random Generation of Equivalences

8.3 Custom Keyboard

8.3.1 Logic symbols and variables

8.4 Undo/Redo Functionality

8.4.1 Stack of trees maintained

8.4.2 OnClick listeners

8.5 Formation Tree Representation

8.5.1 Tree Layout

8.5.2 Walker's Algorithm

8.5.3 Alternatives

8.6 Rule Application

8.6.1 How rules are displayed within the app

8.7 Difficulty

8.7.1 Length of generated equivalences

8.7.2 Number of variables/operators

8.7.3 Number of rules different

Evaluation

9.1 Testing

9.1.1 JUnit tests

9.1.2 Survey - People who do/don't understand logic

9.2 Performance

Once the application has been built we will need to find a way to test that it has met the objectives we set out to achieve. The main objective is to create an intuitive Android application that helps students improve in solving equivalences; as such, we will survey a wide range of test subjects. These tests will be split into two groups: people who understand logic and equivalences well and people who are new to propositional logic.

We need to ask people who understand logic (e.g. lecturers) so we can confirm that it works as a teaching tool. People who are new to logic (e.g. first year students) are necessary so we can evaluate how effective it is as a learning tool. This can be carried out through asking the target groups to use the application and provide feedback.

Conclusions

10.1 Completed objectives

10.2 Comparisons to related work

Future Work

11.1 Improvements to completed objectives

11.2 Device support

11.2.1 Reduced app without trees for phones

11.3 More complex help

11.3.1 Calculating the optimal route

11.3.2 Suggested next move

11.3.3 Future dead end detection

11.4 First Order Logic

11.4.1 Extend grammar

11.4.2 Add and handle new rules

Bibliography

[1] Wikipedia, *Propositional Logic*. http://en.wikipedia.org/wiki/Propositional_calculus

[2] IEP, *Propositional Logic*. http://www.iep.utm.edu/prop-log

[3] Wikipedia, *Android (Operating System)*. http://en.wikipedia.org/wiki/Android_(operating_system)

[4] Harry McCracken, *Whos Winning, iOS or Android?*. http://techland.time.com/2013/04/16/ios-vs-android

[5] Colin Allen, Chris Menzel, *Logic Daemon*. http://logic.tamu.edu

[6] Imperial College London, *Pandora*. http://www.doc.ic.ac.uk/pandora/newpandora/index.html

[7] Enrico Franconi, *Propositional Logic Calculator*. http://www.inf.unibz.it/~franconi/teaching/propcalc

User Guide

12.1 How to use the app