A.Ndlovu

215001111

CSC3A10: Mini Project Research Assignment

Automated Theorem Prover

1. Introduction

The phase, “There’s an App for that!” is one that has become inexhaustibly true. Especially in this day and age where new applications of software technology are readily available for free on various online software stores. However, there is one that may not have been investigated as much as I recon it should have.

The modern science student has found great utility in the use of a calculator, be it for basic arithmetic, calculus, algebra or statistics. It has helped students get through the number crunching with ease and focus on the greater details of their argument (mostly, the structure of their arguments). In other words, the logic of their arguments. Logic is the foundation upon which mathematical reasoning is built. And as a Computer Science student, it follows that there be some pursuit towards Abstraction with regards to the self-evident relationship between Logic and the very interesting instance of its application which is Mathematics.

Great utility has been found in Automated Calculators, perhaps we can find Automated Theorem Provers to be just as useful. For the case of this investigation, an Automated Theorem Prover for Propositional Logic. Fortunately, Logic not only studies and guides our reasoning, it also provides a formal and unambiguous logical language structure by which we are able to express logical arguments. This logical language structure is provided by means of logical formulae. By inspecting the syntax and semantics of our logical formulae, we can deduce its logical validity (Conradie and Goranko, 2015). So, reasonably, an application that is able to do this could serve as a decent Theorem Prover, externalizing the burden of scrutinizing smaller components of proposed theorems, and by consequence, increasing the productivity of the student.

Below, the problem background will be explored. The objective of the investigation will be formally presented, along with what other investigations have already uncovered and how that was done. The model we have opted for as a means for in carrying out the investigation will also be presented in great detail. From the structures we have put in place to investigate our problem background and how they work, to how they relate with one another. The outcome of our investigation and what conclusion we come to from the results will also be discussed.

2. Problem background

In this investigation, we are concerned with whether it is feasible to translate the mechanisms of Logical Deduction in Propositional Logic into a simple software application (An Automated Theorem Prover) that can be easily used by people who seek to lessen the burden of proving the smaller propositions or theorems employed in the larger scale theorems being crafted. A time-saver for those employing Propositional Logic to express their theorems and want them validated so that incorrect reasoning is exposed as soon as it is proposed.

The notion of Logical Consequence is one derived from the idea of a propositional formula being logically valid (Conradie and Goranko, 2015). Logical validity of a Propositional Formula means whether or not that formula attains a truth value true, no matter what truth value is assigned to the atomic propositions it uses. Basically, a tautology is a Logically Valid argument. It is true regardless of what meaning you assign to its atomic propositions. It is semantically correct! Logical Consequence is about whether or not the propositional formulae you suppose lead to the propositional formula you propose under any truth value assignment of the atomic proposition used. This is, do your premises for your theorem lead to the conclusion in any given context? Are your propositional inferences Logically Valid? This is the question that Deductive Systems in Propositional Logic answer.

Currently, a few of deductive systems have been developed. Namely, Semantic Tableaux, Propositional Resolution and Truth Tables (Ben-Ari 2012). A number of software applications exist for Automated Theorem Proving, but they are not easily accessible or user friendly to Propositional Logic beginners.

- 3TAP developed at the University of Karlsruhe,
- Meteor developed using Unix based the C programming language for Predicate Logic,
- Parthenon, a parallel theorem prover for Predicate Logic heavily dependent on C’s multithreading capabilities,
- PTTP which is implemented using Prolog and Lisp for Predicate Logic,
- SETHEO based on the C programming language,
- LoTREC based on Java used for Modal Logic,
- MLTP based on C++,
- FaCT++ based on C++,
- And Pellet based on Java.
  (Islam *et al.*, 2012)

These are all applications based on the Semantic Tableaux Deductive System, which is the same deductive system used in this investigation for reasons that will be discussed under the model segment of the report.

3. Model

Of the 3 above-mentioned deductive systems, Semantic Tableaux seems to be the best-fitted system/model for the deductive process as it has the least constraints and easily translates to well-known data structures implementable via Java, making the development of an Automated Theorem Prover feasible for a Java programmer like myself.

The Semantic Tableau method is based on a Binary Tree Structure assisted with Linked Lists and their Iterators, Stack and Queues. So at least 4 data structures are employed with the objective of reducing time complexity as much as possible. This method is one that is highly dependent on the proper execution of the expansion rules that drive this algorithm.

The table below explains how the expansion on an inference works (Gaintzarain *et al.*, 2008):

Branching Rules:	Non-Branching Rules:
A and B : F Expands to A:F or B:F	A and B : T Expands to A:T, B:T
A or B : T Expands to A:T or B:T	A or B : F Expands to A:F, B:F
A implies B : T Expands to A:F or B:T	A implies B : F Expands to A:T, B:F
A bi-implies B : T Expands to A:T, B:T or A:F, B:F	Not A : T Expands to A:F
A bi-implies B : F Expands to A:T, B:F or A:F, B:T	Not A : F Expands to A:T

Most of how the tableau algorithm for this application works mimics what is done when the logical validity of a formula is deduced by pen and paper. For example: Q, (P ∨ (¬ Q)) |= P

```
((Q∧ (P ∨ (¬ Q))) → P): F
            |
(Q∧ (P ∨ (¬ Q))): T
            |
            P: F
            |
            Q: T
            |
(P ∨ (¬ Q)): T

/                     \

P: T                  (¬ Q): T

 X                        |

                       Q: F

                         X
```

This is how the tableau algorithm will be implemented:
A linked binary tree will be instantiated with a node that points to a parent node, left child node, right child node, a linked list of formulae to be expanded and their truth values, a Boolean for whether each formula has been expanded and a Boolean value indicating whether or not that node is closed.

The root of the tree will hold one formula in its formula list. A formula constructed by the conjunction of all the premises, if any, implying the consequent. Queues will be used to aid in heuristics, as it would be better to exhaust all non-branching formulas before branching formulas when dealing with formula expansion. One queue will hold the nodes to be expanded (just the root node in the beginning), another will hold branching formulas. When a non-branching formula is expanded, the resulting formula(s) will be added to the end of that node's formula list and the expanded formula will be marked as expanded, and a search for that formula will traverse up its ancestor nodes' formula lists to check for the existence of that formula with a contradictory truth value. If such a formula is found that node is closed and nothing else can be done to that node. If none is found, the expansions continue and the expansions that lead to branching formulas have those formulas stored in the queue for branching formula until all non-branching formulas are expanded.

The expansion of branching formulae is different. The resulting formulas from the expansion have their own nodes and formula lists created for them and they are stored in those new nodes' formula lists, a similar search for closure is applied and these nodes are added to the node queue if that node is not closed (Islam *et al.*, 2012).

After all formulas have been expanded, a search to see if all leaf nodes are closed is implemented. If all leaf nodes are closed, the Propositional Inference is Logically Valid (the consequent formula is a logical consequence of the premises).

Due to the nature the user's input, parentheses are a very crucial aspect to making sure the application can process the user input accordingly, so a helper method that makes sure that opening parentheses have matching closing parentheses is used. It is based on the Stack ADT (GoodRich *et al.* 2014).

Searching for closure and checking the closure of all leaf nodes uses Iterators over Linked Lists that hold Linked Binary Tree Nodes. Checking the closure of all leaf nodes uses in order traversal of the Binary tree.

4. Results

The implementation of the semantic tableaux algorithm proved to be rather complicated when it came to debugging as there are iterations over lists and trees. It seemed to work better for inferences that had little to no branching formulas. The example given above in the model segment of this report was logically valid and the same result was achieved via the application. Inferences like, $\neg (p \rightarrow \neg q) \rightarrow (p \lor \neg r)$ and $((p \land \neg q) \rightarrow \neg r) \leftrightarrow ((p \land r) \rightarrow q)$ yielded false results. It may be due to the fact that there are missing cases in the expansion rules and that leads to a falsely structured Tableaux Tree. The problem compounds with more branching formulas being nested in the inferences as the structure of the Tableaux Tree is driven by branching formulae that adds nodes to the tree. The algorithm returns a Boolean value upon receiving a string input expression determining the validity of the input expression (inference formula).

Parentheses greatly affected the structure of how branching and non-branching formulae are determined. When testing, it became clear that irregular/ varying use of parentheses led to false results, so a rule had to be imposed on the user in order for there to be uniformity in the expected cases of the input. All unary and binary connectives applied onto atomic formulae had to have parentheses around them. And connectives applied onto compound formulae had to have each argument surrounded by parentheses.

5. Conclusion

Automated Theorem Provers are a feasible application in assisting logicians and boosting their productivity by externalizing the analysis of smaller inferences. They are not too difficult to develop as they are implementable via well-known data structures. It may also be worth mentioning that results are generated almost immediately as the inferences are inputted. This is due to the fact that there are a finite number of expansions for any Propositional Inference and searches for closure do not traverse every node in the tree. Just the ancestor nodes of that particular node. A search through the whole tree happens once and is not very significant as there are usually for formula expansions than there are nodes in the Tableaux Binary Tree.

This structure could be adapted for Predicate Logic in future, and maybe a number of other kinds of logic applicable to the study of discrete mathematics and Logic itself. The branching rules could still be adapted for operator precedence as a means of helping to make the parentheses issue less constraining on the side of the user. They could also be better structured, in general so as to reduce the number of inferences that this application fails to validate accurately.

Bibliography

Gaintzarain, J., Hermo, M., Lucio P. and Navarro1, M. (2008). Systematic Semantic Tableaux for PLTL. *Electronic Notes in Theoretical Computer Science.* 206: 59-73. Doi: 10.1016/j.entcs.2008.03.075

Conradie, W. and Goranko, V. (2015). *Logic and Discrete Mathematics: A Concise Introduction* (1st ed.). West Sussex: John Wiley and Sons, Ltd.

Islam, Z., Mashiyat A. S., Khan K. N., and Karim S. M. M. (2012). A Tableau Based Automated Theorem Prover Using High Performance Computing. 7(3): 597-607.

GoodRich M. T., Tamassia R. T. and Goldwasser M. H. (2014). *Data Structures and Algorithms in Java* (6th ed.). New Jersey: John Wiley and Sons, Ltd.

Ben-Ari M. (2012). *Mathematical Logic for Computer Sciences.* London: Springer Verlag London.