Natural Style for Resolution Proofs in Theorema

Diana Dubu[1] Tudor Jebelean[2]

[1]Department of Computer Science, West University of Timişoara
[2]Research Institute for Symbolic Computation, Hagenberg,Austria,
email: [1]ddubu@.info.uvt.ro, [2]jebelean@risc.uni-linz.ac.at

Abstract

The proofs generated by clausa reasoners are often too long and hard to follow by the user (even if experienced), because most of the structure of the initial problem is not expressed by the clausal formula. For this reason, it is very important, especially for practical applications, to express the proofs in a more natural style. This paper analyzes the method suggested by [Meier], namely to perform the transformation on the proofs obtained by resolution at the assertion level. More precise, the emphasis was on integrating the algorithms suggested by [Meier] in the *Theorema* system which is an automated theorem prover built on *Mathematica*. A first step was to bring the proofs generated by a resolution prover - in this case Otter - in a uniform representation - i.e. refutation graphs.

Keywords: Resolution proofs, natural deduction, Otter, Theorema

1 Introduction

Automated theorem provers (ATPs) have reached reasonable strength and their application is not only restricted to Mathematics but extends also to Computer Science - allowing software generation or verification, hardware verification - , Engineering and even Social Science. Since they are powerful systems they need to interact effectively with the user so the output they produce must be human-readable. Such is the case of the proofs generated, field in which there is a continuous concern for the natural style of the produced proofs. Most of the automated techniques though have the drawback that the proofs they generate are difficult to read even for experienced users. This resides in the fact that proofs are produced in a machine-oriented formalism.

As a consequence, many efforts have been made to bring these proofs to intelligible formats. Rewrite-rules have been proposed by [Buchberger, Jebelean], S-decomposition in [Jebelean]. While these methods search for an optimal proof applying the transformation rules on-the-fly, the method proposed by [Meier] reconstructs the proof obtained already by resolution. Other approaches such as ProVerb [Huang, Fiedler] or ILF [Dahn] suffer mainly of two problems. First, the transformation procedures generate very often natural deduction proofs with many indirect parts since they operate at the literal level step. Secondly, the natural deduction calculus is not eligible for presenting mathematical proofs as the inferences are at the level of syntactical manipulations of logical connectives and quantifiers and not at the level of theorem or definition applications as in mathematical textbooks.

To address these problems, an algorithm for transformation of refutation graphs - which are abstract representations of resolution proofs - into natural deduction proofs at the assertion level is proposed in [Meier]. The algorithm is based on transformations at the level of assertions - i.e., theorems, lemmata or definitions - such that there is no unnecessary decomposition at the literal level followed by an abstraction of the resulting proofs.

This paper will present the suggestion for adapting this idea in the Theorema system which is an integrated environment for proving, solving and computing built on top of mathematical software system Mathematica. Since in order to verify in practice the eligibility of the algorithm one must first reach the uniform representation of the resolution proofs, this paper deals with the generation refutation graph for Otter proofs.

The paper is organized as follows. Section 2 presents briefly the interaction between Theorema and Otter. Section 3 introduces the main concepts - resolution, refutation graphs. Section 4 summarizes the implementation of the transformation procedure in Theorema and section 5 introduces the principles for further work.

2 Theorema and Otter

Theorema is a software system for interactive mathematics. The system is implemented on top of Mathematica, thus it is backed by the full algorithmic and computing power of the currently most popular computer algebra system. The system interacts with the user in the language of predicate logic, which is the natural language for expressing mathematical properties and algorithms. Proving is done with specific methods for several mathematical domains: propositional logic, general predicate logic, induction over integers and over lists, set theory, boolean combinations of polynomial [in]equalities (using Groebner Bases), combinatorial summation (using Paule–Schorn–Zeilberger), and a novel technique for proving in higher-order logic with equality: PCS (proving–computing–solving), introduced by Buchberger [Jebelean, Buchberger]. The aim of the *Theorema* project is to provide an easy-to-use intelligent environment for the working scientist, which integrates the computing capabilities of a computer algebra system with the deduction capabilities of automatic theorem provers [Theorema Group].

Otter (Organized Techniques for Theorem-proving and Effective Research) is a resolution-style theorem-proving program for first-order logic with equality. Otter includes the inference rules binary resolution, hyperresolution, UR-resolution and binary paramodulation. Otter is coded in C, is free and is portable to many different kinds of computer.

The interaction between Theorema and Otter is described in [Kutsia, Nakagawa]. The interface implements two types of direct links from Theorema to external systems: black box style and white box style links. A black box style direct link consists of two parts: the translator component and the linking component. A white box style direct link consists of the translator, the linking component and the back translator. Both links work as follows: first, the translator gets the Theorema goal, knowledge base and options and translates them into the prover format thus preparing the input for the prover call. The linking component gets the translated string and options, writes the string in a temporary file and calls the prover with the options. Finally, the prover output is passed back to Theorema. The user has a full control over the external system using prover options. The sequence of operations performed by the white box style direct link is the same as those of the black box style link until the external prover output is passed back. Here the back translation component gets the prover output, transforms it into Theorema syntax and places into the Theorema proof object. Finally, the Theorema style proof is shown in the proof notebook. Note that if the prover failed to prove the goal, the back translation component does not affect the output and the link behaves like a black box style link.

3 Refutation Graphs

The resolution principle, introduced in 1965 by Robinson, is described in [Chang, Lee] as an inference rule that generates resolvents from a set of clauses.

A *clause* is a disjunction of literals. Let C_1 and C_2 be two clauses with no variables in common. Let L_1 and L_2 be two literals in C_1 and C_2, respectively. If L_1 and $\sim L_2$ have a most general unifier σ, then the clause $(C_1\sigma - L_1\sigma) \cup (C_2\sigma - L_2\sigma)$ is called a *binary resolvent* of C_1 and C_2.

Given a set S of clauses, a (resolution) *deduction* of C from S is a finite sequence $C_1, C_2, ..., C_k$ of clauses such that each C_i either is a clause in S or a resolvent of clauses preceding C_i and $C_k = C$. A deduction of from S is called a *refutation* or *proof* of S [Chang, Lee].

A *clause graph* is a quadruple $G = (\mathcal{L}, \mathcal{C}, M_{Lit}, \Pi)$, where

- $\mathcal{L}$ is a finite set. Its members are called the *literal nodes of G*.
- $\mathcal{C} \subset 2^{\mathcal{L}}$ is a partition of the set of literal nodes. Elements of C are called the *clause nodes* of G and $\emptyset \in \mathcal{C}$.
- M_Lit is a mapping from $\mathcal{L}$ to a set of literals, labelling the literal nodes with literals.
- The set of links Π is a partition of a subset of $\mathcal{L}$, such that for all $\Lambda \in \Pi$ the following link conditions hold:
  π_1: all the literal nodes in one link are labelled with literals whose atoms are unifiable
  π_2: there must be at least one positive and one negative literal in a link.

Each link Λ has two opposite *shores*, a *positive shore* $S^+(\Lambda)$ and a *negative shore* $S^-(\Lambda)$, consisting of the literal nodes with positive and negative literals, respectively.

A clause graph that does not contain trails - i.e. a walk in which all links are distinct - joining a node to itself (*cycles*) is called acyclic. A *refutation graph* is a deduction graph without pure literal nodes, where a *deduction graph* is a non-empty, ground (all its literals are ground) and acyclic clause graph [Meier].

4 Implementation

In order to obtain the refutation graph for a proof obtained by resolution with Otter, one must analyze the output imported in Theorema as a string, identify and extract the needed information and generate the structure. The algorithm is in brief as follows:

Given $C_1, C_2, ..., C_n$ the initial set of clauses - after normalization which is performed by Otter - identify the resolvents $R_1, R_2, ..., R_m$ generated by the inference rules.

- $C_1, C_2, ..., C_n$ - and them only - will represent the nodes in the refutation graph.
- the links in the refutation graph are obtained by analyzing the resolvents and the clauses used as described bellow

1. Identify in each resolvent R_i from the set $R_1, R_2, ..., R_m$ which original clause has been used. For the inferences using resolvents to generate new ones, identify from which clauses have the former been generated.
2. Extract from the initial clauses the literals remaining after the resolution step and connect them such that each link has a positive shore and a negative one. All possible connections must be established between the literals from the original clauses which have not been deleted during the inference process.

Note: In order to apply step one, we have used labels for the literals in $C_1, C_2, ..., C_n$ and assigned those labels to the literals in the resolvents obtained throughout the proof. It is sometimes the case that one literal in R_i originates from more than one initial clause and therefore all this must be stored in order to obtain all possible links between the nodes of the refutation graph, as it has already been mentioned in the second step of the algorithm.

Operations are mainly performed on strings and consist of extracting the parts representing the useful information. Intermediate streams are used to ease the transformation process from strings into Mathematica syntax. Mathematica's indexing on lists eased the process of labelling the literals in the initial clauses.

Let us consider a simple example generating a reasonable complicated refutation graph.

Given the initial set of assumptions: $Q \Rightarrow R, R \Rightarrow (P \wedge Q), P \Rightarrow (Q \vee R)$, prove that $P \Leftrightarrow Q$. Otter does the normalization itself as seen in Figure 1 a) and then generates the proof and the resulting clauses are as in Figure 1 b)

```
formula_list(usable).
-(p<->q).
q->r.
r->p&q.
p->q|r.
end_of_list.

list(usable).
0 [] p|q.
0 [] -p| -q.
0 [] -q|r.
0 [] -r|p.
0 [] -r|q.
0 [] -p|q|r.
end_of_list.
```

1 a) Normalized assertions

```
---------------- PROOF ----------------
1 [] -p| -q.
2 [] -q|r.
3 [] -r|p.
4 [] -r|q.
5 [] -p|q|r.
6 [] p|q.
7 [hyper,6,2] p|r.
8 [hyper,7,3] p.
9 [hyper,8,5] q|r.
10 [hyper,9,4] q.
12 [hyper,10,1,8] $F.
------------ end of proof -------------
```

1 b) Otter proof (by resolution)

Figure 1: Otter output

For the considered example the refutation graph generated has multiple links with more than one shore and is presented in Figure 2. The corresponding structure is shown in Figure 3.

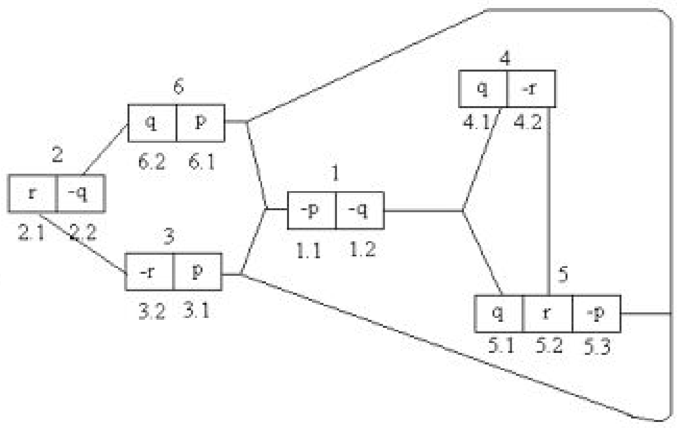
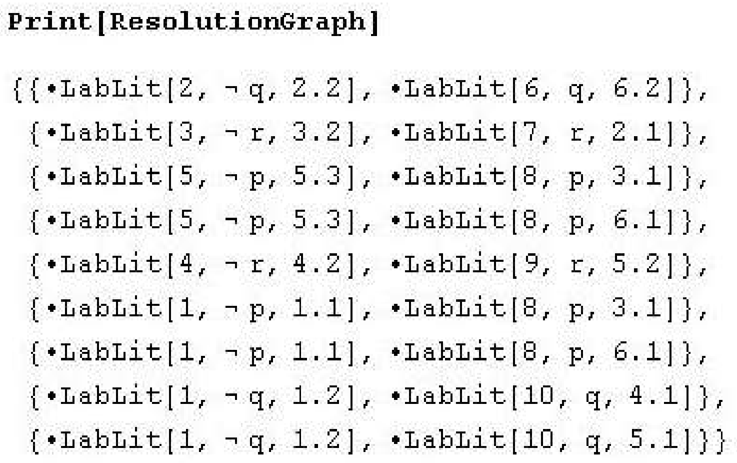


Figure 2: The refutation graph

Various proofs from Theorema's manual were analyzed and the results were encouraging. Extensions to the predicate logic and in the sense of quantifiers need to be taken into consideration, as underlined in the flowing section.



```
Print[ResolutionGraph]

{{•LabLit[2, ¬ q, 2.2], •LabLit[6, q, 6.2]},
 {•LabLit[3, ¬ r, 3.2], •LabLit[7, r, 2.1]},
 {•LabLit[5, ¬ p, 5.3], •LabLit[8, p, 3.1]},
 {•LabLit[5, ¬ p, 5.3], •LabLit[8, p, 6.1]},
 {•LabLit[4, ¬ r, 4.2], •LabLit[9, r, 5.2]},
 {•LabLit[1, ¬ p, 1.1], •LabLit[8, p, 3.1]},
 {•LabLit[1, ¬ p, 1.1], •LabLit[8, p, 6.1]},
 {•LabLit[1, ¬ q, 1.2], •LabLit[10, q, 4.1]},
 {•LabLit[1, ¬ q, 1.2], •LabLit[10, q, 5.1]}}
```

Figure 3: The structure chosen to implement the refutation graph

5 Future Work

A further research will therefore be focused on implementation of the proposed algorithm and perhaps refinements will be added. The transformation is operated on links in the refutation graph in order to simplify the presentation of the proof by eliminating unnecessary details from it.

Since the analysis was performed up to now only in propositional logic, a further level of detail will be extending it to the more general case of the predicate logic, dealing with quantifiers and therefore involving as well substitutions.

References

[Meier] Andreas Meier, *Transformation of Machine-Found Proofs into Assertion Level Proofs*, PhD Thesis, Draft, April 25, 2001

[Buchberger, Jebelean] Bruno Buchberger, Tudor Jebelean, *A Propositional Prover in Natural Deduction Style*, November 6, 2001

[Jebelean] Tudor Jebelean, *Natural Proofs in Elementary Analysis by S-Decomposition*

[Huang, Fiedler] Xiarong Huang and Armin Fiedler, *Presenting Machine-found Proofs*, Proceedings of the 13th Conference on Automated Deduction, pages 221-225, New Brunswick, NJ, SUA, 1996, Springer Verlag

[Dahn] B.I. Dahn and al. *Integration of Automated and Interactive Theorem Proving in ILF*, Proceedings of CADE-14, pages 57-60, 1997

[Kutsia, Nakagawa] Temur Kutsia and Koji Nakagawa, *System Description: Interface between Theorema and External Automation Deduction Systems*, Calculemus, June 2001

[Chang, Lee] Chin-Liang Chang, Richard Char-Tung Lee, *Symbolic Logic and Mechanical Theorem Proving*, Academic Press New York and London

[Theorema Group] The Theorema Group (B. Buchberger, K. Aigner, C. Dupre, T. Jebelean, F. Kriftner, M. Marin, K. Nakagawa, O. Podisor, E. Tomuta, Y. Usenko, D. Vasaru, W. Windsteiger): *Theorema: An Integrated System for Computation and Deduction in Natural Style*. In Proceedings of the Workshop on Integration of Deductive Systems at CADE-15, Lindau, Germany, July 1998.

[Jebelean, Buchberger] Tudor Jebelean, Bruno Buchberger, *Theorema: A System for the Working Mathematician*

[McCune] William W. McCune, *OTTER 3.0 Reference Manual and Guide*, ARGONNE NATIONAL LABORATORY, Distribution Category: Mathematics and Computer Science (UC-405)