1 Introduction

Unfortunately first-order logic has certain limitations which are felt in many applications such as in hardware verification. The use of proof assistants like the HOL system ([Gord88]) is therefore resorted to. However many theorems of higher order logic can also be proven by methods of first order logic as well. Being aware of this situation, we have implemented a prover based on Sequent Calculus within the HOL system, which can be used to mechanize proofs of necessary, but tedious lemmata required for a large proof in HOL. In order to find an efficient implementation, we have introduced the concept of unification in our prover. These modifications have resulted in a calculus called the “restricted sequent calculus” (RSEQ) and an automatic prover based on it called FAUST\(^1\)\(^2\).

The well known sequent calculus\(^3\) (SEQ) introduced by Gentzen [Gent35] has a major disadvantage as far as the so called γ-rules are concerned. The application of a γ-rule extends a sequent by an instance of a quantified formula of the sequent on which the rule is applied to. Unfortunately the γ-rule cannot be easily automated as the ‘right’ choice of the term for instantiation cannot be easily computed at the stage of rule application. The rule itself allows the use of any term, but usually only special terms lead to the desired proof. To overcome this deficiency we have introduced the concept of metavariables described in the next section.

2 The Concepts in RSEQ

Metavariable: γ-rule applications in RSEQ do not introduce arbitrary terms, but special place-holders called metavariables\(^4\), which do not belong to the language of our logic. For syntactical convenience Metavariables can be treated like usual variables, but it should be noted that input formulae must not contain any metavariables, since they are available only for the proof process.

\(^1\) First-Order Automation using Unification in a modified Sequent calculus Technique
\(^2\) A public domain version of FAUST can be obtained from the authors or along with the HOL system sources.
\(^3\) For details of syntax and semantics of SEQ the reader is referred to standard textbooks on logic, e.g. [Gall86].
\(^4\) A similar concept was developed independently by Reeves [Reev87].
In figure 2, an example illustrating the application of \( \gamma, \alpha, \) and \( \delta \)-rules are shown. It can be observed that before the \( \gamma \)-rule on \( \Phi \) is applied, one could substitute \( m_1 \leftarrow y_1 \) \( [m_2 \leftarrow f(a, y_1)] \) in order to obtain non-disjoint antecedent and succedent. However, this substitution is forbidden, since \( y_1 \in f_{m_1} \). Therefore the tree cannot be closed at this stage and further rule applications are necessary.

\[
\begin{align*}
\vdash \exists x. \forall y. \exists z. P(x, z) &\rightarrow P(y, f(a, y)) \parallel \{\} \\
&\phi \\
\vdash \forall y. \exists z. P(m_1, z) &\rightarrow P(y, f(a, y)), \Phi \parallel \{(m_1, \{\})\} \\
&\downarrow \gamma \\
\vdash \exists z. P(m_1, z) &\rightarrow P(y_1, f(a, y_1)), \Phi \parallel \{(m_1, \{y_1\})\} \\
&\downarrow \delta \\
\vdash P(m_1, m_2) &\rightarrow P(y_1, f(a, y_1)), \Psi, \Phi \parallel \{(m_1, \{y_1\}), (m_2, \{\})\} \\
&\downarrow \gamma \text{ applied on } \Phi \\
P(m_1, m_2) &\vdash P(y_1, f(a, y_1)), \forall y. \exists z. P(m_3, z) \rightarrow P(y, f(a, y)), \Phi \parallel \{(m_1, \{y_1\}), (m_2, \{\}), (m_3, \{\})\} \\
&\downarrow \delta \\
P(m_1, m_2) &\vdash P(y_1, f(a, y_1)), \exists z. P(m_3, z) \rightarrow P(y_2, f(a, y_2)), \Phi \parallel \{(m_1, \{y_1\}), (m_2, \{y_2\}), (m_3, \{\})\} \\
&\downarrow \gamma \\
P(m_1, m_2) &\vdash P(y_1, f(a, y_1)), \exists z. P(m_3, z) \rightarrow P(y_2, f(a, y_2)), \Phi \parallel \{(m_1, \{y_1\}), (m_2, \{y_2\}), (m_3, \{y_2\}), (m_4, \{\})\} \\
&\downarrow \alpha \\
P(m_1, m_2), P(m_3, m_4) &\vdash P(y_1, f(a, y_1)), \Psi, P(y_2, f(a, y_2)), \Phi \parallel \{(m_1, \{y_1\}), (m_2, \{y_2\}), (m_3, \{y_2\}), (m_4, \{\})\} \\
&\downarrow \left[ m_3 \leftarrow y_1 \right][m_4 \leftarrow f(a, y_1)]
\end{align*}
\]

Figure 2: Example Proof Tree

3 Implementational Details of \( \mathcal{F}A\textsc{U}ST \)

Since the proof construction process generates a tree, it is possible to either have a depth-first or a breadth-first construction procedure. The depth-first procedure continues to apply rules on the leftmost branch until it can be closed by computing a set of unifiers. These unifiers can then be refined by the unifiers found in the next leftmost branch so that the next branch is also closed. This process is repeated until all branches are closed. Details of the depth-first algorithm are given in [ScKK91a]. The breadth-first procedure on the other hand applies the rules on all the branches until a closing substitution can be obtained. If none can be found, each branch of the tree is extended by further rule applications.

The depth-first search is much faster than the breadth-first approach, since the breadth-first approach extends each branch even if it can already be closed by some unifier. On the other hand, the depth-first approach is inherently incomplete, while
the breadth-first search can prove each valid formula (except for complexity con-
straints).

**Optimizations.** If the above-mentioned technique for depth-first search is implemented naively, then the number of unifiers to be manipulated grows in an uncontrolled manner. In order to minimize the number of unifiers and to speed up the process of proof tree construction, the following enhancements have been made (see [Schn91] for details):

- Metavariables are managed locally within the branches. Thus if two unifiers only differ with respect to the instantiation of metavariables local to a branch, the unifiers become identical after leaving this branch.
- According to the generality of the unifiers only the most general ones need to be refined and all others can be removed.
- Antecedents and succedents are split up into four sets: $\alpha-$ and $\delta-, \beta-, \gamma-$ and for atomic formulae. For example, looking for an $\alpha-$rule now avoids searching through the whole sequent.
- All forbidden sets are stored in a global list outside the proof tree.
- There is no need to use proof trees at all. All the information to extend a branch or to search for a closing substitution is kept in the leaves. Hence, we work on a list of sequents rather than on a binary proof tree.

4 Experimental Results

The set of problems given by Pelletier [Pell86] has been proven and their runtimes (in seconds) on a SUN 4/330 are given in the table below. The first column contains the runtimes for a simple depth-first search, the second column contains the runtimes of a prover which skolemizes its input before the depth-first search and in the third column the runtimes of a breadth-first prover are given.

<table>
<thead>
<tr>
<th>$P_1$</th>
<th>$P_2$</th>
<th>$P_3$</th>
<th>$P_4$</th>
<th>$P_5$</th>
<th>$P_6$</th>
<th>$P_7$</th>
<th>$P_8$</th>
<th>$P_9$</th>
<th>$P_{10}$</th>
<th>$P_{11}$</th>
<th>$P_{12}$</th>
<th>$P_{13}$</th>
<th>$P_{14}$</th>
<th>$P_{15}$</th>
<th>$P_{16}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.1</td>
<td>0.0</td>
<td>0.1</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.1</td>
<td>0.1</td>
<td>0.1</td>
<td>0.1</td>
<td>0.1</td>
<td>0.1</td>
<td>0.1</td>
<td>0.1</td>
</tr>
<tr>
<td>0.3</td>
<td>0.1</td>
<td>0.3</td>
<td>0.3</td>
<td>0.3</td>
<td>0.3</td>
<td>0.3</td>
<td>0.3</td>
<td>0.3</td>
<td>0.3</td>
<td>0.3</td>
<td>0.3</td>
<td>0.3</td>
<td>0.3</td>
<td>0.3</td>
<td>0.3</td>
</tr>
<tr>
<td>0.2</td>
<td>0.2</td>
<td>0.2</td>
<td>0.4</td>
<td>0.4</td>
<td>0.4</td>
<td>0.4</td>
<td>0.4</td>
<td>0.4</td>
<td>0.9</td>
<td>0.7</td>
<td>0.7</td>
<td>0.7</td>
<td>0.4</td>
<td>0.4</td>
<td>0.4</td>
</tr>
<tr>
<td>1.6</td>
<td>0.3</td>
<td>0.7</td>
<td>0.8</td>
<td>2.5</td>
<td>3.5</td>
<td>3.5</td>
<td>3.5</td>
<td>3.5</td>
<td>5.6</td>
<td>2.5</td>
<td>2.5</td>
<td>2.5</td>
<td>5.4</td>
<td>5.4</td>
<td>5.4</td>
</tr>
<tr>
<td>$P_{17}$</td>
<td>$P_{18}$</td>
<td>$P_{19}$</td>
<td>$P_{20}$</td>
<td>$P_{21}$</td>
<td>$P_{22}$</td>
<td>$P_{23}$</td>
<td>$P_{24}$</td>
<td>$P_{25}$</td>
<td>$P_{26}$</td>
<td>$P_{27}$</td>
<td>$P_{28}$</td>
<td>$P_{29}$</td>
<td>$P_{30}$</td>
<td>$P_{31}$</td>
<td>$P_{32}$</td>
</tr>
<tr>
<td>0.2</td>
<td>0.2</td>
<td>0.3</td>
<td>0.3</td>
<td>0.4</td>
<td>0.2</td>
<td>0.2</td>
<td>0.3</td>
<td>0.4</td>
<td>0.9</td>
<td>0.7</td>
<td>0.7</td>
<td>0.7</td>
<td>0.4</td>
<td>0.4</td>
<td>0.4</td>
</tr>
<tr>
<td>1.6</td>
<td>1.6</td>
<td>1.6</td>
<td>1.6</td>
<td>1.6</td>
<td>1.6</td>
<td>1.6</td>
<td>1.6</td>
<td>1.6</td>
<td>1.6</td>
<td>1.6</td>
<td>1.6</td>
<td>1.6</td>
<td>1.6</td>
<td>1.6</td>
<td>1.6</td>
</tr>
<tr>
<td>$P_{33}$</td>
<td>$P_{34}$</td>
<td>$P_{35}$</td>
<td>$P_{36}$</td>
<td>$P_{37}$</td>
<td>$P_{38}$</td>
<td>$P_{39}$</td>
<td>$P_{40}$</td>
<td>$P_{41}$</td>
<td>$P_{42}$</td>
<td>$P_{43}$</td>
<td>$P_{44}$</td>
<td>$P_{45}$</td>
<td>$P_{46}$</td>
<td>$P_{47}$</td>
<td>$P_{48}$</td>
</tr>
<tr>
<td>0.7</td>
<td>17.6</td>
<td>0.4</td>
<td>1.6</td>
<td>1.6</td>
<td>2.5</td>
<td>0.4</td>
<td>2.7</td>
<td>0.7</td>
<td>0.7</td>
<td>0.4</td>
<td>1.2</td>
<td>5.7</td>
<td>184.0</td>
<td>0.5</td>
<td>0.7</td>
</tr>
<tr>
<td>0.7</td>
<td>0.4</td>
<td>0.8</td>
<td>1.6</td>
<td>1.6</td>
<td>3.5</td>
<td>0.4</td>
<td>0.7</td>
<td>0.7</td>
<td>0.7</td>
<td>0.4</td>
<td>1.2</td>
<td>2.5</td>
<td>170.4</td>
<td>0.5</td>
<td>0.7</td>
</tr>
<tr>
<td>2.4</td>
<td>***</td>
<td>0.4</td>
<td>1.0</td>
<td>1.0</td>
<td>9.6</td>
<td>0.4</td>
<td>1.4</td>
<td>0.7</td>
<td>0.7</td>
<td>0.4</td>
<td>1.2</td>
<td>24.4</td>
<td>9.9</td>
<td>0.7</td>
<td>0.7</td>
</tr>
</tbody>
</table>
Restricted Sequents: The introduction of metavariables introduces new problems as far as the application of the \( \delta \)-rules are concerned. A \( \delta \)-rule application requires that the variable introduced for the quantifier elimination is new, i.e. the variable must not occur in the sequent on which the \( \delta \)-rule is applied to. As the sequent may contain several metavariables which are not instantiated at this stage, one can not check whether a variable occurs after instantiation of the metavariables in the given sequent or not. The use of a \( \delta \)-rule restricts therefore the domain of the metavariables occurring in the sequent, because only terms which do not contain the variable introduced by the \( \delta \)-rule are allowed for instantiation of the metavariables. Hence, for each metavariable \( m \) of the sequent all variables introduced by further \( \delta \)-rule applications are stored in its so-called forbidden set \( f_{sm} \). The forbidden sets of all metavariables occurring in a sequent are kept in a restriction list which contains an ordered pair \((m, f_{sm})\) for each metavariable \( m \). Therefore, a restricted sequent has the form \( \Gamma \vdash \Delta \parallel R \).

Metasubstitution: As in \( SEQ \), the axiom scheme is a sequent with non-disjoint antecedent and succedent. Hence, given a restricted sequent \( \Gamma \vdash \Delta \parallel R \), we try to find a substitution \( \sigma \) of the metavariables such that \( \sigma(\Gamma) \cap \sigma(\Delta) \neq \{\} \) holds. Of course, the restriction has to be observed, in other words, for each metavariable \( m \) no variable of \( \sigma(m) \) is an element of \( f_{sm} \).

Given a restricted sequent \( \Gamma \vdash \Delta \parallel R \) one can compute a “closing substitution” by unifying each formula of the antecedent \( \Gamma \) with each formula of the succedent \( \Delta \). The underlying unification algorithm must therefore consider only metavariables as replaceable subterms and other variables are treated like constants. A closing substitution is a unifier which does not violate the restrictions.

Rules and Proof Trees in \( RSEQ \): Using the concepts of metavariables and restricted sequents the rules of \( SEQ \) are modified as follows: \( \alpha - \) and \( \beta - \)rules are quite the same as in \( SEQ \), except that the restrictions in the premise are copied to each conclusion of the rule. In the \( \gamma - \) and \( \delta - \) rules given in figure 1, both the variable \( y \) and the metavariable \( m \) have to be new, i.e. they do not appear in the sequent on which the rule is applied to. The function \( e_y \) updates the restrictions of the existing metavariables by adding \( y \) to each forbidden set.

\[
\begin{align*}
\forall x. \phi, \Gamma \vdash \Delta & \parallel R \\
\frac{[\phi]^m, \forall x. \phi, \Gamma \vdash \Delta \parallel \{(m, \{\})\} \cup R}{\Gamma \vdash \forall x. \phi, \Delta \parallel R} \\
\exists x. \phi, \Gamma \vdash \Delta & \parallel R \\
\frac{[\phi]^m, \Gamma \vdash \Delta \parallel e_y(R)}{\Gamma \vdash [\phi]^m, \exists x. \phi, \Delta \parallel \{(m, \{\})\} \cup R} \\
\end{align*}
\]

Figure 1: Quantifier rules of \( RSEQ \)
All the benchmarks except Schubert’s Steamroller Problem and the first-order formulae with equality (since our application domain does not require them) have been proven in acceptable times, although the implementation was done in a very naive way to obtain a prototype and the ML used for implementation is itself bootstrapped using an underlying Lisp. This implementation language has been chosen since our prover has been primarily developed for proving tedious lemmata in large HOL proofs.

5 Summary and Conclusions

We have shown how the integration of unification within sequent calculus can greatly improve its efficiency. This has been achieved by the introduction of the metavariables and its associated restrictions. The calculus thus developed (RSEQ) has been proven to be sound and complete and has been implemented in the HOL environment. The resulting prover FAUST is not only capable of solving first-order formulae, but also certain second-order formulae which occur in the domain of hardware verification. We have also implemented a function, which generates a HOL-tactic representing a proof by natural deduction. FAUST has also been embedded in an environment for proving Hardware called MEPHISTO [ScKK91b].

We are currently reimplementing the prover for use within the new HOL system based on Standard ML. We are also adding some new concepts which make the proof-process faster and are also going to incorporate some hardware domain specific rules within the calculus. Experimentation will also been undertaken with linear unification algorithms.

References


[Reev87] S. Reeves, Semantic tableaux as a framework for automated theorem-proving, Department of Computer Science and Statistics, Queen Mary College, Univ. of London, 1987


This article was processed using the \LaTeX macro package with LLNCS style