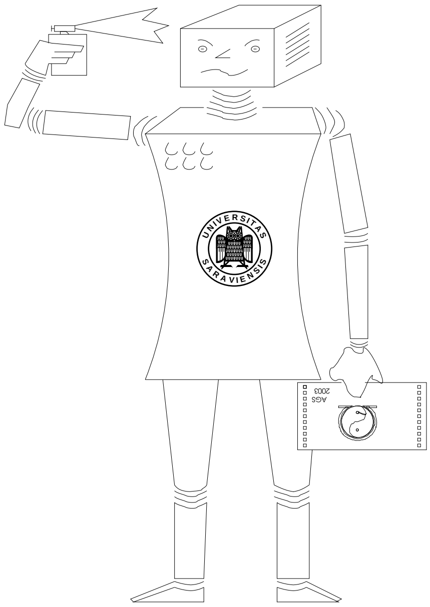
SEKI-REPORT ISSN 1437-4447

UNIVERSITÄT DES SAARLANDES
FACHBEREICH INFORMATIK
D–66123 SAARBRÜCKEN
GERMANY
WWW: http://www.ags.uni-sb.de/



An Approach to Assertion Application via Generalised Resolution

Quoc Bao Vo[1], Christoph Benzmüller[1] and Serge Autexier[1,2]
[1]FR Informatik, Universität des Saarlandes, 66041 Saarbrücken, Germany
{bao,chris}@ags.uni-sb.de
[2]DFKI GmbH, 66123 Saarbrücken, Germany
autexier@dfki.de

SEKI Report SR–2003–01

Editor of SEKI series:
Claus-Peter Wirth
FR Informatik, Universität des Saarlandes, D–66123 Saarbrücken, Germany
E-mail: `cp@ags.uni-sb.de`
WWW: `http://www.ags.uni-sb.de/~cp/welcome.html`

An Approach to Assertion Application via Generalised Resolution

Quoc Bao Vo[1], Christoph Benzmüller[1] and Serge Autexier[1,2]
[1]FR Informatik, Universität des Saarlandes, 66041 Saarbrücken, Germany
{bao,chris}@ags.uni-sb.de
[2]DFKI GmbH, 66123 Saarbrücken, Germany
autexier@dfki.de

July 22, 2003

Abstract

In this paper we address assertion retrieval and application in theorem proving systems or proof planning systems for classical first-order logic. Due to Huang the notion of assertion comprises mathematical knowledge such as definitions, theorems, and axioms. We propose a distributed mediator module between a mathematical knowledge base KB and a theorem proving system TP which is independent of the particular proof representation format of TP and which applies generalised resolution in order to analyze the logical consequences of arbitrary assertions for a proof context at hand.

Our approach is applicable also to the assumptions which are dynamically created during a proof search process. It therefore realises a crucial first step towards full automation of assertion level reasoning. We discuss the benefits and connection of our approach to proof planning and motivate an application in a project aiming at a tutorial dialogue system for mathematics.

1 Introduction

Mathematical assistant systems provide users with tools that support them in constructing proofs. Thereby it is an important issue that the constructed proof fragments are presented in a format that is easy to understand for human users. This is the central problem of proof presentation and explanation. On the other hand, sophisticated proofs for real mathematical problems have now been constructed in mathematical assistant systems (either automatically or interactively) at different levels of granularity[1]. In some of the employed approaches, most low level logical operations are abstracted away and mainly domain specific mathematical knowledge is used to guide the construction of the proofs.

[1]There is a ongoing debate whether *level of abstraction*, *level of detail*, or *level of granularity* is the most appropriate notion to use. In this paper we will use the term *level of granularity*, since we address abstraction on logical derivations and proof operators and not, for instance, abstraction of the object representation language.

The area is known as proof planning and such domain specific mathematical knowledge comes in as the strategies or proof methods which are made available to the proof planner (cf. [9] and [17, 21, 20] and the references therein).

The assertion level introduced by Huang [16] is argued to be just the right level to which machine generated proofs should be transformed before being presented to (human) users. In this paper we argue further that the assertion level can also serve as one of those levels of granularity on which knowledge-based proof planning should be based. This viewpoint is also taken by Autexier [3], which provides a proof representation framework subsuming the assertion level. However, proof automation of the framework, i.e. the heuristically guided choice of the assertions to apply, is still a challenge and not tackled in this paper.

Huang [16] achieves proofs at the assertion level by reconstructing machine generated proofs. This process basically involves clustering several proof steps of the input proof into segments. >From the outside, each segment can be viewed as one (assertion level) proof step. While this partly solves the problem of proof presentation as each assertion level proof step can be mapped naturally to single statements of the natural language presentation of the proof, it has not been investigated yet how the proof planning process may directly benefit from such a representation level. We are convinced that by planning directly on the assertion level it will be possible to overcome at least some of the identified limitations and problems of proof planning (see [5, 10])— in particular, those, that are caused by an unfortunate intertwining of proof planning and calculus level theorem proving. The perspective we therefore motivate in this paper is to consider the assertion level as a well chosen borderline between proof planning and machine oriented methods. Determining the logical consequences of assertions in a proof context is the task of machine oriented methods (in our case generalised resolution). The tasks on top of this level — for instance, a domain-dependent collection of the initial assertions to be considered, the heuristic selection of the most promising among the computed logical consequences, the introduction, constraining and handling of *meta-variables*, etc. — belong to the scope of domain specific proof planning.

In summary, instead of reconstructing natural deduction (ND) proofs to obtain assertion level proofs as suggested by Huang, we propose to directly plan for proofs at the assertion level. This should improve the quality of the resulting proof plans and also facilitate better user interaction. In contrast to tactical theorem proving, we do not want each assertion associated with a particular tactic but aim at one general module which works for arbitrary assertions and computes their consequences in arbitrary proof contexts. These consequences are then suggested to the user and/or the automated prover. Note that this is different from proposing intermediate subgoals and then verifying them with the help of tactics as done by systems like Mizar (`http://www.mizar.org/`).

The development of our ideas revolves around the mathematical assistant system Omega [27] and the current initiative in this project to rebuild the system on top of the proof representation framework in [3]. We furthermore employ Omega's agent-based search mechanism Oants [7] for a distributed modeling of our framework and we motivate an application of the approach in a project aiming at a tutorial dialogue system for mathematics.

2 Proof planning at the assertion level

A brief review on the assertion level.

This part is reproduced from [16] for completeness. Consider the problem of presenting machine generated proofs. Assume that the target formalism is natural deduction which is a common practice for many existing proof transformation systems. In contrast to proofs found in mathematical textbooks, proofs constructed by these systems are composed of derivations from elementary logic, where the focus of attention is on syntactic manipulations rather than on the underlying semantical ideas. The problem seems to come from the lack of intermediate structures in ND proofs that allow atomic justifications at a higher level of abstraction. Huang then went on to introduce the following three levels of justifications:

- *Logic level* justifications are simply verbalizations of the ND inference rules, such as the rule of Modus Ponens.
- *Assertion level* justifications account for derivations in terms of the application of an axiom, a definition or a theorem (collectively called an assertion). For instance, an extract from a textbook proof may read:

  > "since e is a member of the set A, and A is a subset of B, *according to the definition of subset*, e is a member of B".

- *Proof level* justifications are at a yet higher level. For instance, a proof can be suppressed by resorting to its similarity to a previous proof.

Assertion level and proof planning.

Scrutinizing the above example for the assertion level justifications, it seems just the right level for proof plans to be carried out. Rather than having to work around with ND inference rules, the proof planner simply goes straight to the intended conclusion, *viz.* e is a member of B, from the given premises, *viz.* e is a member of A and $A \subseteq B$, and the assertion, *viz.* the definition of subset. This would in the first place save the proof planner from exploding its own search space. Furthermore, it should fit better to the structural mechanisms of other high level proof methods such as analogy, induction, diagonalization, etc. and in particular island proof planning, cf. [19]. It should also fit well to the framework of *incremental proof planning* proposed by Gerberding and Pientka [15] in which assertion application can be considered as a special form of meta-rules.

Now let's consider an assertion $\mathcal{A}$. As pointed out by Huang, there may be several ways in which this assertion can be used depending on the proof situation to which this assertion is introduced. For instance, let $\mathcal{A}$ be the assertion from our running example:

$$\forall S_1, S_2 : Set.(S_1 \subseteq S_2 \Leftrightarrow \forall x : Element.(x \in S_1 \Rightarrow x \in S_2))$$

This assertion allows us to derive: (1) $a \in V$ from $a \in U$ and $U \subseteq V$, (2) $U \not\subseteq V$ from $a \in U$ and $a \notin V$, (3) $\forall x : Element.(x \in U \Rightarrow x \in V)$ from $U \subseteq V$, (4) etc.

A theorem prover/proof planner that operates on the calculus level can only achieve such conclusions after a number of proof steps to eliminate the quantifiers and other connectives such as implication and conjunction. On the other hand, most human mathematicians would be satisfied having those conclusions derived in one step from the assertion.

Huang [16] overcomes this problem by introducing different inference rules to allow all possible applications of such an assertion. These inference rules are called *assertion level inference rules* or *macro-rules* by their nature of being "pseudo inference rules" which actually stand for sequences of more elementary inference rules, called *compound proof segments*. While the more elementary rules (of ND) are chunked into macro-rules (at the assertion level), this in turn potentially introduces an exploded set of inference rules. Therefore, it may not work too well for a proof search mechanism that is required to be efficient. For instance, for the above assertion from our running example, at least seven (7) different macro-rules can be generated (cf. [16].)

Formalization of the problem.

We take as the starting point for our approach the proof development environment OMEGA [27] whose core consists of a proof planner together with a hierarchical plan data structure ($\mathcal{PDS}$). The proof format in OMEGA is based on the natural deduction calculus. A linearised version of ND proofs as introduced by Andrews [1] is employed. In this formalism [17], an ND proof is a sequence of proof lines , each of them is of the form:

$$\texttt{Label} \qquad \Delta \quad \vdash \quad \textit{derived-formula} \qquad (\texttt{Rule} \quad \textit{premise-lines})$$

where `Rule` is a rule of inference in ND or a method, which justifies the *derived-formula* using the knowledge from in the *premise-lines*. `Rule` and *premise-lines* together are called the justification of a line. Δ is a finite set of formulae which are the hypotheses the derived formula depends on.

A problem of proof planning consists of a theorem and the assumptions to be used to prove the theorem. A proof planner operates on a set of methods to be used to construct a proof plan. The theorem and assumptions are expressed as deduction lines in a $\mathcal{PDS}$ where all the assumptions are marked as *closed* and the theorem is marked as *open*. The proof planner then uses the methods to come up with actions to update the $\mathcal{PDS}$. The aim of the proof planning process is to reach a closed $\mathcal{PDS}$, that is one without open lines.

As application of lemmata lies at heart of most (non-trivial) mathematical proofs, it is important that this issue be addressed in a realization of any proof planner.

In OMEGA, a proof planning process starts with a task, a data structure designed to encapsulate a complete (sub-)problem. Formally, a *task* is a pair $\langle SP_{L_{open}}, L_{open}\rangle$ consisting of an open line of the $\mathcal{PDS}$, L_{open}, and a set of lines from the $\mathcal{PDS}$, $SP_{L_{open}}$. L_{open} is called the *task line* whose formula is called *task formula*. Members of $SP_{L_{open}}$ are called the *support lines* or *supports* for the task line L_{open}.

Now consider the situation in which the prover is confronting a list of tasks, called an *agenda* in the OMEGA system, that it needs to solve in order to prove the intended theorem. Among the available strategies/methods, the prover could possibly ponder whether there is a definition/axiom or some previously proved result which it can use in the current proof situation to (i) either obtain further closed lines serving as intermediate steps for solving one of the tasks; (ii) or reduce a goal task (on some open line) to some subtasks which can be resolved by further proof steps.

α	α_1	α_2
$(\varphi \vee \psi)^{\oplus}$	$\varphi^{\oplus}$	$\psi^{\oplus}$
$(\varphi \Rightarrow \psi)^{\oplus}$	$\varphi^{\ominus}$	$\psi^{\oplus}$
$(\varphi \wedge \psi)^{\ominus}$	$\varphi^{\ominus}$	$\psi^{\ominus}$
$(\neg\varphi)^{\oplus}$	$\varphi^{\ominus}$	–
$(\neg\varphi)^{\ominus}$	$\varphi^{\oplus}$	–

β	β_1	β_2
$(\varphi \wedge \psi)^{\oplus}$	$\varphi^{\oplus}$	$\psi^{\oplus}$
$(\varphi \vee \psi)^{\ominus}$	$\varphi^{\ominus}$	$\psi^{\ominus}$
$(\varphi \Rightarrow \psi)^{\ominus}$	$\varphi^{\oplus}$	$\psi^{\ominus}$

(γ, x)	$\gamma_0(x)$
$(\forall_x \varphi)^{\ominus}$	$\varphi^{\ominus}$
$(\exists_x \varphi)^{\oplus}$	$\varphi^{\oplus}$

(δ, x)	$\delta_0(sko)^{(*)}$
$(\forall_x \varphi)^{\oplus}$	$\varphi[sko/x]^{\oplus}$
$(\exists_x \varphi)^{\ominus}$	$\varphi[sko/x]^{\ominus}$

$^{(*)}$ *sko* is a *Skolem term.*

Figure 1: Analyzing signed formulae

It now boils down to the question of how tasks and assertions are to be handled without having to generate all the possible macro-rules as suggested by Huang. And we, of course don't want to sacrifice any possible outcome from applications of assertions.

We follow a representation framework employed by Wallen [29] and Autexier [3] using the concept of polarities and uniform notation (cf. [12, 13, 29]) to represent the formulae the proof planner has to deal with.

Signed formulae:

Each formula is assigned a polarity which is either positive ($\oplus$) or negative ($\ominus$). A set of formulae Γ is assigned a polarity p iff each member of Γ is assigned p. The polarity $(\ominus, \oplus)$ of a signed formula is just another representation of of the succedent/antecedent made by Gentzen [14] wrt. the sequent calculus. In the sequent calculus, all formulae which occur in the antecedent would be annotated with negative ($\ominus$) polarity while formulae occurring in the succedent are of positive ($\oplus$) polarity. In the linearised version of the Natural Deduction calculus employed by Omega, if a deduction line $\Delta \vdash \varphi$ is marked closed (in a $\mathcal{PDS}$) then Δ is assigned $\oplus$ and φ $\ominus$. If $\Delta \vdash \varphi$ is marked open then Δ is assigned $\ominus$ and φ $\oplus$. Intuitively, a signed formula $\varphi^{\oplus}$ is something the prover wishes to prove while $\varphi^{\ominus}$ is something given.

We introduce further notations for technical considerations:

(i) $form(\varphi^p) \stackrel{\text{def}}{=} \varphi$ and $polarity(\varphi^p) \stackrel{\text{def}}{=} p$; and

(ii) the operator $-: \{\ominus, \oplus\} \to \{\ominus, \oplus\}$ such that $-\ominus = \oplus$ and $-\oplus = \ominus$.

Signed formula trees: Using the rules from Fig. 1, a signed formula can be represented in the form of a tree whose sub-trees represent its signed sub-formula, these trees are called *signed formula trees*, or *SFT*s. In such a tree, each node has a natural interpretation as a signed formula, and we shall speak of nodes and signed formulae as if they were one and the same (e.g., a "node" $(\varphi \vee \psi)^{\oplus}$ is one labelled with $(\varphi \vee \psi)^{\oplus}$ and having arcs to nodes $\varphi^{\oplus}$ and $\psi^{\oplus}$.)

Example:[2]

The following is the signed formula tree of the formula $\mathcal{A}$ given as $\forall_{S_1,S_2}(S_1 \subseteq S_2 \Leftrightarrow \forall_x(x \in S_1 \Rightarrow x \in S_2))$. Thereby we assume that assumptions are initially assigned with the polarity $\ominus$. (In our tree F abbreviates the formula $\forall_x(x \in S_1 \Rightarrow x \in S_2)$).

[2]We choose to suppress all references to sorts for the sake of readability.

$$\begin{tikzcd}[column sep=tiny]
 & & & (\forall_{S_1,S_2}(S_1 \subseteq S_2 \Leftrightarrow \forall_x(x \in S_1 \Rightarrow x \in S_2)))^{\ominus}_{(\gamma,S_1,S_2)} \arrow[d] & & & \\
 & & & (S_1 \subseteq S_2 \Leftrightarrow F)^{\ominus}_{\alpha} \arrow[dll] \arrow[drr] & & & \\
 & (S_1 \subseteq S_{\Rightarrow} F)^{\ominus}_{\beta} \arrow[dl] \arrow[dr] & & & & (F \Rightarrow S_1 \subseteq S_2)^{\ominus}_{\beta} \arrow[dl] \arrow[dr] & \\
(S_1 \subseteq S_2)^{\oplus} & & (F)^{\ominus}_{(\gamma,x)} \arrow[d] & & (F)^{\oplus}_{(\delta,x)} \arrow[d] & & (S_1 \subseteq S_2)^{\ominus} \\
 & & (x \in S_1 \Rightarrow x \in S_2)^{\ominus}_{\beta} \arrow[dl] \arrow[d] & & (sko(S_1,S_2) \in S_1 \Rightarrow sko(S_1,S_2) \in S_2)^{\oplus}_{\alpha} \arrow[d] \arrow[dr] & & \\
 & (x \in S_1)^{\oplus} & (x \in S_2)^{\ominus} & & (sko(S_1,S_2) \in S_1)^{\ominus} & (sko(S_1,S_2) \in S_2)^{\oplus} &
\end{tikzcd}$$

Now we can go back to our problem. Assume that a task $\langle SP_{L_{open}}, L_{open}\rangle$ together with an assertion $\mathcal{A}$ are currently under consideration. As the goal is to come up with a desirable conclusion derivable from the assertion $\mathcal{A}$ (relative to $\langle SP_{L_{open}}, L_{open}\rangle$), the following computational process revolves around the signed formula tree of $\mathcal{A}$, called $\mathcal{T}_{\mathcal{A}}$. Using unification-based algorithms we unify the signed atoms at the leaves of the signed formula trees from the task $\langle SP_{L_{open}}, L_{open}\rangle$ with the (signed) atoms at the leaf nodes of $\mathcal{T}_{\mathcal{A}}$.

Secondly, we will need the following notation to talk about the interrelations between formulae: Given a signed formula φ^p,

- if an α-rule is applicable to φ^p then its children are said to be *directly α-related*, e.g. the two sub-formulae $\varphi^{\oplus}$ and $\psi^{\oplus}$ of $(\varphi \vee \psi)^{\oplus}$ are α-related;
- if a β-rule is applicable to φ^p then its children are said to be *directly β-related*, e.g. the two sub-formulae $\varphi^{\oplus}$ and $\psi^{\oplus}$ of $(\varphi \wedge \psi)^{\oplus}$ are β-related;

A signed formula φ is α (resp. β) *-related* to ψ if and only if either φ or one of its ancestors is directly α (resp. β) -related to ψ.[3]

s-unifiable and complementary:

Two nodes φ^p and ψ^q are *s-unifiable* iff $p \neq q$ and φ and ψ are unifiable. φ^p and φ^q are *complementary* iff $p \neq q$.

NOTATION: We will liberally use the notions applicable to logical formulae and apply them to signed formulae by absorbing them through the polarities of signed formulae. E.g., given a signed formula τ and a substitution θ, we write $\tau\theta$ to denote the signed formula $(form(\tau)\theta)^{polarity(\tau)}$.

We also observe that it is possible to reconstruct formulae from an SFT in such a way that the semantics is preserved. And because all information is conveyed from a node to its children, it is possible to reconstruct a formula from the leaf nodes of its SFT together

[3]Note that the notions of α and β-related formulae are more general than the notions of conjunction and disjunction as the α (resp. β)-relations can refer to both conjunctions and disjunctions depending on whether they occur in the antecedent or the succedent of a sequent.

with the α and β -relations between the subtrees without having to trace back through all of the internal nodes.

Now we come up with the algorithm for manipulating SFTs in order to obtain desirable conclusions from an assertion (in relation to other given assumptions or open goal tasks). The idea generalises Robinson's resolution principle [25] and in particular reminds to path resolution [23] and the connection graph principle [18, 2, 8]. In much the same way that an arbitrary formula can be reduced to the normal form which contains a set of clauses, i.e. disjunctions of literals, we observe that a given signed formula tree can be reduced to a set of SFTs whose members don't have any α-related pair of sub-formulae.

As with resolution, the idea is to replace a literal in a clause (correspondingly, a leaf node of an SFT) by the information entailed by it that comes from another clause (correspondingly, another SFT). However, unlike resolution theorem proving formalism where the focus is on a refutation procedure in order to establish the unsatisfiability of a set of formulae, our focus of attention is on the derivation of new formulae (be it new assumptions or new subgoals) from the existing ones. Therefore, we won't transform every signed formula (or equivalently, SFT) to the normal form which contains only β-related subtrees as in resolution theorem proving, but rather we delay that until as late as possible. The idea is: instead of normalizing all formulae to the normal form, we just need to take out from the formulae to be resolved the *relevant parts* when resolution is carried out. For instance, consider a set of formulae $S = \{(A \wedge B) \vee (C \wedge D), \neg A \vee E\}$, we can just take out the relevant formula $A \vee (C \wedge D)$ from $(A \wedge B) \vee (C \wedge D)$ and resolve it using $\neg A \vee E$. This result in the following new set of formulae: $S' = \{(A \wedge B) \vee (C \wedge D), \neg A \vee E, (C \wedge D) \vee E\}$.

Now we can go back to the more sophisticated representation of SFTs. >From the above example with resolution, it is important to know how the so-called relevant parts can be computed. While it is not so clear how this can be done with logical formulae, it is interesting that SFTs contain structures supporting such operations: Instead of resolving complementary pairs of literals from clauses, we resolve s-unifiable pairs of leaf nodes (or, atomic signed formulae) from SFTs. We therefore term our inference principle *generalised resolution*, or *GR*.

We observe that in addition to the leaf nodes of SFTs on which we perform GR, the structure of SFTs, i.e. the internal nodes and links between nodes, also plays an important part in GR inferences.[4]

The algorithm:

The following algorithm takes as input two SFTs τ_1 and τ_2 with τ_1 being updated by the information from τ_2 by performing resolution on the two leaf nodes η_1 and η_2 from τ_1 and τ_2, resp., under the substitution θ.

GR(τ_1: SFT, η_1: leaf_node; τ_2: SFT, η_2: leaf_node; θ: substitution)

begin

[4]Given a pair of s-unifiable leaf nodes φ^p and ψ^{-p} on two SFTs τ_1 and τ_2, respectively: the results of resolving φ^p on τ_1 by ψ^{-p} from τ_2 and resolving ψ^{-p} on τ_2 by φ^p from τ_1, in general lead to two syntactically different, albeit logically equivalent, SFTs.

If $\eta_1\theta$ and $\eta_2\theta$ are complementary **then**

1. Instantiate all variables in τ_1 and τ_2 with the substitution θ;
2. For $i = 1, 2$, prune all α-related subtrees wrt. η_i on τ_i;
3. Replace the leaf node η_2 in τ_2 by $(\bot)^{\ominus}$ (resp. $(\top)^{\oplus}$) if $polarity(\eta_2) = \ominus$ (resp. $\oplus$);
4. Replace the formula at each ancestor of η_2 in τ_2 with a question mark (?) while retaining the polarity and the rule originally applied there;
5. Reconstruct the formulae at the internal nodes of τ_2 currently marked (?) using the formula(e) from their child(ren) and the polarity/rule at those nodes;
6. Replace the formula at each ancestor of η_1 in τ_1 with a question mark (?) while retaining the polarity and the rule originally applied there;
7. **If** $polarity(\tau_2) = polarity(\eta_1)$ **then:** Replace the leaf node η_1 on τ_1 by τ_2
   else: Replace the leaf node η_1 on τ_1 by $\neg form(\tau_2)^{polarity(\eta_1)}$;
8. Reconstruct the formulae at the internal nodes of τ_1 currently marked (?) using the formula(e) from their child(ren) and the polarity/rule at those nodes;
9. **Return** the new τ_1;

else

{ Do nothing }

end;

All the above operations should be clear except the reconstruction of the SFTs. Consider an internal node $\tau = (\varphi)_r^p$ of an SFT. After a subtree of τ is modified during the execution of $GR()$, the formula at τ, *viz.* φ, is replaced by a question mark (?). Once all children of τ have been reconstructed, we can proceed to reconstruct the formula at τ:

1. Special treatment is due when at least one of τ's children is $(\bot)^{\ominus}$ or $(\top)^{\oplus}$:
   (a) Case $r = \beta$: let υ be the other child of τ. If $polarity(\tau) = polarity(\upsilon)$ then τ and υ are collapsed into one node; otherwise, (?) is replaced by $\neg form(\upsilon)$.
   (b) otherwise: the question mark (?) is replaced by $(\bot)^{\ominus}$ or $(\top)^{\oplus}$ depending on $polarity(\tau)$.
2. Case $r = \alpha$:
   - Case $\alpha_2 = nil$: the question mark (?) is replaced by $\neg form(\alpha_1)$.
   - Case $\alpha_2 \neq nil$: the question mark (?) is replaced by $form(\alpha_1) \wedge form(\alpha_2)$ if $p = \ominus$, and by $form(\alpha_1) \vee form(\alpha_2)$, otherwise.
3. Case $r = \beta$: similar to the second case above and according to the table for β-rule.
4. Case $r = (\gamma, x)$: Attention must be taken to avoid clash of variables. If x is a bound variable in $form(\gamma_0(x))$ then it must be renamed to a variable name that does not clash with any other variable in $form(\gamma_0(x))$. Then, (?) is replaced by $\forall_x form(\gamma_0(x))$ if $p = \ominus$, and by $\exists_x form(\gamma_0(x))$, otherwise.

5. Case $r = (\delta, x)$: the question mark (?) is replaced by $form(\delta_0(\tau, sko))$.

Correctness of algorithm GR together with other related results are established in the Appendix.

Now we are in the position to apply GR on the assertion $\mathcal{A}$ and all the relevant formulae coming from an agenda $\mathcal{D}$. We are fortunate not having to perform proof search, we only have to find out derivable information (which can be new assumptions, i.e. closed lines, or new subgoals, i.e. open lines) from $\mathcal{A}$ using the information provided in $\mathcal{D}$. That is, we have to resolve the leaf nodes of $\mathcal{T}_\mathcal{A}$ by the s-unifiable leaf nodes on other SFTs. To simplify the presentation, we assume that our algorithm takes as input the SFT of $\mathcal{A}$, *viz.* $\mathcal{T}_\mathcal{A}$, and a set of SFTs representing the current reasoning context $\mathcal{RC}$ so that we don't have to worry about other details such as closed/open lines, assumptions, goals, etc. The algorithm also employs the on-the-fly *skolemization* whenever generalised resolution is performed.

Assertion Application($\mathcal{T}_\mathcal{A}$: SFT; $\mathcal{RC}$: set_of_SFTs)

begin

1. $All\_results \leftarrow \emptyset$;
2. $Processed \leftarrow \emptyset$;
3. **For** each leaf node s of $\mathcal{T}_\mathcal{A}$ **do:**

   While there is a leaf node t of an SFT $\tau \in \mathcal{RC} \cup \{\mathcal{T}_\mathcal{A}\}$ such that s and t are s-unifiable and $(s, t, \theta) \notin Processed$ **do:**

   (a) $C_{\mathcal{T}_\mathcal{A}} \leftarrow \mathcal{T}_\mathcal{A}$; { Create a working copy of $\mathcal{T}_\mathcal{A}$ }
   (b) $p \leftarrow polarity(root(\tau))$;
   (c) **Repeat:**
      i. $current\_resolved \leftarrow s$;
      ii. $current\_resolver \leftarrow t$;
      iii. $\theta \leftarrow mgu(form(s), form(t))$;
      iv. $Processed \leftarrow Processed \cup \{(s, t, \sigma)\}$;
      v. **Repeat:**
         A. $\tau \leftarrow GR(C_{\mathcal{T}_\mathcal{A}}$, *current_resolved*, τ, *current_resolver*, $\theta)$;
         B. **If** there is a s-unifiable pair of leaf nodes s' and t' from $C_{\mathcal{T}_\mathcal{A}}$ and τ, respectively, **then**
            - $current\_resolved \leftarrow s'$;
            - $current\_resolver \leftarrow t'$;
            - $\sigma \leftarrow mgu(form(s'), form(t'))$;
            - $\theta \leftarrow \theta\sigma$;
            - $Processed \leftarrow Processed \cup \{(s', t', \sigma)\}$;

            else
            - $current\_resolved \leftarrow nil$;
            - $current\_resolver \leftarrow nil$;

         until $current\_resolved = nil$;

vi. **If** $p = \oplus$ and $polarity(root(\tau)) = \ominus$ **then**
$C_{\mathcal{T_A}} \leftarrow Negate(\tau)$;
else
$C_{\mathcal{T_A}} \leftarrow \tau$;

until there is not any further s-unifiable pair of leaf nodes from $C_{\mathcal{T_A}}$ and any SFT in $\mathcal{RC} \cup \{\mathcal{T_A}\}$;

(d) $All\_results \leftarrow All\_results \cup \{form(root(C_{\mathcal{T_A}}))\}$;

4. **Return** $All\_results$;

end;

In the above algorithm, the function $Negate(\tau)$ adds a parent to the root node of the SFT τ. The formula and the polarity at the root node of the new tree are the negations of the formula and the polarity at the root node of τ. We have to carry out that operation because we do not want the outcome of applying an assertion to an open line, i.e. a goal, to be a closed line. For instance, applying the assertion $A \Rightarrow B$ to a goal $B^{\oplus}$ should result in a new subgoal $A^{\oplus}$ rather than a conclusion $(\neg A)^{\ominus}$.

Example (Forward Reasoning):

Let the following deduction lines be marked closed in the input task:

(1) $\vdash A \subseteq B$; and

(2) $\vdash e \in A$.

By applying the first tree $A \subseteq B^{\ominus}$ to $C_{\mathcal{T_A}}$ which is the same as $\mathcal{T_A}$ for the moment, we obtain the following updated SFT for $C_{\mathcal{T_A}}$ (under the substitution $\theta = \{S_1 \mapsto A, S_2 \mapsto B\}$).

$$\begin{tikzcd}[column sep=tiny]
 & (\forall_x(x \in A \Rightarrow x \in B))^{\ominus}_{\gamma} \arrow[d] & \\
 & (x \in A \Rightarrow x \in B)^{\ominus}_{\beta} \arrow[dl] \arrow[dr] & \\
(x \in A)^{\oplus} & & (x \in B)^{\ominus}
\end{tikzcd}$$

Figure 2: The SFT for $(\forall_x(x \in A \Rightarrow x \in B))^{\ominus}$

By applying the second tree $e \in A^{\ominus}$ to the resulting $C_{\mathcal{T_A}}$, we obtain as updated SFT for $C_{\mathcal{T_A}}$ (under the substitution $\theta_1 = \{x \mapsto e\}$) the tree $e \in B^{\ominus}$ which is translated back to the OMEGA-proof format as a (closed) deduction line (3) $\vdash e \in B$. This is also is also the desirable logical consequence of assertion $\mathcal{A}$ for lines (1) and (2).

Example (Resolve internal nodes):

To see the necessity of the inner iteration, we re-run our example with another task consisting of the following closed deduction line: (4) $\vdash \forall_x(x \in A \Rightarrow x \in B)$. This assumption results in the SFT depicted in Figure 2. Applying this tree to $\mathcal{T_A}$ and assuming that we unify first the two nodes $(sko(S_1, S_2) \in S_1)^{\ominus}$ (on $\mathcal{T}_{\mathcal{A}}$) and $(x \in A)^{\oplus}$ on the above SFT we obtain the updated $C_{\mathcal{T}_{\mathcal{A}}}$ below (under the substitution $\theta = \{x \mapsto sko(S_1, S_2), S_1 \mapsto A\}$).

$$\begin{tikzcd}
& (\neg sko(S_1, S_2) \in B \Rightarrow A \subseteq S_2)^{\ominus}_{\beta} \arrow[dl] \arrow[dr] & \\
(\neg sko(S_1, S_2) \in B)^{\oplus}_{\alpha} \arrow[d] & & (A \subseteq S_2)^{\ominus} \\
(sko(S_1, S_2) \in B)^{\ominus} & &
\end{tikzcd}$$

Notice how the α-related branches on the SFT and the nodes labelled with the α and γ and δ-rules are pruned away by $GR()$.

Now we try to unify a leaf node on this tree to some leaf node on $\mathcal{T}_{\mathcal{A}}$ again which is achievable through the unification on the two nodes $(sko(S_1, S_2) \in B)^{\ominus}$ on the above SFT and $(sko(S_1, S_2) \in S_2)^{\oplus}$ (on $\mathcal{T}_{\mathcal{A}}$).

The updated $C_{\mathcal{T}_{\mathcal{A}}}$ now becomes the tree displayed to the right (under the substitution $\theta_1 = \{S_2 \mapsto B\}$)[5].

$$\begin{tikzcd}
& (\neg A \subseteq B \Rightarrow A \subseteq B)^{\ominus}_{\beta} \arrow[dl] \arrow[dr] & \\
(\neg A \subseteq B)^{\oplus}_{\alpha} \arrow[d] & & (A \subseteq B)^{\ominus} \\
(A \subseteq B)^{\ominus} & &
\end{tikzcd}$$

This tree translates into the desirable logical consequence of assertion $\mathcal{A}$ for line (4) and the closed line (5) $\vdash A \subseteq B$ is obtained.

Example (Sidewards Reasoning):

We now consider an example in which a goal line (i.e. an open line from the input task) and a closed line is applied. Assume that our current task consists of an open line (6) $\vdash \epsilon \in B$ and a closed line (7) $\vdash A \subseteq B$, which correspond to the two singleton trees $(\epsilon \in B)^{\oplus}$ and $(A \subseteq B)^{\ominus}$ respectively.

Applying the former to $\mathcal{T}_{\mathcal{A}}$ by unifying it to the leaf node $(x \in S_2)^{\ominus}$, we obtain the following SFT:

$$\begin{tikzcd}
& (\forall_{S_1}(S_1 \subseteq B \Rightarrow \neg \epsilon \in S_1))^{\ominus}_{\gamma} \arrow[d] & \\
& (S_1 \subseteq B \Rightarrow \neg \epsilon \in S_1)^{\ominus}_{\beta} \arrow[dl] \arrow[dr] & \\
(S_1 \subseteq B)^{\oplus} & & (\neg \epsilon \in S_1)^{\ominus}_{\alpha} \arrow[d] \\
& & (\epsilon \in S_1)^{\oplus}
\end{tikzcd}$$

[5] Note that the root node of the computed SFT is: $(A \subseteq B)^{\ominus}$ has a different polarity from that of the node it is going to replace, *viz.* $(sko(S_1, S_2) \in S_2)^{\oplus}$, on the SFT $\mathcal{T}_{\mathcal{A}}$. Therefore, we replace that node with $(\neg A \subseteq B)^{\oplus}$ instead.

Then we can apply $(A \subseteq B)^{\ominus}$ to this SFT by unifying it to the node $(S_1 \subseteq B)^{\oplus}$, the resulting tree is:

$$
\begin{tikzcd}
(\neg \epsilon \in A)^{\ominus}_{\alpha} \arrow[d] \\
(\epsilon \in A)^{\oplus}
\end{tikzcd}
$$

This corresponds to a new open line (8) $\vdash \epsilon \in A$, which is the desirable consequence of assertion $\mathcal{A}$ for lines (6) and (7).

3 Concurrent search for applicable assertions

In this section we propose a module called M which models assertion application as distributed search processes in the OANTS approach [7]. This agent based formalism is the driving force behind a distributed proof search approach in OMEGA. It enables the distribution of proof search among groups of reasoning agents.

First we briefly sketch the general application scenario that motivates our approach. We assume a scenario where a theorem prover TP is connected to a mathematical knowledge base KB. TP is currently focusing a proof task $\mathcal{T} = \langle SP_{L_{open}}, L_{open} \rangle$ and candidate assertions $\{\mathcal{B}_i\}$ are determined in KB and handed over to our assertion module M. The task of M is to compute wrt. proof task $\mathcal{T}$ all possible logical consequences of the available assertions $\mathcal{B}_i$.

We propose to create for each assertion $\mathcal{B}_i$ one associated instance $AG_{\mathcal{B}_i}$ of a generic *assertion agent* AG. The generic assertion agent AG is based on the algorithm *AssertionApplication* provided in this paper. Note that this algorithm only depends on the SFT of the focused assertion and a further set of SFTs for the proof context, and both are specified as parameters of *AssertionApplication*. Each assertion agent instance $AG_{\mathcal{B}_i}$ computes and suggests the logical consequences of $\mathcal{B}_i$ in proof context $\mathcal{T}$ to our module M which passes them further to TP.

$$
\begin{tikzcd}
 & (\forall_{S_1}(S_1 \subseteq \wp(C) \Rightarrow \neg A \in S_1))^{\ominus}_{\gamma} \arrow[d] & \\
 & (S_1 \subseteq \wp(C) \Rightarrow \neg A \in S_1)^{\ominus}_{\beta} \arrow[dl] \arrow[dr] & \\
(S_1 \subseteq \wp(C))^{\oplus} & & (\neg A \in S_1)^{\ominus}_{\alpha} \arrow[d] \\
 & & (A \in S_1)^{\oplus}
\end{tikzcd}
$$

Depending on the size of the knowledge base KB there could be too many applicable assertions passed to M and also too many ways an assertion can be applied to be handled in practice (recall the remark above about the number of possible ND inference rules associated with the assertion of our running example). For instance, going back to our running example again, let our current task be $\langle \{\Delta \vdash A \subseteq B; \quad \Delta \vdash B \subseteq C;\}, A \in \wp(C) \rangle$. The assertion $\mathcal{A}$ is applicable in this situation and the outcome of applying our algorithm to this open line is the SFT displayed at the top right corner of this paragraph.

As the root node is non-empty and the input trees include an open line, we have to reproduce the root node to have polarity $\oplus$, which is $(\neg\forall_{S_1}(S_1 \subseteq \wp(C) \Rightarrow \neg A \in S_1))^{\oplus}$. This corresponds to a new open line: $\Delta \vdash \exists_{S_1}(S_1 \subseteq \wp(C) \wedge A \in S_1)$ which is, even though logically correct, not a very useful subgoal to be pursued.[6]

We sum up the above argument by claiming that restricted application of assertion is necessary. One possible and simple restriction is to impose prerequisite(s), such as simple syntactical criteria or domain restrictions, when selecting the candidate assertions that are passed from KB to M. These kind of specific constraints can be realized provided the constraints are encoded in the assertion specification. For instance, regarding our running example, a simple but useful heuristic to prove the membership of an object wrt. a set S using the subset definition is that the task includes a closed line stating that S is a superset of some other set. It is such a heuristically constrained version of assertion application that we are currently employing in the OMEGA system.

Proof planning, however, has developed more sophisticated ways to guide and constrain possible instantiations and applications of assertions. The investigation on how some of these techniques can optimally be employed on top of our assertion application module M in order to better constrain the search for potential assertion candidates is further work.

Example.

We present a proof constructed in the OMEGA system using assertion agents. The initial $\mathcal{PDS}$ formulating the problem is as follows:

1.	1;	$\vdash$	$symmetric(A)$	Hyp ✓
2.	2;	$\vdash$	$symmetric(B)$	Hyp ✓
THM	1,2;	$\vdash$	$symmetric(A \cap B)$	(?)

In the above $\mathcal{PDS}$, the lines marked with ✓ are the closed lines and the open line is marked with the question mark (?).

The following proof (produced interactively in OMEGA) for the above problem uses *the definition of the symmetric relation, viz.* (Sym-Def): $\forall_R symmetric(R) \Leftrightarrow (\forall_{x,y}\langle x,y\rangle \in R \Rightarrow \langle y,x\rangle \in R)$, and *the definition of intersection (on sets), viz.* ($\cap$-Def): $\forall_{S_1,S_2}\forall_x x \in S_1 \cap S_2 \Leftrightarrow x \in S_1 \wedge x \in S_2$. The successive order of the constructed proof lines is: $\langle$ 1.; 2.; THM; 3.; 4.; 8.; 5.; 6.; 7.; $\rangle$

1.	1;	$\vdash$	$symmetric(A)$	(Hyp) ✓	
2.	2;	$\vdash$	$symmetric(B)$	(Hyp) ✓	
3.	1,2;	$\vdash$	$\forall_{x,y}\langle x,y\rangle \in A \Rightarrow \langle y,x\rangle \in A$	(*aa*(Sym-Def)	1) ✓
4.	1,2;	$\vdash$	$\forall_{x,y}\langle x,y\rangle \in B \Rightarrow \langle y,x\rangle \in B$	(*aa*(Sym-Def)	2) ✓
5.	5;	$\vdash$	$\langle c_1,c_2\rangle \in A \wedge \langle c_1,c_2\rangle \in B$	(Hyp) ✓	
6.	1,2,5;	$\vdash$	$\langle c_2,c_1\rangle \in A$	(*aa*([3])	5) ✓
7.	1,2,5;	$\vdash$	$\langle c_2,c_1\rangle \in B$	(*aa*([4])	5) ✓
8.	1,2;	$\vdash$	$\forall_{x,y}\langle x,y\rangle \in A \cap B \Rightarrow \langle y,x\rangle \in A \cap B$	(*aa*($\cap$-Def)	5,6,7) ✓
THM	1,2;	$\vdash$	$symmetric(A \cap B)$	(*aa*(Sym-Def)	8) ✓

[6]In fact, Huang's [16] framework fails to produce the above inference step. This poses a serious question about the completeness of his approach due to Huang's proposal of replacing the natural deduction inference rules altogether by the generated macro rules whenever an assertion is applicable.

[Hyp] indicates that the given proof line is a given hypothesis. The scopes of the hypotheses are rendered by the antecedents of the proof lines, i.e. those on the left of $\vdash$). Rules denoted by aa(`assertion`) indicate the application of `assertion`. In particular, on line (6), the `assertion` is `[3]` which indicates that the succedent of the sequent on line (3) has been added to the current mathematical database and now serves as a normal assertion. Similar remark can be made for line (7) (and the assertion introduced on line (4)).

While the above proof may be quite intuitive to a mathematician, the proof step on line (8) may be too hard to understand for a student who is trying to learn mathematics (see the Application described below). Observe that the assertion agent has carried out the application of `Sym-Def` together with $\supset$-Introduction and $\forall$-introduction in that single proof step. However, the OMEGA system is capable of expanding complex proof steps to yield sub-proofs at lower levels of abstraction. This again emphasises the expressive power of our machinery.

An application: The DIALOG project.

We are currently experimenting the assertion agent modules described above in the setting of an intelligent tutoring system for mathematics developed within the DIALOG project [4]. The simple proof steps to be presented to the students are automatically generated by the proof assistant system OMEGA. The assertion agents assists the OMEGA system to produce these assertion-level proof steps. We want to stress that the student model may be updated during a tutoring session, hence the set of relevant assertions may dynamically change during an interactive session.

It is easy to motivate the design of our assertion application module for this scenario. Its capabilities for assertion application for a dynamically varying set of assertions are crucial for the project. It is also essential that reasoning is facilitated at a human oriented level of granularity, since we do not want the user to puzzle around with the peculiarities of, for instance, logical derivations in sequent or natural deduction calculus.

4 Related Work

Autexier [3], which himself employs ideas from [29, 26], suggests a proof representation and manipulation framework based on the uniform notation idea, while we here only employ it at the heart of an assertion mediator module which can be coupled with arbitrary proof systems.

The generalised resolution algorithm we propose also reminds to ideas employed in the connection graph method [18, 2, 8] and the *clause graph method* in [24]. However, since it is a crucial concern for us not to break down the structure of the assertions we operate with tree or graph structures generated for arbitrary formulas instead of just clause sets. Works in the literature of automated reasoning that are more closely related to our generalised resolution inference method include Non-clausal (NC)-Resolution introduced by Neil Murray [22] and Nested Resolution introduced by Traugott [28]. Both frameworks also employ the notion of polarity to recognise the unifiable occurrences of atomic formulas in the theory to be resolved. The resolved pair of literals are then simplified away from the formulae containing them. The resolvent is the disjunction of the two simplified formulae.

In their approaches, the structures of the formulae are clearly broken after a sequence of resolution steps.

Benzmüller *et al.* [6] motivate and also present a mediator between mathematical knowledge bases and theorem provers. A main difference of the approach here is that assertion agents are uniformly modeled; their individual behaviour is only determined by the particular assertion and the proof context they are instantiated with. Further related work is described in [11].

5 Conclusion and Future Work

We argued that the assertion level introduced by Huang [16] is one of the more interesting abstract levels where proof planning should be carried out. Therefore, it is necessary that the proof planner at the assertion level be equipped with an adequate infrastructure to be able to take full advantage of the inferences offered by the assertions. We went on to develop a framework for extracting important information from assertions in accordance to the reasoning context (i.e. the proof situation) in which the assertion is invoked. We describe a distributed modelling for our framework which serves as an assistant for proof search in OMEGA. This mechanism also reflects the cognitive process human agents employ when performing theorem proving: A skilled mathematician would in general accept conclusions emerged from algorithm *AssertionApplication*() as a trivial or obvious step and normally not require a detailed (i.e. logic level) explanation of such steps. The research bears immediate fruit through our application in the DIALOG project aiming at building an intelligent mathematical tutoring system. The realization of natural language dialog in such a project should take full advantage of the assertion level proofs developed by our formalism.

Future work includes: (i) We want to investigate whether our approach scales to higher-order contexts. Additional problems have to be addressed, such as undecidability of higher-order unification, primitive substitution, etc. In higher-order contexts our distributed modeling will gain impact, since this will provide a flexible basis, for instance, to model iterative deepening over constraints such as a maximal unification depth. (ii) Equational reasoning is not yet addressed in our framework and needs to be investigated. In particular adapting our framework to (extensional) equational reasoning within higher-order logic contexts is a challenge. (iii) We propose to fully integrate automated proof planning with our approach to assertion level reasoning.

References

[1] P. Andrews. Transforming matings into natural deduction proofs. In W. Bibel and R. A. Kowalski, editors, *Proc. 5th CADE*, pages 281–292. Springer-Verlag, 1980.

[2] P. B. Andrews. Theorem proving via general matings. *Journal of the ACM*, 28(2):193–214, 1981.

[3] S. Autexier. A proof-planning framework with explicit abstractions based on indexed formulas. *Electronic Notes in Theoretical Computer Science*, 58(2), 2001.

[4] C. Benzmüller *et al.* Tutorial dialogs on mathematical proofs. In *Proceedings of IJCAI-03 Workshop on Knowledge Representation and Automated Reasoning for E-Learning Systems*, 2003.

[5] C. Benzmüller, A. Meier, E. Melis, M. Pollet, and V. Sorge. Proof planning: A fresh start? In *Proceedings of the IJCAR 2001 Workshop: Future Directions in Automated Reasoning*, pages 25–37, Siena, Italy, 2001.

[6] C. Benzmüller, A. Meier, and V. Sorge. Bridging theorem proving and mathematical knowledge retrieval. In *Festschrift in Honour of Jörg Siekmann*, 2002.

[7] C. Benzmüller and V. Sorge. Oants – an open approach at combining interactive and automated theorem proving. In M. Kerber and M. Kohlhase, editors, *Symbolic Computation and Automated Reasoning*, pages 81–97. A.K.Peters, 2000.

[8] W. Bibel. Matings in matrices. *Communications of the ACM*, 26:844–852, 1983.

[9] A. Bundy. The use of explicit plans to guide inductive proofs. In *Proc. CADE*, pages 111–120, 1988.

[10] A. Bundy. A critique of proof planning. In A. C. Kakas and F. Sadri, editors, *Computational Logic. Logic Programming and Beyond*, volume 2408 of *Lecture Notes in Computer Science*. Springer, 2002.

[11] I. Dahn, A. Haida, T. Honigmann, and C. Wernhard. Using mathematica and automated theorem provers to access a mathematical library. In *Proceedings of the CADE-15 Workshop on Integration of Deductive Systems*, 1998.

[12] M. Fitting. Tableau methods of proof for modal logic. *Notre Dame Journal of Formal Logic*, 13:237–247, 1972.

[13] M. Fitting. *First Order Logic and Automated Theorem Proving*. Springer, 1990.

[14] G. Gentzen. Untersuchungen über das logische schliessen. *Mathematische Zeitschrift*, 39(176–210):405–431, 1935.

[15] S. Gerberding and B. Pientka. Structured incremental proof planning. In *KI - Kunstliche Intelligenz*, pages 63–74. Springer-Verlag, 1997.

[16] X. Huang. Reconstructing proofs at the assertion level. In A. Bundy, editor, *Proc. 12th Conference on Automated Deduction*, pages 738–752. Springer-Verlag, 1994.

[17] X. Huang, M. Kerber, J. Richts, and A. Sehn. Planning mathematical proofs with methods. *J. Information Processing and Cybernetics, EIK*, 30(5-6):277–291, 1994.

[18] R. Kowalski. A proof procedure using connection graphs. *J. ACM*, 22(4):572–595, 1975.

[19] E. Melis. Island planning and refinement. Tech. report, Saarland University, 1996.

[20] E. Melis and A. Meier. Proof planning with multiple strategies. In *Proc. CL-2000*, volume 1861, pages 644–653, 2000.

[21] E. Melis and J. Siekmann. Knowledge-based proof planning. *AIJ*, 115(1):65–105, 1999.

[22] N. V. Murray. Completely non-clausal theorem proving. *AIJ*, 18(1):67–85, 1982.

[23] N. V. Murray and E. Rosenthal. Inference with path resolution and semantic graphs. *Journal of the ACM (JACM)*, 34(2):225–254, 1987.

[24] H. J. Ohlbach and J. H. Siekmann. The Markgraf Karl refutation procedure. In J. L. Lassez and G. Plotkin, editors, *Computational Logic, Essays in Honor of Alan Robinson*, pages 41–112. MIT Press, 1991.

[25] J. A. Robinson. A machine-oriented logic based on resolution principle. *Journal of the ACM*, 12(1):23–41, 1965.

[26] K. Schütte. *Proof Theory*. Springer Verlag, 1977.

[27] J. Siekmann *et al.* Proof development with OMEGA. In A. Voronkov, editor, *Proc. 18th CADE*, LNAI 2392, pages 144–149, Copenhagen, Denmark, 2002.

[28] J. Traugott. Nested resolution. In Jörg Siekmann, editor, *Proc. 8th CADE*, number 2392 in LNAI, pages 394–402, Copenhagen, Denmark, 1986. Springer.

[29] L. Wallen. *Automated proof search in non-classical logics: effcient matrix proof methods for modal and intuitionistic logics*. MIT Press series in AI, 1990.

Appendix

We introduce some notations which will be used in the establishments of the results presented in the following.

Model theory

We define the *first-order language* for the syntax of the calculus of interest in the standard way, cf. e.g. [13]. Let $\mathcal{L}$ be a first-order language, a (set-theoretic) structure for $\mathcal{L}$ is a pair $\langle D, \imath \rangle$ where D is a non-empty set, the *domain* of the structure, and $\imath$ an interpretation of $\mathcal{L}$ in D. The satisfaction relation $\models$ between the set of structures and the wffs from $\mathcal{L}$ is also defined as usual.

For a structure M, we define:

$$M \models \varphi^{\ominus} \text{ iff } M \models \varphi$$

and

$$M \models \varphi^{\oplus} \text{ iff } M \not\models \varphi$$

The following is an immediate corollary of the above definitions:

Corollary 1 *Let M be a structure.*

1. *Exactly one of $M \models \varphi^{\ominus}$ or $M \models \varphi^{\oplus}$.*
2. *$M \models \alpha$ iff $M \models \alpha_1$ and $M \models \alpha_2$.*
3. *$M \models \beta$ iff $M \models \beta_1$ or $M \models \beta_2$.*
4. *$M \models (\gamma, x)$ iff for every $c \in D$, $M \models \gamma_0(x)[c/x]$.*
5. *$M \models (\delta, x)$ iff for some $c \in D$, $M \models \delta_0(c)$. In particular, the constant sko chosen by the (δ, x)-rule above is one of those constants c.*

Notations

Let an SFT τ be given and η a node of τ, if the rule that is applied to η is an α (resp. β, γ, δ)-rule then η is said to be *of type* α (resp. β, γ, δ).

We warn the reader that we will systematically abuse our notation:

1. We will refer to an SFT as its root node and vice versa. Hence, we will be able to e.g. talk about the leaf node of a signed formula.
2. We shall also adopt the following practice of using the names of types to denote the formulae they are applied to (possibly with the reference to the formulae). That is:
   - If φ is a signed formula of type α then we also refer to φ as $\alpha(\varphi)$ and its child(ren) as $\alpha_1(\varphi)$ and $\alpha_2(\varphi)$.
   - If φ is a signed formula of type β then we also refer to φ as $\beta(\varphi)$ and its children as $\beta_1(\varphi)$ and $\beta_2(\varphi)$.
   - If φ is a signed formula of type (γ, x) then we also refer to φ as $\gamma(\varphi, x)$ and its child as $\gamma_0(\varphi, x)$.
   - If φ is a signed formula of type (δ, x) then we also refer to φ as $\delta(\varphi, x)$ and its child as $\delta_0(\varphi, sko)$.

Soundness of algorithm GR

Theorem 1 *(Soundness of GR) Let two SFTs τ_1 and τ_2 be given. For any structure M and any substitution θ, if M is a model of $\{\tau_1\theta, \tau_2\theta\}$ then M is a model of $GR(\tau_1, \eta_1, \tau_2, \eta_2, \theta)$, for any pair of leaf nodes η_1 of τ_1 and η_2 of τ_2.*

The theorem is vacuously true in case η_1 and η_2 are not θ-complementary. Thus we only consider the case when they are θ-complementary. We will also assume that the only connectives we have to deal with are negation ($\neg$), conjunction (*wedge*) and disjunction (*vee*) together with the first-order quantifiers ($\forall$ and $\exists$)to simplify the presentation. We first show the following lemma:

Lemma 1 *Let τ be an arbitrary SFT and ς a singleton SFT (i.e. ς itself is the only node of ς). For any structure M and any substitution θ, if M is a model of $\{\tau\theta, \varsigma\theta\}$, M is a model of $GR(\tau, \eta, \varsigma, \varsigma, \theta)$, for any leaf node η of τ.*

Proof: By induction on the structure of τ.

Base case: τ is a singleton SFT with the only node being τ itself. τ and ς are θ-complementary. In other words, the set of models of $\{\tau\theta, \varsigma\theta\}$ is empty. Therefore, the lemma is vacuously true.

Inductive case:

1. Case τ is of type α:

   (a) Case $\alpha_2(\tau) = nil$, i.e. τ has only one child $\alpha_1(\tau)$: Obviously, $form(\tau) = form(\alpha(\tau)) = \neg(form(\alpha_1(\tau)))$ and $polarity(\alpha_1(\tau)) = -polarity(\alpha(\tau))$.

   For any structure M:

   M is a model of $\{\tau\theta, \varsigma\theta\}$
   iff M is a model of $\{\alpha(\tau)\theta, \varsigma\theta\}$
   iff M is a model of $\{\alpha_1(\tau)\theta, \varsigma\theta\}$ (from Corollary 1).

   Then, M is a model of $GR(\alpha_1(\tau), \eta, \varsigma, \varsigma, \theta)$ (inductive hypothesis). Therefore, M is a model of $GR(\tau, \eta, \varsigma, \varsigma, \theta)$.

   The justification for the last step in the above proof is based on the reconstruction of the root node $GR(\tau, \eta, \varsigma, \varsigma, \theta)$ from its only child $GR(\alpha_1(\tau), \eta, \varsigma, \varsigma, \theta)$ and from Corollary 1.

   (b) Case $\alpha_2(\tau) \neq nil$:

   For any structure M:

   M is a model of $\{\tau\theta, \varsigma\theta\}$
   iff M is a model of $\{\alpha(\tau)\theta, \varsigma\theta\}$
   iff M is a model of $\{\alpha_1(\tau)\theta, \varsigma\theta\}$ and M is a model of $\{\alpha_2(\tau)\theta, \varsigma\theta\}$ (from Corollary 1).

   - Case η is a leaf node of $\alpha_1(\tau)$: M is a model of $GR(\alpha_1(\tau), \eta, \varsigma, \varsigma, \theta)$ (inductive hypothesis) and M is a model of $\{\alpha_2(\tau)\theta, \varsigma\theta\}$.
   - Case η is a leaf node of $\alpha_2(\tau)$: M is a model of $\{\alpha_1(\tau)\theta, \varsigma\theta\}$ and M is a model of $GR(\alpha_2(\tau), \eta, \varsigma, \varsigma, \theta)$ (inductive hypothesis).

   Therefore, M is a model of $GR(\tau, \eta, \varsigma, \varsigma, \theta)$ (from the reconstruction of $GR(\alpha(\tau), \eta, \varsigma, \varsigma, \theta)$ from $GR(\alpha_1(\tau), \eta, \varsigma, \varsigma, \theta)$ and $GR(\alpha_2(\tau), \eta, \varsigma, \varsigma, \theta)$).

2. Case τ is of type β:

   For any structure M:

   M is a model of $\{\tau\theta, \varsigma\theta\}$
   iff M is a model of $\{\beta(\tau)\theta, \varsigma\theta\}$
   iff M is a model of $\{\beta_1(\tau)\theta, \varsigma\theta\}$ or M is a model of $\{\beta_2(\tau)\theta, \varsigma\theta\}$ (from Corollary 1).

- Case η is a leaf node of $\beta_1(\tau)$: then, M is a model of $GR(\beta_1(\tau), \eta, \varsigma, \varsigma, \theta)$ (inductive hypothesis). Therefore, M is a model of $GR(\beta(\tau), \eta, \varsigma, \varsigma, \theta)$, from the reconstruction of $\beta(\tau)\theta$ from $GR(\beta_1(\tau), \eta, \varsigma, \varsigma, \theta)$ and $\beta_2(\tau)\theta$;
- Case η is a leaf node of $\beta_2(\tau)$: similar to the preceding case, M is a model of $GR(\beta(\tau), \eta, \varsigma, \varsigma, \theta)$.

3. Case τ is of type (γ, x):

   For any structure M:

   M is a model of $\{\tau\theta, \varsigma\theta\}$
   iff M is a model of $\{\gamma(\tau, x)\theta, \varsigma\theta\}$
   iff M is a model of $\{\gamma_0(\tau, x)\theta, \varsigma\theta\}$ (from Corollary 1).

   - Case $x\theta = x$ or $x\theta = y$ for some free variable y: x (resp. y) is a free variable in $\gamma_0(\tau, x)\theta$ and $form(\gamma(\tau, x)) = \forall x(form(\gamma_0(\tau, x)))$ (resp. $form(\gamma(\tau, x)) = \forall x(form(\gamma_0(\tau, x))[y/x])$). Thus, every model of $GR(\gamma_0(\tau, x), \eta, \varsigma, \varsigma, \theta)$ is also a model of $GR(\gamma(\tau, x), \eta, \varsigma, \varsigma, \theta)$ (which is essentially $GR(\tau, \eta, \varsigma, \varsigma, \theta)$). But from the inductive hypothesis, the structure M is a model of $GR(\gamma_0(\tau, x), \eta, \varsigma, \varsigma, \theta)$.
   - Case $x\theta = c$ for some $c \in D$: straightforwardly, $\gamma(\tau, x)\theta = \gamma_0(\tau, x)\theta$ and from the reconstruction of $GR(\gamma(\tau, x), \eta, \varsigma, \varsigma, \theta)$ from $GR(\gamma_0(\tau, x), \eta, \varsigma, \varsigma, \theta)$, M is a model of $GR(\gamma(\tau, x), \eta, \varsigma, \varsigma, \theta)$ (inductive hypothesis).

4. Case τ is of type (δ, x):

   For any structure M:

   M is a model of $\{\tau\theta, \varsigma\theta\}$
   iff M is a model of $\{\delta(\tau, x)\theta, \varsigma\theta\}$
   iff M is a model of $\{\delta_0(\tau, sko)\theta, \varsigma\theta\}$ (from Corollary 1).

   Then, as $\delta(\tau, x)\theta = \delta_0(\tau, sko)\theta$ and from the reconstruction of $GR(\delta(\tau, x), \eta, \varsigma, \varsigma, \theta)$ from $GR(\delta_0(\tau, sko), \eta, \varsigma, \varsigma, \theta)$, M is a model of $GR(\delta(\tau, x), \eta, \varsigma, \varsigma, \theta)$ (inductive hypothesis).

□

Proof: (of theorem 1)

As with the above lemma, we prove by induction on the structure of τ_1. First, we observe that for an arbitrary SFT τ_2 and any substitution θ, if the structure M is a model of $\tau_2\theta$ then M is also a model of the SFT obtained by (i) instantiating τ_2 according to θ, then (ii) removing a leaf node of τ_2, and then (iii) reconstructing its internal nodes in the way described in algorithm GR. That result is essentially what has been shown in Lemma 1.

Base case: τ_1 is a singleton SFT with the only node being τ_1 itself, i.e. $\tau_1 = \eta_1$. The theorem holds by the above observation and from the fact that M is a model of φ^p if and only if M is a model of $(\neg\varphi)^{-p}$ for any formula φ and any polarity p. Thus it doesn't matter whether η_1 is replaced by $\tau_2\theta$ or $(\neg form(\tau_2\theta))^{polarity(\tau_2\theta)}$, the theorem should always hold.

Inductive case: The proof of the inductive case proceeds in the similar way to that for Lemma 1. I.e. we consider different cases on the type of the root node of τ_1. In particular, the same argument works for all four different types of the root node of τ_1, except for the γ and δ-types where special care must be taken for possible clashes of variables during the reconstruction process. □

When (classical) resolution on a pair of clauses Φ and Ψ is performed, no ordering between Φ and Ψ is necessary. The situation is slightly different for generalized resolution: $GR(\tau_1, \eta_1, \tau_2, \eta_2, \theta)$ and $GR(\tau_2, \eta_2, \tau_1, \eta_1, \theta)$ generally output two syntactically different SFTs. So, it is important that they be still logically equivalent. This is a straightforward corollary of the above theorem:

Corollary 2 *Let two SFTs τ_1 and τ_2 be given. For any structure M and any substitution θ, M is a model of $GR(\tau_1, \eta_1, \tau_2, \eta_2, \theta)$ if and only if M is a model of $GR(\tau_2, \eta_2, \tau_1, \eta_1, \theta)$, for any pair of leaf nodes η_1 of τ_1 and η_2 of τ_2.*

Proof: First, we observe that: If M is a model of $GR(\tau_1, \eta_1, \tau_2, \eta_2, \theta)$ then

- M is a model of $GR(\tau_1, \eta_1, comp(\eta_1), comp(\eta_1), \theta)$; and
- M is a model of $GR(\tau_2, \eta_2, comp(\eta_2), comp(\eta_2), \theta)$.

where, given a signed formula τ, $comp(\tau)$ returns the complementary signed formula of τ. The above observation is immediate from Lemma 1.

Now, to prove that if M is a model of $GR(\tau_1, \eta_1, \tau_2, \eta_2, \theta)$ then M is a model of $GR(\tau_2, \eta_2, \tau_1, \eta_1, \theta)$, we can proceed by induction on the structure of τ_2 with the base case follows from the above observation. Similarly, the inverse direction is proved by induction on the structure of τ_1. □

A further remark on algorithm $GR()$:

REMARK: Line (2) of algorithm $GR()$ is inessential to the generalized resolution framework presented in this paper, i.e. it can be removed without affecting the correctness of algorithm $GR()$. This is a form of the inference rules `Weakening` well known in sequent calculi. This, however, brings out another advantage from the chosen representation: The redundancy that is apparent in the generalized resolution framework without `Weakening` can be easily eliminated by this line which effectively prunes all the subtrees that are α-related to the resulted resolvent, i.e. the redundant ones.

Resolving a pair of complementary non-atomic formulae

Observe that the framework presented so far in this paper could have been described for this general setting without any modification on the algorithm or the theorems. That will come with a price however. A large number of internal nodes will potentially explode the search space. Moreover, unification on internal nodes is more complex than unification on the leaf nodes only. Nevertheless, we will show that the framework we have proposed is sufficient to achieve such general steps of resolution, even though it could require more than one step of resolution on the leaf nodes.

Theorem 2 *Let SFTs τ_1 and τ_2 be given. If η_1 and η_2 are two nodes of τ_1 and τ_2, respectively, and θ is a substitution such that η_1 and η_2 are θ-complementary then the resolution of τ_1 and τ_2 over η_1 and η_2 can be achieved by repeatedly applying $GR()$ on τ_1 and τ_2 and their resulting resolvents a finite number of times.*

Proof: By induction on the structure of η_1:

Base case: Trivial as η_1 and η_2 are the leaf nodes of τ_1 and τ_2.

Inductive case:

1. Case η_1 is of type α:
   (a) Case $\alpha_2(\eta_1) = nil$: η_1 is of the form $(\neg\varphi)^p$ for some polarity p. As η_1 and η_2 are θ-complementary, η_2 must be of the form $(\neg\psi)^{-p}$ so that $\varphi\theta = \psi\theta$. Hence, $\alpha_1(\eta_1) = (\varphi)^{-p}$ and $\alpha_1(\eta_2) = (\psi)^p$. Obviously, $\alpha_1(\eta_1)$ and $\alpha_1(\eta_2)$ are also θ-complementary. From the inductive hypothesis, we can resolve them after repeatedly applying $GR()$ on τ_1 and τ_2 and their resulting resolvents k times. Therefore, the resolution of τ_1 and τ_2 over η_1 and η_2 can also be achieved after k applications of $GR()$ on τ_1 and τ_2 and their resulting resolvents.
   (b) Case $\alpha_2(\eta_1) \neq nil$:
   - Case $polarity(\eta_1) = \ominus$: η_1 is of the form $(\varphi_1 \wedge \varphi_2)^\ominus$. Then η_2 must be of the form $(\psi_1 \wedge \psi_2)^\oplus$ so that $\varphi_1\theta = \psi_1\theta$ and $\varphi_2\theta = \psi_2\theta$. Therefore, from the inductive hypothesis, $\alpha_1(\eta_1)$ and $\alpha_1(\eta_2)$ are resolved after k applications of $GR()$ on τ_1 and τ_2 and their resulting resolvents; and $\alpha_2(\eta_1)$ and $\alpha_2(\eta_2)$ are resolved after l applications of $GR()$ on τ_1 and τ_2 and their resulting resolvents. As a consequence, η_1 and η_2 can be resolved after $k + l$ applications of $GR()$ on τ_1 and τ_2 and their resulting resolvents.
   - Case $polarity(\eta_1) = \oplus$ is also similar.

2. Case η_1 is of type β: similar to the case η_1 is of type α and $\alpha_2(\eta_1) \neq nil$.

3. Case η_1 is of type (γ, x): η_1 is of the form $(\mathsf{Q}_x\Phi(x))^p$ where $\mathsf{Q} \in \{\forall, \exists\}$. Then η_2 must be of the form $(\mathsf{Q}_y\Psi(y))^{-p}$ so that $\Phi(x)(\theta \circ \{x \mapsto y\}) = \Psi(y)\theta$. Therefore, from the inductive hypothesis, $\gamma_0(\eta_1, x)$ and $\delta_0(\eta_2, sko)$ are resolved (under the substitution $\theta' = \theta \circ \{x \mapsto sko\}$) after k applications of $GR()$ on τ_1 and τ_2 and their resulting resolvents. As a consequence, η_1 and η_2 can be resolved after k applications of $GR()$ on τ_1 and τ_2 and their resulting resolvents.

4. Case η_1 is of type (δ, x): is symmetric to the preceding case, i.e. η_1 is replaced by η_2 and vice versa in the above argument.

□