Automated Deduction in Frege-Hilbert Calculi

ELMAR EDER
University of Salzburg
Department of Computer Sciences
Jakob-Haringer-Str. 2
AUSTRIA
eder@cosy.sbg.ac.at

Abstract: In this paper the appropriateness of Frege-Hilbert calculi and other cut calculi for automated deduction is discussed. Unification of formula schemes and composition of rules of Frege-Hilbert calculi lead to proof systems which may be suited for automation.

Key–Words: Automated deduction, Frege-Hilbert calculi, Composition of rules, Cut rule

1 Introduction

In this paper we give a brief account of some results on automated deduction in Frege-Hilbert calculi using the operation of composition of rules. We restrict ourselves to classical first order predicate logic with or without function symbols. Whereas there are many successful systems for automated deduction in resolution and in systems related to backward reasoning in cut-free sequent calculi (connection method, tableau calculus), automated reasoning in Frege-Hilbert calculi or generally in proof systems with the cut rule is much more difficult. For a long time it seemed to be entirely impossible because in each proof step rules can be applied in an infinite number of possible ways making automatic proof search a nightmare. On the other hand, resolution and cut-free calculi are hopelessly inefficient even in terms of shortest possible proofs (not even regarding the problem of finding the proof) for some classes formulas. Also, cut is a natural tool used by humans when reasoning and so should be used in automated reasoning too.

In conventional automated deduction, a proof is constructed by constructing the formulas of a proof tree one by one. This cannot be done efficiently in calculi with cut. However, it is possible to construct formula schemes and compositions of rules of the calculus and later instantiate them to obtain the final proof. So the proof procedure proposed in this paper has as its inference rule the operation of composition of rules of the Frege-Hilbert calculus. In this way it is possible to obtain a proof, for example in a Frege-Hilbert calculus, without having to choose from an infinite number of possibilities in a single proof step.

2 Calculi With Cut and Calculi Without Cut

The first formulation of a language of full first order predicate logic given in the history of logic was Gottlob Frege's Begriffsschrift [5] (concept language) in 1879. In his Begriffsschrift, Frege also provided a proof calculus which was later proved to be sound and complete when restricted to what we now call first order predicate logic. Frege's idea was to provide a set of *axioms* which are formulas known to be valid, and a set of *rules* to derive new valid formulas from formulas which have already been derived. Similar calculi were introduced and investigated to a great extent by David Hilbert and others.

Here is a typical Frege-Hilbert calculus with four axiom schemes

$$A \to B \to A \tag{1}$$

$$(A \to B \to C) \to (A \to B) \to A \to C \tag{2}$$

$$(\neg A \to \neg B) \to B \to A \tag{3}$$

$$\forall x F \to F^x_t \tag{4}$$

and two rules

$$\frac{A \qquad A \to B}{B} \tag{5}$$

$$\frac{A \to F^x_p}{A \to \forall x F} \tag{6}$$

where "F^x_t" denotes the result of replacing every free occurrence of the variable x in the formula F with the ground term t. The symbol p in the last rule is called a *parameter*. It is subject to the constraint that

p must not occur in the conclusion $A \to \forall x F$. Rule (5) is called *modus ponens*. An axiom scheme can be considered as a rule with no premise.

Even nowadays most proof calculi used by logicians for classical as well as non-classical logic are of the Frege-Hilbert type. The reason for this is that Frege-Hilbert calculi have a simpler syntactic structure than some of the calculi introduced later and yet are very powerfull for expressing proofs of logical formulas.

On the other hand, Frege-Hilbert calculi are difficult to use for automated deduction. If you want to prove some given formula, it would be very inefficient to construct the proof in a forward way, randomly producing axioms and trying to use the rules to derive conclusions from them until you get the formula you wanted to prove. Rather, the more obvious way would be to use backward reasoning. You start from the formula you want to prove and apply the rules backward until all your premises are axioms. The problem with this idea is that a backward application of modus ponens is not unique. If you want to prove a formula B with modus ponens, you have to guess the formula A, and there are an infinite number of possibilities what A could be. An automated system would be completely lost. This problem appears in a Frege-Hilbert calculus whenever there is a rule with a formula A occurring in at least one of the premises but not occurring in the conclusion. Such a rule is called a *cut rule* and the formula A is called a *cut formula*. Unfortunately, there is no sound and complete Frege-Hilbert calculus which does not have a cut rule.

In the history of logic, cut rules caused a headache to logicians for similar reasons when they wanted to prove the consistency of logical calculi long before the advent of modern computers and thus long before the birth of automated deduction. In 1935 Gerhard Gentzen introduced his *sequent calculus* [6] in a version with cut rule and a version without cut rule. The sequent calculus is similar to Frege-Hilbert calculi but uses the data structure of a sequent (of formulas) instead of formulas. So you prove a sequent of formulas rather than proving a formula. Gentzen showed that in his calculus applications of the cut rule can be eliminated. A proof with cut can be transformed to a proof without cut. The cut-free version of Gentzen's sequent calculus can be used for *analytic* backward reasoning. This means that the backward reasoning (backward application of rules) amounts to decomposing (analyzing) the given formula rather than guessing a cut formula.

There are a number of calculi and proof systems derived from Gentzen's sequent calculus. One of them is Wolfgang Bibel's *connection method* [3]. Another one is Beth's and Smullyan's *tableau calculus* [1, 2, 9]. Both of them can be viewed as refinements of the sequent calculus with the use of structure sharing techniques. All these calculi are characterized by a rather directed proof search.

The calculus most often used for automated deduction is the *resolution calculus* [8] introduced in 1965 by J. A. Robinson. It works by proving the unsatisfiability of a formula in conjunctive normal form. It is characterized by a particularly simple deduction rule, the *resolution rule*, and works by deriving new clauses from a given set of clauses until an empty clause is derived, indicating a contradiction. In resolution it is more difficult to get a well directed proof search than in calculi based on the sequent calculus.

Unfortunately, cut elimination may be extremely expensive, as R. Statman [10] and V. P. Orevkov [7] have shown. More precisely, there is a sequence (F_n) of formulas and a polynomial p such that each F_n has a proof of length $\leq p(n)$ in the full sequent calculus, but the shortest proof of F_n in the cut-free sequent calculus has length $\geq \underbrace{2^{2^{\cdot^{\cdot^{\cdot^{2^2}}}}}}_{n \text{ times}}$. Moreover, these formulas are not just some academic examples but formulas from combinatory logic which may very well occur in the everyday life of a logician or of a computer scientist. As a consequence, for these formulas even a stupid generation of all possible finite sequences of formulas and testing whether they are proofs in the full sequent calculus would be more efficient than even the most clever proof search in the cut-free sequent calculus.

This result of Statman and Orevkov carries over to all calculi based on the cut-free sequent calculus including the connection method and the tableau calculus, and also to some other calculi such as the resolution calculus. Some of these calculi can be extended to allow something similar to a cut rule. For example, the tableau calculus can be extended by allowing a case distinction between A and $\neg A$ where A is some arbitrary formula. This amounts to the same thing as the cut rule in the sequent calculus. However, again this destroys the analytic character of the calculus. An automated deduction system would have the same problem guessing the formula A as in the full sequent calculus. Even for a human mathematician it is often hard to decide how to make a case distinction, i.e., which formula A to choose. However, as every mathematician knows, case distinction is a very powerful tool in mathematical theorem proving. Therefore we should try hard to find ways to also employ case distinction or a cut rule in systems of automated deduction.

3 Construction of a Proof

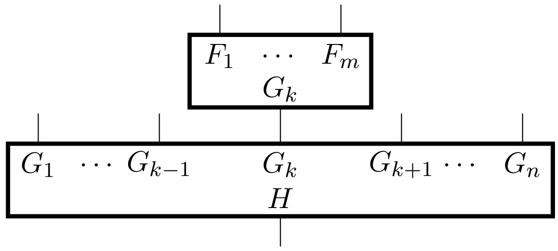
Since Frege-Hilbert calculi are syntactically comparatively simple, it seems that for a theoretical investigation of possible proof systems with cut the Frege-Hilbert calculi would be a good start. Of course, for a later practical system the full sequent calculus and systems based on it seem to be more adequate, since the sequence calculus is complete also without cut and therefore a well dosed sparse application of cuts is possible there.

So let us consider the Frege-Hilbert calculus given at the beginning of section 2 again. One idea how to deal with a cut rule such as modus ponens is not to guess the cut formula A but to leave the symbol "A" there instead and to continue applying rules backwards. Assume, for example, that we want to prove the formula $P(a)$. If we try to do this with the modus ponens, we get the premises A and $A \to P(a)$. At this point we do not know yet which formula the symbol "A" is going to stand for. The decision which formula "A" is to stand for is postponed until later. The same technique is used by automated deduction calculi on the level of terms rather than formulas. There a term is determined by *unification* (or equation solving). For our purpose we need unification of formulas in addition to unification of terms.

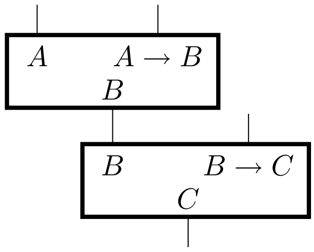
The symbol "A" can be viewed as a meta-symbol standing for a formula. Similarly we have meta-symbols for terms, variables, and parameters. Since we have meta-symbols such as "A" during our deduction process, we do not only deal with formulas. Rather we deal with *formula schemes*. A formula scheme is the result of replacing within a formula some subformulas or terms with adequate meta-symbols.

Now, backward reasoning in a Frege-Hilbert system (with cut) can be done as follows. We start with the formula to be proved. Then we apply rules of the calculus in a backward direction, in each step replacing the conclusion of the rule application with the premises. In each step, a formula scheme (the conclusion of the rule application) is replaced with zero or more new formula schemes (the premises of the rule application). Since axiom schemes are rules with zero premises, our proof is finished when no formula schemes are left.

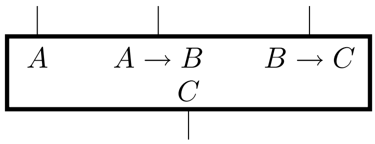
Actually, the premises and the conclusion of a rule of a Frege-Hilbert calculus are formula schemes. It turned out [4] that two rules R_1 and R_2 of a Frege-Hilbert calculus can be composed to give a new rule R such that H is the conclusion of an instance of R with premises $G_1, \ldots, G_{k-1}, F_1, \ldots, F_m, G_{k+1}, \ldots, G_n$ if and only if there is a formula G_k such that G_k is the conclusion of an instance of R_1 with premises $F_1, \ldots, F_m$ and H is the conclusion of an instance of R_2 with premises $G_1, \ldots, G_n$. Composition of rules can be depicted in a way similar to composition of components of an electric circuit:



For example, a composition of the modus ponens rule with itself can be depicted as



which results in the rule



or

$$\frac{A \qquad A \to B \qquad B \to C}{C}$$

for short.

The idea in computing the composition of two rules is to use a kind of unification with meta-symbols. However, formula schemes and unification alone do not suffice. Rather, some constraints have to be added.

So we can go a step further and use the operation of composition of rules in order to construct a proof tree. The proof tree can then not only be constructed from its root. Rather we can start constructing parts of the tree in various places somewhere in the middle of the tree and later join them, again using the operation of composition of rules.

4 Composition of Rules

As a simple example to see how the composition of rules works in first order predicate logic, let us look at a proof of the formula $\forall x P(x) \to \forall y P(y)$ in our Frege-Hilbert calculus:

$$\forall x P(x) \to P(p)$$
$$\forall x P(x) \to P(y).$$

In the first step we obtain $\forall x P(x) \to P(p)$ as an instance of the axiom scheme (4). From this we derive $\forall x P(x) \to P(y)$ by rule (6). Now let us compute the composition of rules (4) and (6). First we observe that (4) and (6) share the meta-symbols "F" and "x". Therefore we standardize the two rules apart by renaming in (6) the meta-symbol "F" to "G" and the meta-symbol "x" to "y" and we get

$$\frac{A \to G^y_p}{A \to \forall y G}$$

Now, if an application of rule (4) is to be followed by an application of this rule then $A = \forall x F$ and $F^x_t = G^y_p$ must hold. Also, the parameter p must not occur in $A \to \forall y G$ and thus it must not occur in F and not in G. We write $p \notin FG$ for short. In summary, the composition of the rules (4) and (6) is the axiom scheme (i.e. nullary rule)

$$\forall x F \to \forall y G$$

with the constraints $F^x_t = G^y_p$ and $p \notin FG$.

We can depict this process of composition of rules as follows.

$$\boxed{\forall x F \to F^x_t}$$

$$\boxed{\begin{array}{c} \forall x F \to G^y_p \\ \forall x F \to \forall y G \end{array}}$$

with the constraints $F^x_t = G^y_p$ and $p \notin FG$. The constraint $F^x_t = G^y_p$ guarantees that the conclusion $\forall x F \to F^x_t$ of the application of rule (4) equals the premise $\forall x F \to G^y_p$ of the application of rule (6). The resulting rule (axiom scheme) is depicted as

$$\boxed{\forall x F \to \forall y G}$$

with the same constraints.

If we set $F = P(x)$ and $G = P(f(y))$ and $t = f(p)$ then we see that this composition of rules also yields a proof of the formula

$$\forall x P(x) \to \forall y P(f(y)).$$

The proof is

$$\begin{array}{c} \forall x P(x) \to P(f(p)) \\ \forall x P(x) \to \forall y P(f(y)). \end{array}$$

The most general form of a rule obtained by repeated compositions of rules of Frege-Hilbert calculi is

$$\frac{\Phi_1 \quad \dots \quad \Phi_n}{\Psi}$$

together with a sequence $\mathcal{C}$ of constraints where $\Phi_1, \dots, \Phi_n$ and Ψ are formula schemes. Symbols such as "F^x_t" and "F^x_p" denoting the result of replacing a variable with a term can be pushed into the constraints. For example, our Frege-Hilbert calculus can then be presented as follows.

Axioms:

$$A \to B \to A \tag{7}$$

$$(A \to B \to C) \to (A \to B) \to A \to C \tag{8}$$

$$(\neg A \to \neg B) \to B \to A \tag{9}$$

$$\forall x F \to E \qquad \text{if C10} \tag{10}$$

In axiom scheme (10) there is a constraint

$$\text{C10:} \qquad E = F^x_t.$$

Rules:

$$\frac{A \qquad A \to B}{B} \tag{11}$$

$$\frac{A \to E}{A \to \forall x F} \qquad \text{if C12} \tag{12}$$

In rule (12) there is a constraint

$$\text{C12:} \qquad E = F^x_p, \quad p \notin AF.$$

Now the formula schemes occurring in the rules without their constraints can be regarded as terms of a free term algebra with the logical connectives and quantifiers as function symbols and the meta-symbols as variables. Therefore the composition of two rules can be computed with the usual technique of first standardizing apart and then unifying the conclusion of the first rule with one of the premises of the second rule. Finally, the sets of constraints have to be merged.

If we look at the constraints C10 and C12 we see that they do not contain any propositional connectives or quantifiers. The formula schemes occurring in constraints of rules have the form $\Gamma^{\xi \dots \zeta}_{\mu \dots \nu}$ where Γ is a meta-symbol for a formula, $\xi, \dots, \zeta$ are meta-symbols for variables, and $\mu, \dots, \nu$ are meta-symbols for terms or parameters. We call such a formula scheme an *atomic formula scheme*. Constraints have always the form $\Phi = \Psi$ or $\alpha \notin \Phi$ where Φ and Ψ are atomic formula schemes and α is a meta-symbol for a parameter.

Note that any set of such constraints has a solution. For example, replacing all meta-symbols for formulas with one and the same proposition symbol P solves all possible constraints. Let us call a composition of rules *failed* if it has no instance. Then it follows that a failure of a composition of rules is always accompanied by a failure of the unification process, never by a failure of the constraints. It can therefore easily be recognized.

5 Conclusion

Automated deduction in proof calculi with a cut rule can be used to prove formulas which have extremely long shortest proofs in cut-free calculi and in resolution if suitable search strategies for proof search can be found. The results stated in this paper show at least that the problem of infinitely branching search trees can be overcome by using unification on the formula level to compute compositions of rules. A student at our department is currently implementing an interactive and automatic deduction system based on composition of rules in Frege-Hilbert calculi as his master's thesis. This deduction system will be used for further investigation of rules and the operation of composition as well as for practical testing.

References:

[1] Evert W. Beth. Semantic entailment and formal derivability. *Mededlingen der Koninklijke Nederlandse Akademie van Wetenschappen*, 18(13):309–342, 1955.

[2] Evert W. Beth. *The Foundations of Mathematics*. North-Holland, Amsterdam, 1959.

[3] Wolfgang Bibel. *Automated Theorem Proving*. Artificial Intelligence. Vieweg, Braunschweig/Wiesbaden, second edition, 1987.

[4] Elmar Eder. Backward reasoning in systems with cut. In Jaques Calmet, John A. Campbell, and Jochen Pfalzgraf, editors, *Artificial Intelligence and Symbolic Computation, International Conference, AISMC-3*, volume 1138 of *Lecture Notes in Computer Science*, pages 339–353. Springer, September 1996.

[5] Gottlob Frege. Begriffsschrift, eine der arithmetischen nachgebildete Formelsprache des reinen Denkens. Halle, 1879.

[6] Gerhard Gentzen. Untersuchungen über das logische Schließen. *Mathematische Zeitschrift*, 39:176–210, 405–431, 1935.

[7] V. P. Orevkov. Lower Bounds for Increasing Complexity of Derivations after Cut Elimination. *Zapiski Nauchnykh Seminarov Leningradskogo Otdeleniya Matematicheskogo Instituta im V. A. Steklova AN SSSR*, 88:137–161, 1979. English translation in *J. Soviet Mathematics*, 2337–2350, 1982.

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

[9] Raymond M. Smullyan. *First-Order Logic*. Ergebnisse der Mathematik und ihrer Grenzgebiete. Springer-Verlag, Berlin, Heidelberg, New York, 1971.

[10] R. Statman. Lower bounds on Herbrand's theorem. *Proc. AMS*, 75, 1979.