Presenting Intuitive Deductions via Symmetric Simplification

Frank Pfenning
School of Computer Science
fp@cs.cmu.edu

Dan Nesmith
Department of Mathematics
nesmith@cs.cmu.edu

Carnegie Mellon University
Pittsburgh, PA 15213, USA

Abstract

In automated deduction systems that are intended for human use, the presentation of a proof is no less important than its discovery. For most of today's automated theorem proving systems, this requires a non-trivial translation procedure to extract human-oriented deductions from machine-oriented proofs. Previously known translation procedures, though complete, tend to produce unintuitive deductions. One of the major flaws in such procedures is that too often the rule of indirect proof is used where the introduction of a lemma would result in a shorter and more intuitive proof.

We present an algorithm, *symmetric simplification*, for discovering useful lemmas in deductions of theorems in first- and higher-order logic. This algorithm, which has been implemented in the TPS system, has the feature that resulting deductions may no longer have the weak subformula property. It is currently limited, however, in that it only generates lemmas of the form $C \vee \neg C'$, where C and C' have the same negation normal form.

1 Introduction

In this paper we deal with the problem of *proof presentation*, a problem that is often overlooked in automated reasoning: many of today's automatic systems focus exclusively on the search for the proof of a theorem. A presentation of the proof that is discovered is often no more than a literal tracing of the search process. Such a proof presentation serves more as a verification of the automatic procedure than as an intelligible argument for the theorem's veracity. For some applications (such as program verification) the answer `Yes!` may be all that is desired, but in many other applications we would like more information. For example, in decision support systems such as (E)MYCIN [25, 27] the inability to provide a convincing argument that the machine has correctly diagnosed a problem, deduced a consequence, or constructed a plan renders the program much less

This work was supported in part by NSF grant CCR-8702699 and in part by the Defense Advanced Research Projects Agency (DOD), ARPA Order No. 5404, monitored by the Office of Naval Research under contract N00014-84-K-0415.

trustworthy and thus much less effective. In systems designed to support the teaching of logic a flexible facility for giving advice should know how to construct *good* proofs in the deductive system underlying the course. In systems where proofs are the data to be manipulated to fit other tasks such as explanation-based generalization [10] or the extraction of programs [5], the structure of proofs becomes of primary importance which, unfortunately, in practice prohibits the use of state-of-the-art theorem proving technology in such applications. Further evidence that proof presentation is a hard and important problem is that mathematicians spend a large percentage of their time analyzing and reformulating proofs.

In the literature one finds two basic approaches to the proof presentation problem. One is to conduct the search for a proof in such a way that, when it is found, it can be displayed intelligibly in a very straightforward manner. This requires the theorem prover to be based on natural deduction [7, 6, 9], or at least to be very close to the natural deduction format. The disadvantage of this approach is that, to date, other theorem proving procedures are superior in that they can prove more theorems, and can prove them faster. Moreover, even natural deduction proofs are not always "natural" and many would profit tremendously from some transformations (Example 5 provides a case in point). The second approach is to decouple the search process from the proof presentation process, in which case proof presentation is reduced to translation from one proof format, say, resolution, to another, typically natural deduction. This approach has been taken by Lingenfelder [16, 17] for the connection graph method, Andrews [1] for mating refutations, and Miller [18, 19], Felty [11], and Pfenning [21, 22] for the closely related expansion proofs.

There are many subproblems one faces when trying to present a proof. What deductive system should be used? How much detail should be presented? How do we measure how "good" a presentation is? How do we interact with the user when presenting a deduction? In this paper, we assume that we have chosen the natural deduction formalism as a target for translation, since it appears to be intuitive to (most) humans and can also be used as a basis for further transformation, for example into natural language (see Chester [8] and Huang [14]) or into other, related deductive systems. We will ignore questions of detail of presentation and user interaction and concentrate on improving "structural" qualities of natural deductions.

In our experience with ETPS [23] (used as a teaching tool in logic classes) and TPS [3, 2] (used as a research tool in theorem proving for higher-order logic) and in the literature, it seems that the problem of translating *analytic proofs* (*normal* proofs in natural deduction terminology, *cut-free* proofs in sequent calculi) between different deductive systems is relatively well understood for a number of systems, in particular for expansion proofs and resolution refutations. The crucial property characterizing the deductions representing the proofs is that they have the *weak subformula property*: only subformulas of the theorem and instances thereof (possibly with an added negation, hence "weak") will appear in the final deduction. Though analytic deductions are often "right," especially for many small problems, they are also often "wrong" in the sense that a non-analytic deduction with a judicious use of a lemma might have drastically reduced the size and complexity of the deduction.[1] A word on terminology: we will use "lemma" to refer to a non-subformula in a non-analytic deduction.

Thus we believe that in order to make further progress, we must study ways to introduce lemmas into the presentation of a proof. In this paper, we present just one method for introducing lemmas into deductions, but one which we have found to be of great practical value. Even though our deductions do *not* have the weak subformula property, the lemmas are not very far-fetched, and we believe that this is but a small step towards the overall goal of presenting intuitive deductions. Other, complementary approaches come to mind, such as applying methods from the field of inductive inference [4] to achieve a more global restructuring of deductions.

As our deductive systems we have chosen expansion proofs [2, 18, 19, 20, 21, 22] (on the analytic side) to be translated into natural deductions. Expansion proofs simplify a number of translation issues over, say, resolution. First and foremost, the structure of the original formula is preserved rather than translated into clausal form. Secondly, certain properties that our algorithm is required to check can be read off fairly directly. Thirdly, TPS [3] is a theorem prover for first- and higher-order logic which generates expansion proofs and thus allows for immediate experimentation with many examples in its library of theorems. We emphasize, however, that, while details certainly would differ widely, we believe that the basic idea of our translation procedure can be adapted to apply to the problem of translating other forms of automatically generated proofs into natural deduction. Moreover, other forms of analytic proofs such as resolution refutations, can be translated into expansion proof format, though the value of such translations is unclear.

2 A Summary of the Method

We have been using a succession of translation algorithms from expansion proofs into natural deductions in the TPS and ETPS systems for several years. ETPS is used in logic classes in first-order and higher-order logic, TPS is a theorem proving system for first-order and higher-order logic. The scale of the examples TPS typically deals with is relatively small (5–50 line proofs), though the proofs, especially in higher-order logic, are sometimes quite intricate (such as proofs of Cantor's theorem). This gives us the luxury of neglecting certain aspects of global proof restructuring, such as determining whether a certain subproof could be done more directly with the use of some lemma stored in a database. Level of detail of the proof can be controlled by the user to some extent, though more work is certainly required on this aspect of the translation.

From our experience with the system it became obvious that the least intelligible deductions arose from the use of the rule of indirect proof. But how can the application of this rule be avoided? There are some answers to this question in the literature, since it is exactly the rule of indirect proof which separates classical logic from intuitionistic logic. But very general methods[2] do not seem reasonable for our application. It is also well-known that instead of a rule of indirect proof, we can include the axiom schema $A \vee \neg A$

[1] A theoretical analysis of this phenomenon with precise bounds is given by Statman [26].

[2] For a discussion and further references see Kreisel & MacIntyre [15]

of excluded middle in the deductive system. Many of our examples can be proven very intuitively with a judicious use of the law of excluded middle, but the “right” formula is not always a subformula of the theorem.

This basic observation led us to ask when and how we should use the law of excluded middle in the construction of natural deductions from expansion proofs. We believe we have found a good answer and it generates intuitive deductions in all of our motivating examples and others.

The first step is to analyze when and why the translation procedure would have to use a classical rule. A little analysis shows that there are exactly three types of situations in which we need a classical rule of inference, and where the previous translation algorithm would, as a last resort, fall back on the rule of indirect proof. Imagine that we have some assumptions $\mathcal{S}$ and a conclusion C and our goal is to derive C from $\mathcal{S}$. There are two types of steps one can apply: one is to work forward from the assumptions; the other is to work backward from the conclusion. Informally, the cases in which we will have to apply a classical rule of inference are

1. $C = A \vee B$ and neither A nor B alone follow from the assumptions, and no progress can be made by applying an intuitionistic rule to an assumption;
2. $C = \exists x\, A$ and there is no single term t such that $[t/x]A$ follows from the assumptions, nor can progress be made by applying an intuitionistic rule to an assumption;
3. C is atomic and no progress can be made by applying an intuitionistic rule to an assumption. In this case, for any assumption of the form $A \rightarrow B$ or $\neg A$, the formula A does not follow from the remaining assumptions. Moreover, there must be at least one such assumption.

The examples in Section 4 illustrate these impasses. We will see that the second type of impasse actually encompasses two quite different situations.

Section 5 deals with the details of how we can sometimes employ the law of excluded middle to our advantage in these situations. Let us illustrate here the general idea of the procedure in the simplest of these cases where the conclusion is a disjunction. Since this case is the simplest and our example is propositional, it may appear that the problem is in general trivial, and that easier methods than symmetric simplification should apply. Though we cannot rule out such a possibility, we have arrived at symmetric simplification only after considering and discarding many more specialized and apparently simpler methods. The list of “difficult” (in the sense that it is difficult to find an intuitive deduction) theorems at the end of this paper give a range for the kinds of problems which may arise.

As an example, assume we are trying to prove $[A \wedge B] \vee [A \wedge \neg B]$ from the assumption A. Clearly neither disjunct follows from A and the assumption A does not lend itself to any useful forward reasoning, and thus we are at a disjunctive impasse.

What is an intuitive way of proving this theorem? Proof by contradiction is certainly possible, but clearly not very intuitive. The following argument seems natural: (1) either

B is true or false. (2) If B is true, we can conclude $A \wedge B$. (3) If B is false, we can conclude $A \wedge \neg B$. (4) So the disjunction holds in either case.

In outline, here is how our translation algorithm arrives at the lemma (1). First we observe that we can always pick either disjunct when trying to break a disjunctive impasse and continue by distinguishing two cases: the disjunct might be true or it might be false.

If we picked the left disjunct, this would construct the lemma $[A \wedge B] \vee \neg[A \wedge B]$ which we add to our assumptions and now proceed by cases and two $\vee$-introductions, yielding the obligations (1) to prove $A \wedge B$ from A and $A \wedge B$ in the first case (which is immediate), and (2) to prove $A \wedge \neg B$ from the assumptions A and $\neg[A \wedge B]$. Even though this is clearly possible, this is also circuitous and not the desired proof.

The general idea underlying our translation procedure is to examine the subproof obligations arising from this first attempt at a useful lemma, namely

1. to prove $A \wedge B$ from A and $C = A \wedge B$ in the first case, and
2. to prove $A \wedge \neg B$ from the assumptions A and $\neg C = \neg[A \wedge B]$.

We have highlighted C and $\neg C$, since these two formed our lemma (application of excluded middle). Note that we have the freedom to change and particularly *simplify* C as long as both subproof obligations can still be fulfilled. This is a balancing act, since strengthening C will weaken $\neg C$ and vice versa. Here one can easily see that after erasing the left conjunct A from both C and $\neg C$ both remaining proof obligations can still be fulfilled, and also that the proofs are much simpler now. In fact, we have arrived at the desired intuitive deduction shown before.

The essence of symmetric simplification[3] is to simplify C (and consequently $\neg C$) such that both subgoals remain provable. This implies simultaneous changes to the proofs that C implies one conclusion and $\neg C$ implies the other conclusion, where C may be varied. Our procedure does not require a general theorem prover (and it is hard to see how to take advantage of one) but uses the original expansion proof as a strong guide.

To show the limitations of this method, let us reconsider the goal of deriving $[A \wedge B] \vee [A \wedge \neg B]$ from A. This time we proceed with an application of the distributivity of conjunction over disjunction to change the goal to $A \wedge [B \vee \neg B]$ and then proceed in the obvious fashion. This is a deduction beyond our current method, and we know of no other general method which could produce this deduction. However, we believe that the general problem of using previously proved lemmas both in theorem proving and in proof presentation and the problem of finding useful "local" lemmas in a deduction are orthogonal—it is only the latter we are addressing here.

[3] "Symmetric" since both C and $\neg C$ remain identical. One could imagine "asymmetric simplification" which simplifies the lemma $C \vee \neg C$ to $C' \vee C''$ or even $C' \rightarrow C''$ as long the lemma remains provable. Currently we have no algorithm or heuristics for the asymmetric simplification problem.

3 Natural Deductions

The formulation of natural deduction system we use is basically Gentzen's NK [12], but our implementation uses a variant with higher-order rules (as in Prawitz [24]) and rules for equality. The additional complexity introduced into the proof presentation process does not invalidate the analysis made here, but requires some separate considerations (see [22, Chapters 4–6]).

The logical language contains the propositional constants and connectives $\wedge$, $\vee$, $\neg$, $\rightarrow$, and $\bot$ along with the quantifiers $\forall$ and $\exists$.

The inference rules of the system include the usual introduction and elimination rules for each connective and quantifier. There are also two rules involving $\bot$. The first is $\bot_I$, the intuitionistic absurdity rule (from falsehood we can conclude anything), the second is $\bot_C$, or classical proof by contradiction. Below we also show the rule of proof by cases (the $\vee$-elimination rule $\vee$E), since proof by cases is one of the more frequent inferences generated by our proof presentation procedure which uses symmetric simplification.

$$\frac{\bot}{A}\,\bot_I \qquad\qquad \begin{array}{c}[\![\neg A]\!]^1\\ \vdots \\ \dfrac{\bot}{A}\,\bot_C{}^1\end{array} \qquad\qquad \dfrac{A\vee B \quad \begin{array}{c}[\![A]\!]^1\\ \vdots\\ C\end{array} \quad \begin{array}{c}[\![B]\!]^1\\ \vdots \\ C\end{array}}{C}\,\vee\mathrm{E}^1$$

As shown above in $\bot_C$ and $\vee$E, cancelled assumptions are written as $[\![A]\!]$. A numbered assumption is cancelled in the inference with the same superscript. An inference may cancel 0 or more assumptions with the same formula, so the inference remains correct if the assumption does not occur in the deduction.

4 Translation from Expansion Proofs to Natural Deduction

Our general approach to the proof presentation problem is goal-oriented: at any stage during the translation from an analytic proof[4] we have a number of assumptions $\mathcal{S}$ (a list of formulas, though their order is irrelevant) and a conclusion A. The goal is to fill in the gap in the deduction which shows that A follows from $\mathcal{S}$, given an expansion proof which shows that the gap can indeed be filled. The implementation of the translation is centered around tactics [13] which may consult the expansion proof to check if certain conditions are satisfied. In return for this benefit of expansion proofs, the tactics also have the obligation to construct expansion proofs for any subgoals they may produce.

A complete set of tactics for the translation process is described in [22, Chapter 6]. The tactics decompose into five different categories: *bookkeeping tactics* (which do not apply any inference rules), *minimal conclusion tactics* (which suggest introduction rules to be applied to the conclusion), *minimal assumption tactics* (which suggest elimination rules

[4]Henceforth we will say *expansion proof*, since this is what our implementation actually uses.

to be applied to assumptions), *non-minimal tactics* (which suggest use of the absurdity rule or indirect proof), and *lemma tactics* (which suggest the use of a lemma). The set of all tactics in the first four categories are sound and complete when tied together properly using tacticals: they are guaranteed to produce a natural deduction of the original goal without the use of any derived rules of inference. In this paper we concentrate on the description of the lemma tactics, which, in their simplest form, generate lemmas of the form $A \lor \neg A$. In order to understand when and why lemma tactics are invoked, we give a brief description of a few crucial minimal tactics in this section.

Tactic 1 ($\lor$ Introduction Left) *Assume our goal is to prove $A \lor B$ from $\mathcal{S}$. If the expansion proof for this goal provides a proof of A from $\mathcal{S}$, infer $A \lor B$ by $\lor \mathrm{I}_L$, leading to the subgoal of proving A from $\mathcal{S}$.*

This tactic and its obvious symmetric variant require that one of the disjuncts of the conclusion by itself follows from the assumptions. What if neither disjunct follows? If no assumption tactic applied, we would be forced to apply the rule of indirect proof $\bot_C$, unless we can find a good way of introducing a lemma. Such a situation occurs, for example, when trying to prove $[A \land B] \lor [A \land \neg B]$ from A (see the example in Section 2).

Tactic 2 ($\exists$ Introduction) *Assume our goal is to prove $\exists x A$ from $\mathcal{S}$. If the expansion proof for this goal contains a single instantiation term t for this occurrence of x, and the instantiation term contains no parameter which has yet to be introduced into the deduction, then infer $\exists x A$ by $\exists \mathrm{I}$ from $[t/x]A$, leaving the subgoal to deduce $[t/x]A$ from $\mathcal{S}$.*

This tactic may fail to apply for two different reasons (assuming no tactics can be applied to the assumptions). The proofs of the formula $\exists x \forall y[Px \to Py]$ and the example in Section 7 illustrate one reason for failure, namely that there is more than one instantiation term required for x, but at least one of them is free of parameters yet to be introduced into the deduction. The other, more insiduous failure is when there are one or more instantiation terms, but all of them still contain a parameter which has yet to be introduced. An example of this is the goal to deduce $\exists x \neg Px$ from $\neg \forall x Px$.

In either case we have to apply indirect proof or find a good lemma, though finding a reasonable lemma with our symmetric simplification algorithm is much more likely in the first situation.

As an example of an assumption tactic which may fail to be applicable, consider $\to$ Elimination.

Tactic 3 ($\to$ Elimination) *Assume our goal is to prove some conclusion C from $\mathcal{S}$ and $A \to B$. If the expansion proof for this goal provides a proof of A from $\mathcal{S}$ apply $\to$E and set up two new subgoals: one to deduce A from $\mathcal{S}$ and one to deduce C from $\mathcal{S}$, $A \to B$, and B.*

Tactic 3 may fail to apply if the negation of the conclusion is necessary to prove the antecedent A, for example when trying to deduce C from the assumption $\neg C \to C$. In such a case we have to apply proof by contradiction before $\to$E.

In the next section, we will analyze more thoroughly the cases in which these tactics do not apply, and will discuss how we try to generate useful and intuitive lemmas in these situations.

5 Setting up a Symmetric Simplification Problem

As outlined in Sections 2 and 4 and there are three situations in which symmetric simplification is of potential interest: they are exactly the situations in which one can no longer make progress by applying intuitionistic rules of inference. We refer to this as a *translation impasse*. In this section we attempt to illustrate how the symmetric simplification problem which is treated by our algorithm is set up, given that we have reached a translation impasse. It should be noted that these impasses are not mutually exclusive—we will return to this point at the end of this section.

Imagine again that we have some assumptions $\mathcal{S}$ and a conclusion A and our goal is to derive A from $\mathcal{S}$. Different types of impasses can arise. It is a non-trivial theorem which shows that these impasses are exhaustive, and thus at least one of these cases and possibly more must be applicable.

Disjunctive Impasse. $A = A' \lor A''$ and neither A' nor A'' alone follow from the assumptions, and no progress can be made by applying an intuitionistic rule to an assumption.

In this case, we have two symmetric opportunities for setting up a symmetric simplification problem. An obvious lemma is $A' \lor \neg A'$ with the intent to build the deduction

$$\dfrac{A' \lor \neg A' \qquad \dfrac{\begin{matrix}\mathcal{S}, [\![A']\!]^1\\ \mathcal{D}_1\\ A'\end{matrix}}{A' \lor A''}\lor\mathrm{I}_L \qquad \dfrac{\begin{matrix}\mathcal{S}, [\![\neg A']\!]^1\\ \mathcal{D}_2\\ A''\end{matrix}}{A' \lor A''}\lor\mathrm{I}_R}{A' \lor A''}\lor\mathrm{E}^1$$

Since $A' \lor A''$ follows from $\mathcal{S}$ it is easy to see that now A'' must follow from $\mathcal{S}$ and $\neg A'$. Also, the corresponding guiding expansion proof is easily constructed. It is much more difficult, though not impossible, to construct a normal natural deduction for this subproof, given an original deduction which proceeded using the rule of indirect proof.

In either case, we end up with two subproofs, one ($\mathcal{D}_1$) showing that $\mathcal{S}$ and $C = A'$ implies A', the other ($\mathcal{D}_2$) showing that $\mathcal{S}$ and $\neg C = \neg A'$ implies A''. This problem is now passed to the symmetric simplification algorithm with the goal of simplifying C and $\neg C$.

The symmetric case is where we pick $C = A'' \lor \neg A''$. Currently, our heuristic for choosing which disjunct to begin with is to choose the one which contains more negative literals which are used in the expansion proof. This heuristic is aimed at minimizing the number of negations in the lemma, and thereby reducing the number of negation elimination steps in the subsequent subproofs.

Existential Impasse. $A = \exists x\, A'$ and there is no single term t such that $[t/x]A$ follows from the assumptions, and no progress can be made by applying an intuitionistic rule to an assumption. Moreover, there must be at least one instantiation term t_0 for x in the proof of A from $\mathcal{S}$ which does not contain any parameter not yet introduced.

In this case we again use the lemma $[t_0/x]A' \vee \neg[t_0/x]A'$ with the intent to build the deduction

$$\dfrac{[t_0/x]A' \vee \neg[t_0/x]A' \qquad \dfrac{\begin{matrix}\mathcal{S}, [\![[t_0/x]A']\!]^1 \\ \mathcal{D}_1 \\ [t_0/x]A'\end{matrix}}{\exists x\, A'}\,\exists\mathrm{I} \qquad \begin{matrix}\mathcal{S}, [\![\neg[t_0/x]A']\!]^1 \\ \mathcal{D}_2 \\ \exists x\, A'\end{matrix}}{\exists x\, A'}\,\vee\mathrm{E}^1$$

Again, it is easy to construct an expansion proof showing that from $\mathcal{S}$ and $\neg[t_0/x]A'$ we can prove $\exists x\, A'$ with one fewer instantiation term for x. Clearly, this can also be done for natural deductions, though it is much more complicated.

In either case, we end up with two subproofs, one ($\mathcal{D}_1$) showing that $\mathcal{S}$ and $C = [t_0/x]A'$ implies $\exists x\, A'$, and one ($\mathcal{D}_2$) showing that $\mathcal{S}$ and $\neg C = \neg[t_0/x]A'$ implies $\exists x\, A'$. This problem is now passed to symmetric simplification algorithm with the goal of simplifying C and $\neg C$.

As in the disjunctive impasse, our heuristic for choosing which term to use for the lemma (if more than one may be used) is to chose the one which contains the most negative literals which are used in the expansion proof.

This leaves two kinds of impasses to which we will return in Section 8. Assuming that no tactic can be applied to an assumption, they are: (1) The conclusion is atomic and (2) the conclusion is existential and all of its substitution terms contain a parameter yet to be introduced. An example of (1) is the goal to derive A from $\neg A \rightarrow A$, and an example of (2) is to derive $\exists x \neg Px$ from $\neg\forall x Px$.

6 An Abstract Description of Symmetric Simplification

The technical details of symmetric simplification and our implementation are very closely tied to expansion proofs and their properties. In this section we will attempt to provide an intuition about the basic ideas behind symmetric simplification independently of expansion proofs.

Assume, as discussed in Section 5, that we have reached a point where we have set up the following problem:

Given two sets of assumptions $\mathcal{S}_1$ and $\mathcal{S}_2$, two conclusions A_1 and A_2, an initial formula C, and two proofs $\mathcal{D}_1$ and $\mathcal{D}_2$ such that $\mathcal{D}_1$ shows that A_1 follows from $\mathcal{S}_1$ and C, and $\mathcal{D}_2$ shows that A_2 follows from $\mathcal{S}_2$ and $\neg C$.

While keeping $\mathcal{S}_i$ and A_i fixed, vary C and $\mathcal{D}_i$ so as to "simplify" C and $\mathcal{D}_i$ as much as possible. In the absence of a formal measure of the degree of "intuitiveness" of the modified deductions, we will just try to shorten them as outlined below.

Our symmetric simplification procedure combines three steps, each of which is complex, but can be described in isolation. These steps are called *single instantiation*, *single deletion*, and *propositional restructure*.

In the exposition below, we call an occurrence of a subformula in C and the corresponding occurrence in $\neg C$ *dual occurrences*.

Single Instantiation. Let us assume that we have a positive occurrence of $\forall x\, C'$ somewhere in C which is instantiated to t in the deduction $\mathcal{D}_1$. This inference could be avoided, if we could replace the occurrence of $\forall x\, C'$ by $[t/x]C'$. In the dual formula $\neg C$, the dual occurrence is equivalent to $\exists x\, \neg C'$ which at some point in $\mathcal{D}_2$ might have been instantiated with a parameter a. This would be transformed into $\neg[t/x]C'$. The deduction $\mathcal{D}_2$ then would also have to be changed: we may have to permute inferences, and also substitute t for the parameter a in the deduction $\mathcal{D}_2$. The conditions on t guarantee that this is possible.

There are a number of other conditions which must be satisfied for the transformation outlined above to be valid.

1. the assumption $\forall x\, C'$ and none of its instances can be used elsewhere in $\mathcal{D}_1$, since it is replaced by $[t/x]C'$. Similarly for the dual occurrence in $\mathcal{D}_2$.
2. the substitution term t must not contain any parameter b which is introduced somewhere in $\mathcal{D}_1$ or $\mathcal{D}_2$, since this would invalidate the occurrence condition on the inference rule which introduced b (either an $\exists$-elimination or a $\forall$-introduction).

As one might imagine, it is difficult but not impossible to check these conditions on natural deductions. It requires that we can trace occurrences as they multiply and propagate through the deductions. In expansion proofs, it is trivial to check them. We currently only apply *single instantiation* if $\forall x\, C'$ does not lie below any other quantifier, though it would be easy to generalize our algorithm to search for cases where such a replacement would be legal.

Single Deletion. This is the most useful of the three steps. Let us assume we have a positive subformula occurrence $C' \wedge C''$ in C. If none of the instances of C' is used in the derivation $\mathcal{D}_1$ of A_1, we can erase it from C and $\neg C$. Note that this weakens C and thus strengthens $\neg C$, and thus no applicability checks need to be made in $\mathcal{D}_2$. However, $\mathcal{D}_2$ often simplifies. For example, if the dual assumption $\neg C' \vee \neg C''$ is strengthened to $\neg C''$, a proof by cases is simplified to one of its subproofs. Another case where essentially the same method applies is a negative subformula occurrence $C' \vee C''$ in C.

Propositional Restructure.[5] This is the most complex of the steps, and also perhaps the least intuitive. It was not part of the original algorithm we implemented, but we found it necessary for many examples. Because of redundancies introduced when creating

[5]In [22] in the context of expansion proofs this is called *single mating change*.

the lemma $C \vee \neg C$ it is often the case that an assumption C' (as in the scenario for *single deletion*) is used in $\mathcal{D}_1$, even though its use could be avoided if we changed $\mathcal{D}_1$. A general theorem prover is not very good at changing $\mathcal{D}_1$, since the goal is to avoid using any assumption in C as much as possible (which is hard to communicate to a theorem prover), and simply enumerating all subformulas and checking (with a theorem prover) whether A_1 still follows is completely impractical. This is not a completely accurate transcription of what our *single mating change* procedure does, but roughly our solution is to look through the places where forward reasoning and backward reasoning meet, that is, where gaps in the proofs are completely filled. If an instance of a subformula D of C is the only way the gap can be filled, we mark it as *necessary*, otherwise we locally restructure the proof (with propositional inferences only) so that D is not used. This is iterated until no more propositional restructure is possible, unless perhaps after a *single deletion*.

Propositional restructure may enable further *single deletions* and also *single instantiations*, since all instances of some subformula C' of C may now be unnecessary and could thus be deleted. In general, applying any of the above three steps may enable other steps to further simplify the deduction. We impose the following control structure:

1. Repeatedly apply *single instantiation* in a top-down fashion (top-level quantifiers first) until it is no longer applicable. Go to Step 2.

2. Repeatedly apply *single deletion* in a bottom-up fashion (leaves first) until it is no longer applicable. Go to Step 3.

3. Attempt to apply *propositional restructure*. If this fails and Step 2 did not perform any deletions, terminate: no further simplification is possible. Otherwise, go to Step 1.

7 An Example of Symmetric Simplification

In this section, we explain by means of an example how a lemma is generated when we have reached a translational impasse. Suppose we have three blocks, a, b, and c, such that b is on a and c is on b. Moreover, assume that we know that a is red, while c is known not to be red. From these assumptions, prove that there are some blocks x and y such that x is red, y is on x, and y is not red. Symbolically:

From $\mathrm{On}\, c\, b$, $\mathrm{On}\, b\, a$, $\mathrm{Red}\, a$, and $\neg\mathrm{Red}\, c$ conclude $\exists x \exists y\, [\mathrm{On}\, y\, x \wedge \neg\mathrm{Red}\, y \wedge \mathrm{Red}\, x]$

The following intuitive deduction which makes use of the law of excluded middle is the one our algorithm eventually generates.

Deduction 4 *1. Either b is red or it is not red.*

2. If b is red, then b and c satisfy the criterion for x and y, respectively, since c is on b and c is not red, while b is red.

3. If b is not red, then a and b satisfy the criterion for x and y, respectively, since b is on a and b is not red, while a is red.

A deduction which uses indirect proof will be much less intuitive. Here is one possible version[6].

Deduction 5 *We proceed by contradiction, thus (after some rules commuting negations with quantifiers and connectives) it remains to derive a contradiction from the assumptions*

$$\mathrm{On}\, c\, b,\ \mathrm{On}\, b\, a,\ \mathrm{Red}\, a,\ \neg\mathrm{Red}\, c,\ \textit{and}\ \forall x \forall y\, [\neg\mathrm{On}\, y\, x \vee \mathrm{Red}\, y \vee \neg\mathrm{Red}\, x]$$

The final of these assumptions must be true for $x = b$ and $y = c$ and also for $x = a$ and $y = c$, leading to a contradictory propositional formula.

From examining either of these two deductions, we can see that the original goal is at an *existential impasse* as described in Section 5; we cannot directly prove an instantiation of the conclusion, nor can we apply an intuitionistic elimination rule to an assumption and make progress. The reason is that x must somehow be instantiated to two different terms (b and c) to complete the deduction. Instead of using indirect proof, we set up a symmetric simplification problem and apply our earlier algorithm.

Examining the expansion proof, we find that b is an instantiation term for x such that b contains no parameters selected in the proof. It also satisfies our heuristic of containing more negative literals used in the expansion proof than does the tree for the other instantiation term a.

Thus we begin with $C_0 = \exists y\, [\mathrm{On}\, yb \wedge \neg\mathrm{Red}\, y \wedge \mathrm{Red}\, b]$, and the starting point for the lemma is $C_0 \vee \neg C_0$.

1. The existentially quantified subformula of $\neg C_0$ is negative, so the quantifier (on y) is essentially existential in the new subproof. In addition, it is instantiated with a single term c. Hence we can use *Single instantiation* to instantiate y with c. C_1 then becomes $\mathrm{On}\, c\, b \wedge \neg\mathrm{Red}\, c \wedge \mathrm{Red}\, b$; the lemma has now been simplified to $C_1 \vee \neg C_1$. Single instantiation cannot be applied again, so we go to step 2 of the algorithm.

2. *Single deletion* cannot be applied, since every literal of C_1 and $\neg C_1$ is being used in the current subproofs. Thus we go to step 3 of the algorithm, *propositional restructure.*

3. Note that C_1 contains as conjuncts two literals, $\mathrm{On}\, c\, b$ and $\neg\mathrm{Red}\, c$, which also appear as assumptions. Hence we can restructure the proof so that instead of using the literal occurrences in C_1, we use the corresponding assumptions. Once we have done that we go back to step 1 of the algorithm.

4. Single instantiation is not applicable (C_1 contains no quantified subformulas).

5. Single deletion can now be applied to the two literals of C_1 that were made unnecessary by the propositional restructure step above; we delete them and their dual occurrences in $\neg C_1$. C_2 becomes $\mathrm{Red}\, b$. The lemma now has been simplified to $\mathrm{Red}\, b \vee \neg\mathrm{Red}\, b$.

[6] A formal counterpart of this version would have been generated by an earlier version of the translation procedure.

6. Propositional restructure is no longer applicable; every literal in the lemma is necessary. Seeing that single instantiation and single deletion are also not applicable, we terminate.

Our lemma is $C_2 \vee \neg C_2$, that is, $\mathrm{Red}\, b \vee \neg \mathrm{Red}\, b$; this is exactly the lemma our intuition told us we should use. The resulting deduction is Deduction 4.

We chose this example because it illustrates each of the steps of the symmetric simplification algorithm. The result of the algorithm is a propositional lemma, which could leave the impression that lemmas could somehow be generated by looking at contradictory propositional formulas only. However, this is not always the case. As a case in point, consider the theorem $\exists x \forall y [Px \rightarrow Py]$. In the translation of this proof into a natural deduction, we immediately run into an existential impasse. If symmetric simplification is used rather than a proof by contradiction (which does not generate a nice proof) the lemma which results is $\forall y[Py] \vee \neg \forall y Py$.

8 Conclusion

For many theorems, symmetric simplification will provide a lemma that makes the natural deduction more intuitive. While we are guaranteed to make progress in the cases of existential and disjunctive impasse, that is not the case for other impasses; sometimes using indirect proof is "better" than using the lemma that our algorithm could create. This is true specifically for the situations mentioned at the end of Section 5. More work must be done in determining where symmetric simplification is useful in these cases, and criteria must be developed for evaluating the intuitiveness of proofs and for deciding whether using a given lemma yields an improvement or not. We have found one particular tactic that is often useful, though we do not yet have a good heuristic on when to use it. This tactic, which can be used when there is an implication in the assumptions, but the $\rightarrow$ elimination tactic (Tactic 3) does not apply, involves setting up a symmetric simplification problem from the left-hand side of the implication. The last two lemmas in the table below were generated by using this tactic.

Another problem that has not been addressed is that of choosing the "best" lemma. Instead of using a lemma of the form $C \vee \neg C$, we could use any formula $D \vee E$ such that $D \equiv C$ and $E \equiv \neg C$. This may make the resulting subproofs easier, at the expense of more difficulty in proving the lemma. Thus this only appears reasonable when a certain amount of sophistication has been achieved, that is, certain lemmas in a deduction can be assumed without proof. In addition, the lemmas that our algorithm constructs are all themselves theorems. Improvements may result if our lemmas are merely provable from the current assumptions.

Given all tactics and heuristics we have developed and implemented, there are still some remaining unintuitive deductions. They arise primarily due to negated, non-atomic assumptions combined with existential conclusions. The prototypical example of such situation is to derive $\exists x \neg Px$ from $\neg \forall x Px$. The best way we could find to deal with this problem is to assume that there are previously proven theorems or derived rules of inference which allow the permutation of negations with other connectives and quantifiers.

Finally, we end with some "benchmarks," simple theorems for which one of the impasses described in Section 5 arises, along with the lemma (or lemmas, when several are needed) which our algorithm generates.

Theorem	Lemma derived
$\exists y \forall x[Py \rightarrow Px]$	$\forall x Px \vee \neg\forall x Px$
$Pa \wedge \neg P[f[fa]] \rightarrow \exists x[Px \wedge \neg P[fx]]$	$\neg P[fa] \vee P[fa]$
$Pa \wedge \neg P[f[f[f[fa]]]] \rightarrow \exists x[Px \wedge \neg P[fx]]$	$\neg P[fa] \vee P[fa]$
	$\neg P[f[fa]] \vee P[f[fa]]$
	$\neg P[f[f[fa]]] \vee P[f[f[fa]]]$
$\exists x[Px \rightarrow P[fx]]$	$\neg P[fa] \vee P[fa]$
$A \rightarrow [A \wedge B] \vee [A \wedge \neg B]$	$\neg B \vee B$
$[A \rightarrow [B \vee C]] \rightarrow [[A \rightarrow B] \vee [A \rightarrow C]]$	$C \vee \neg C$
$[A \rightarrow B] \rightarrow [\neg A \vee B]$	$A \vee \neg A$
$\exists P P$	$P^0 \vee \neg P^0$
$[A \rightarrow \exists x Bx] \rightarrow \exists x[A \rightarrow Bx]$	$A \vee \neg A$
$[\forall x Px \rightarrow B] \rightarrow \exists x[Px \rightarrow B]$	$\forall x Px \vee \neg\forall x Px$

References

[1] Peter B. Andrews. Transforming matings into natural deduction proofs. In *Proceedings of the 5th Conference on Automated Deduction, Les Arcs, France*, pages 281–292. Springer-Verlag, 1980.

[2] Peter B. Andrews. On connections and higher-order logic. *Journal of Automated Reasoning*, 5:257–291, 1989.

[3] Peter B. Andrews, Sunil Issar, Daniel Nesmith, and Frank Pfenning. The TPS theorem proving system. In Ewing Lusk and Russ Overbeek, editors, *9th International Conference on Automated Deduction, Argonne, Illinois*, pages 760–761, Berlin, May 1988. Spring-Verlag LNCS 310. System abstract.

[4] Dana Angluin and Carl H. Smith. Inductive inference: Theory and methods. *Computing Surveys*, 15(3):237–269, September 1983.

[5] Joseph Bates and Robert Constable. Proofs as programs. *ACM Transactions on Programming Languages and Systems*, 7(1):113–136, January 1985.

[6] Michael Beeson. Some applications of Gentzen's proof theory in automated deduction. To appear, 1988.

[7] W. W. Bledsoe. The UT prover. Technical Report ATP-17B, Departments of Mathematics and Computer Science, University of Texas at Austin, 1983.

[8] Daniel Chester. The translation of formal proofs into English. *Artificial Intelligence*, 7:261–278, 1976.

[9] Robert L. Constable et al. *Implementing Mathematics with the Nuprl Proof Development System.* Prentice-Hall, Englewood Cliffs, New Jersey, 1986.

[10] Thomas Ellman. Explanation-based learning: A survey of programs and perspectives. *ACM Computing Surveys*, 21(2):163–221, June 1989.

[11] Amy P. Felty. Using extended tactics to do proof transformations. Technical Report MS-CIS-86-89, Department of Computer and Information Science, University of Pennsylvania, 1986.

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

[13] Michael J. Gordon, Arthur J. Milner, and Christopher P. Wadsworth. *Edinburgh LCF*. Springer-Verlag LNCS 78, 1979.

[14] Xiaorong Huang. A human oriented proof presentation model. Technical Report SR-89-11, SEKI, 1989.

[15] Georg Kreisel and Angus MacIntyre. Constructive logic versus algebraization I. In A. S. Troelstra and D. van Dalen, editors, *The L.E.J. Brouwer Centenary Symposium*, pages 217–260. North-Holland Publishing Co., 1982.

[16] Christoph Lingenfelder. Transformation of refutation graphs into natural deduction proofs. Technical Report SR-86-10, SEKI, 1986.

[17] Christoph Lingenfelder. Structuring computer generated proofs. In N. S. Sridharan, editor, *Proceedings of the Eleventh IJCAI*, pages 378–383. Morgan Kaufmann, 1989.

[18] Dale Miller. *Proofs in Higher-Order Logic*. PhD thesis, Carnegie-Mellon University, October 1983.

[19] Dale Miller. Expansion tree proofs and their conversion to natural deduction proofs. In R. E. Shostak, editor, *Proceedings of the 7th Conference on Automated Deduction*, pages 375–393, Heidelberg, May 1984. Springer Verlag.

[20] Dale A. Miller. A compact representation of proofs. *Studia Logica*, 46(4):347–370, 1987.

[21] Frank Pfenning. Analytic and non-analytic proofs. In R.E. Shostak, editor, *7th Conference on Automated Deduction*, pages 394–413. Springer-Verlag, May 1984.

[22] Frank Pfenning. *Proof Transformations in Higher-Order Logic*. PhD thesis, Carnegie Mellon University, January 1987.

[23] Frank Pfenning, Sunil Issar, Dan Nesmith, and Peter B. Andrews. *ETPS User's Manual*, fifth edition, 1984. 44+ii pp.

[24] Dag Prawitz. *Natural Deduction*. Almquist & Wiksell, Stockholm, 1965.

[25] Edward H. Shortliffe. *MYCIN: a Rule-Based Computer Program for Advising Physicians Regarding Antimicrobial Therapy Selection*. PhD thesis, Stanford University, 1974.

[26] Richard Statman. Bounds for proof-search and speed-up in the predicate calculus. *Annals of Mathematical Logic*, (15):225–287, 1978.

[27] William van Melle. *A Domain-Independent System that Aids in Constructing Knowledge-Based Consultation Programs*. PhD thesis, Stanford University, 1980.