Structuring Computer Generated Proofs

Christoph Lingenfelder

<lingenf@uklirb.uucp>
Fachbereich Informatik
University of Kaiserslautern, Postfach 3049
D-6750 Kaiserslautern, West Germany

Abstract

One of the main disadvantages of computer generated proofs of mathematical theorems is their complexity and incomprehensibility. Proof transformation procedures have been designed in order to state these proofs in a formalism that is more familiar to a human mathematician. But usually the essential idea of a proof is still not easily visible. We describe a procedure to transform proofs represented as abstract refutation graphs into natural deduction proofs. During this process topological properties of the refutation graphs can be exploited in order to obtain structured proofs.

1 Introduction

A problem for the acceptance of Automated Deduction Systems has been the difficulty to understand proofs that are automatically generated. If this has been an obstacle for mathematicians to accept automatic help, when proving technical lemmata, or trying to find proofs interactively, it has even more hindered the explanation of results in other knowledge based systems. The transformation of these proofs into a natural deduction formulation has solved some of the problems, see [Andrews, 1980, Miller, 1983, Lingenfelder, 1986, Pfenning, 1987], but by and large the increasing length and complexity of the transformed proofs adds to their incomprehensibility rather than to reduce it. It is therefore paramount to be able to state the proofs in a hierarchically structured way, as mathematicians do, formulating subgoals and lemmata. It should also be avoided to overload the proofs with trivial steps, thus hiding its main interesting ideas.

We aim to simplify and transform proofs that are found automatically into that subset of natural language a mathematician might use. This shall be done in several steps. In a first step the automatically constructed proof is transformed into a natural deduction proof, which is still formal but more human-oriented than most other formats. During this process the proof is already structured by the introduction of lemmata and subgoals. Then the proof lines are arranged in a graph representing their dependencies, which allows grouping of lines and a gradual linearization of the natural deduction proof in accordance with its logical structure. Finally a simplified version of this natural deduction proof is to be transformed into an intermediate representation, upon which structural, and stylistic procedures operate in order to find a "human like" proof style and to transform it into mathematical natural language.

2 Definitions

This chapter explains the proof representation formalisms needed in this paper. Exact definitions can be found in Shostak [1976], Lingenfelder [1986], or Eisinger [1988].

2.1 Clause Graphs

Definition: A *clause graph* consists of a set of literal nodes, that are partitioned into clause nodes. Each literal node is labelled with a literal, the distinction between the literal nodes and the literals themselves is needed because the same literal may be attached to several literal nodes. Finally the links of the clause graph connect sets of literal nodes, such that for all links the following conditions hold:

(πi) All the literal nodes in a link are labelled with literals with unifiable atoms.

(π_2) A link must connect at least one positive and one negative literal.

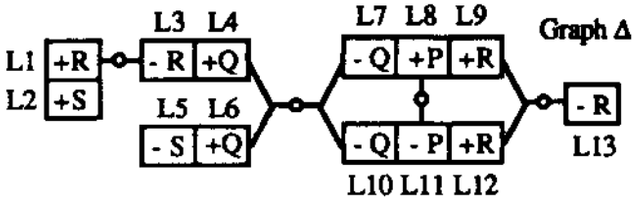
Each link A has two opposite *shores,* a *positive shore* $S^+(A)$, and a *negative shore* S"(A), constituted by the literal nodes with positive and negative literals, respectively. Literal nodes belonging to no link at all are called pure.

A clause graph F is called a *subgraph* of a clause graph r, if it can be obtained from T by any number of the following actions:

(S_1 Remove literal nodes from a link. If the resulting set of literal nodes no longer fulfils condition π_2, the complete link is removed from the graph.

(S_2) Remove clause nodes from the graph. At the same time all its literal nodes must be removed from their respective links according to (s_1.

Below you find an example of a clause graph. Literal nodes are drawn as boxes with the appropriate literals inside. It can be seen that the same literal may belong to several literal nodes. Therefore literal nodes cannot be identified by their literals and the labelling outside of the boxes is for their identification. The example contains six clause nodes, built up by bordering literal nodes, and four links: {L_1 L_3), {L_4, L_6, L_7, L_{10}), {L_8, L_{11}), and [L_9, L_{12},L_{13}). They are drawn as lines with a little dot, which branch on each side to connect the different literal nodes of the opposite shores. The literal nodes L2 and L5 are pure.



Definition: A *walk* in a clause graph T is an alternating sequence $C_o II_1 C_1 \ldots C_{n-1} II_n C_n$ (n>1) of clause nodes and links such that for every pair of clause nodes C_{i-1} $C_i$one of them contains a literal node of the positive shore of the connecting link II_i and the other contains a literal node of its negative shore.

A link is *separating* T, if there exist two clause nodes C and D connected by a walk in T, that are no longer connected when the link is removed. In the example all links except (L_8, L_{11}) are separating. A *trail* in a clause graph T is a walk, where all the links used are distinct. A *cycle* is a trail, whose start and end clause nodes coincide. A *deduction graph* is an acyclic clause graph. A *refutation graph* is a deduction graph without pure literal nodes. The example graph A is a deduction graph; it could be extended to a refutation graph by inserting an additional link {L_2, L_5).

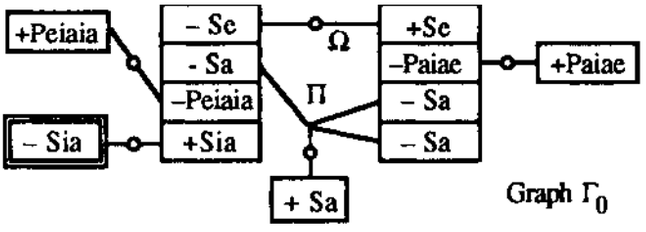
A refutation graph represents a proof for the unsatisfiability of the set of its clauses, while a deduction graph represents a derivation for the disjunction of its pure literals. As an example we give a refutation graph for a theorem known as (part of) the subgroup criterion:

Main Example: *Let G be a group, and let SCG. If for all elements x, y in S the element $x°y^{1}$ is also in S, then for every x in S its inverse x^{1} is also in S.*

This theorem must be formulated in first order logic for the automatic theorem prover. Here Pxyz means x0y=z in a group, Sx means x E S, a subset of the group, and the function i calculates the inverse of the group elements:

$$(\forall u\ Puiue) \wedge (\forall w\ Peww) \wedge$$
$$(\forall xyz\ Sx \wedge Sy \wedge Pxiyz \Rightarrow Sz) \Rightarrow (\forall x\ Sx \Rightarrow Six)$$

The refutation graph T_0 was generated automatically by our theorem prover MKRP, [Eisinger and Ohlbach, 1986]. For the purpose of this paper, we assume that it is given.



2.2 Natural Deduction Proofs

Definition: *A proof line* of natural deduction consists of a finite, possibly empty set of formulae, called the *assumptions,* a single formula, called *conclusion,* and a justification. A proof line with assumptions A, conclusion F and justification $\mathfrak{R}$ is written $\mathfrak{A} \vdash F\ \mathfrak{R}$. Sometimes comments are given to make the proof easier to read, they are written as if they were proof lines. A finite sequence S of proof lines is a *Natural Deduction Proof (NDP)* of a formula F, if

(α) F is the conclusion of the last line of S,
(β) the set of assumptions of this last line is empty,
(γ) every line in S is justified by one of the rules of natural deduction. A complete set of such rules is described by Andrews [1980].

The construction of natural deduction proofs, by humans and computers alike, is conducted in single steps. To prove any valid formula F one always starts with a line 1— F. Such a line is obviously no proof, because it is not correctly justified. Now the proof is constructed by deriving subgoals until the proof is completed. In the intermediate states, called *proof outlines* by Andrews [1980], one may find completed subproofs, but also others that are not yet done. To formalize the procedure of the search for such a natural deduction proof, we use *Generalized Natural Deduction Proofs (GNDPs).* They differ from natural deduction proofs only in point (γ). This allows lines not correctly justified within the calculus, but it is assumed that these lines are "correct", in the sense that a proof exists for (Apremises => conclusion) in an arbitrary formalism, for instance as a refutation graph. Such lines are called *external* lines, lines justified within the calculus are called *internal* When no external lines are present in a GNDP, it is an ordinary NDP.

A GNDP consisting of just one line, which is an external line without premises and with conclusion F, is called the *trivial GNDP* for F. In order to find a natural deduction proof for a formula F, for which a proof π is known, a finite sequence of GNDPs can be constructed, whose first element is the trivial GNDP, and whose last element is an NDP for F. The transition between consecutive GNDPs is governed by the set of rules described in [Lingenfelder, 1986]. As an example, here are two of the rules:

Example: In the description of the transformation rules, A is a list of assumption formulae, capital letters indicate single formulae, small greek letters are used as labels for the lines, the justification R stands for an arbitrary rule of the natural deduction calculus, and the justifications Π, π_1, and π_2 represent proofs of the respective lines. In any case one must make sure that the proofs π_1 and π_2 can be constructed from π or are otherwise known.

E∧:

	(γ) $\mathfrak{A}$	⊢	$F_1 \wedge F_2$	π
⟶	(α) $\mathfrak{A}$	⊢	F_1	π_1
	(β) $\mathfrak{A}$	⊢	F_2	π_2
	(γ) $\mathfrak{A}$	⊢	$F_1 \wedge F_2$	Tau(α,β)

M-Cases:

	(α) $\mathfrak{A}$	⊢	A∨B	$\mathfrak{R}$
	(ζ) $\mathfrak{A}$	⊢	F	π
	(α) $\mathfrak{A}$	⊢	A∨B	$\mathfrak{R}$
	We consider separately the cases of (α)			
	Case 1:			
	(β) $\mathfrak{A}$, A	⊢	A	Hyp
⟶	(γ) $\mathfrak{A}$, A	⊢	F	π_1
	Case 2:			
	(δ) $\mathfrak{A}$, B	⊢	B	Hyp
	(ε) $\mathfrak{A}$, B	⊢	F	π_2
	End of cases (1, 2) of (α)			
	(ζ) $\mathfrak{A}$	⊢	F	Cas(α,γ,ε)

Main Example (cont'd): Starting with the trivial GNDP for our theorem,

(16) |-- $(\forall uPuiue) \wedge (\forall wPeww) \wedge (\forall xyz\ Sx \wedge Sy \wedge Pxiyz \Rightarrow Sz) \Rightarrow (\forall x\ Sx \Rightarrow Six)$ Γ_0

where Γ_0 is the refutation graph of our earlier example, we reach an intermediate GNDP after some applications of transformation rules. Lines 1 and 2 introduce new assumptions, which are in turn removed in lines 14 and 16. Line 13 is not correctly justified within the natural deduction formalism, but by the refutation graph proving the theorem, therefore the proof is a GNDP rather than an NDP.

(1)	1	|-- $(\forall uPuiue) \wedge (\forall wPeww) \wedge (\forall xyz\ Sx \wedge Sy \wedge Pxiyz \Rightarrow Sz)$	Hyp

Let a be an arbitrary constant

(2)	2	|-- Sa	Hyp
(13)	1, 2	|-- Sia	Γ_0
(14)	1	|-- $Sa \Rightarrow Sia$	Ded(13)
(15)	1	|-- $\forall x\ Sx \Rightarrow Six$	$\forall G(14)$
(16)		|-- $(\forall uPuiue) \wedge (\forall wPeww) \wedge (\forall xyz\ Sx \wedge Sy \wedge Pxiyz \Rightarrow Sz) \Rightarrow (\forall x\ Sx \Rightarrow Six)$	Ded(15)

3 The Structure of Proofs Inherent in Topological Properties of Refutation Graphs

3.1 Trivial Subproofs

In the following subsections, subgraphs of the orginal refutation graph will be viewed as deduction graphs representing lemmata in a larger proof. This only makes sense, when the deduction graph in question is complex enough to warrant the introduction of a lemma. Otherwise it may be better to repeat a trivial argument instead of using a lemma. It is of course not straightforward to decide whether a deduction graph is trivial. To make a decision we use a heuristic approach taking into account several properties of the graphs involved.

It is indeed not easy to find objective criteria to decide when a proof is trivial. Davis [1981] proposes that *"an inference is obvious, precisely when a Herbrand proof of its correctness can be given involving no more than one substitution instance of each clause".* Pelletier and Rudnicki [1986] argue along the same lines, but point out that in general it may be difficult to decide if any proof of a given fact is non-obvious because this requires to check a property of all possible proofs. This doesn't pertain to our case, however, since we are only concerned with the question if a given proof is trivial as opposed to the question whether an obvious proof can be found for a given theorem.

So Davis' approach seems to be a good starting point, however there is an additional complication. We have to figure out whether a given proof (deduction graph) is a substantial part of a larger proof. When this is the case, it is normally desirable to use the subgraph as a lemma or as an intermediate step in the overall proof. Therefore we must check, if the rest of the proof - after removing the proof for the lemma - has become "easier". According to Davis this will be the case when the subgraph contains an instance of a clause, of which a different instance appears somewhere else in the rest of the proof. It may even be the case that both resulting proofs are obvious although the total proof wasn't. But that's what dividing large proofs into steps is all about.

Finally, when it comes to make someone understand a proof, other non-logical properties must also be considered. For example its absolute length and the length in relation to the total proof must be taken into account. When the subproof is relatively long, it will always pay to prove it separately as a lemma. If this lemma is already known to the reader one may later dispense with its proof altogether. Doing this intelligently requires a database of known lemmata and a model of the reader's knowledge about the field of mathematics in case. When a freshman uses the system as an explanation for a proof one should not omit arguments which a graduate student might consider trivial. Conversely, it may obscure the idea of a complex proof to mention all the applications of lemmata that have been thoroughly understood long before. As one never knows, however, who will read the proof later, it is useful to postpone this decision as long as possible. At this stage it is not yet necessary to take a user model into account, this will only be done when the natural deduction proof is finally brought into a well-structured linear form.

3.2 Shared Subgraphs as Lemmata

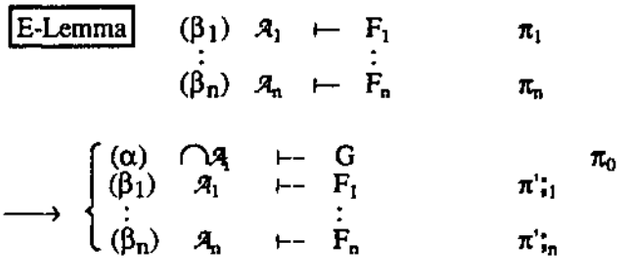
Now we assume that a proof for a formula φ has already been found by an automated deduction system. We will further assume that this proof is represented as a refutation graph T, a form that can easily be constructed from a resolution proof, see [Posegga, 1985, Lehr, 1988]. This means, however, that the equivalent problem of proving the unsatisfiability of the negated formula in a special normal form has been solved. In addition to the refutation graph, we therefore need a correspondence between the literal nodes of T and the atom occurrences in φ. Full details of the material presented in this chapter can be found in [Lingcnfelder, 1988].

An initial "trivial" generalized natural deduction proof can now be constructed to start a transformation process as described in [Lingcnfelder, 1986]. After each application of a transformation rule a number of tasks need to be performed in order to guarantee a smooth transformation process.

1. The relation between literal nodes of the refutation graph and atom occurrences of the conclusion formula of every proof line must be established.
2. The refutation graph is changed or divided according to the rule applied.
3. Additional parts of the refutation graph may become positively polarized; a clause node is positively polarized, when each of its literal nodes corresponds to an atom occurrence of an axiom or a current assumption. This neatly reflects the general idea of natural deduction proofs, where new assumptions are introduced during the proof process.

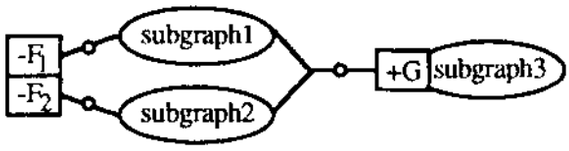
Some of the transformation rules, EA for instance, lead to new external lines, and as a consequence to a division of the refutation graph. In the simplest case the refutation graph proving $F_1 \wedge F_2$ is "cut" through the clause $[-F_1\ -F_2]$, such that the two resulting components arc refutation graphs for F and for F_2. In general these graphs may have a $non\text{-}_{empty}$ intersection, and this is similarly the case for other rules leading to a division of the refutation graph.

If this intersection is comparatively small, it may easily be duplicated and then used twice in the two subproofs. If it is relatively large, however, it may be sensible to prove a lemma first and then use it in both proofs. In order to formalize such a procedure, a new transformation rule E-Lemma is introduced.

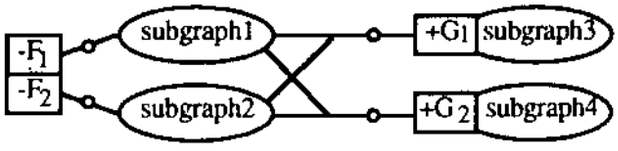


This rule must of course be used with discretion, i.e. only when an appropriate formula G could be found, which simplifies the proof of the formulae F_i. In particular it may only be used, when all the literal nodes in the refutation graph π_0 are positively polarized, so that it is possible to prove G from axioms and current assumptions only. It goes without saying, that π_0 must be a common subgraph of all the graphs π_i. In constructing the graphs π' one is entitled to use the formula G as an additional axiom. The case n=I may also be meaningful, when a lemma is introduced as a subgoal, see section 3.3.

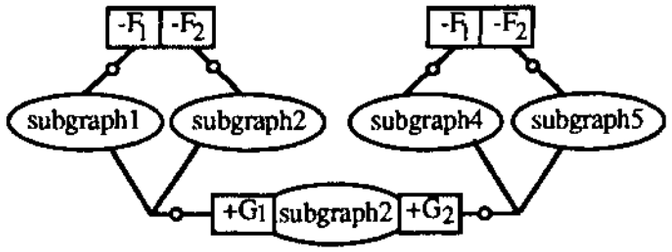
Let us consider for a moment what these shared subgraphs may look like. We always assume that a cut is being made in order to apply EA. In the simplest case the lemma consists of just one atom G. Then the graph has the form



The case where G is a conjunction $G_1 \wedge G_2$ is almost as simple. There are now two independently shared parts.



When G is a disjunction $G_1 \vee G_2$, however, things are no longer as easy. One might think that it suffices to introduce a link between the two subgraphs of the previous case. It is true that we could now prove the disjunction, but a cycle is introduced into the graph, which therefore ceases to be a refutation graph. In fact, a shared subgraph representing a disjunction can only occur, when the theorem $F_1\ F_2$ appears more than once in the graph, as in the next example.

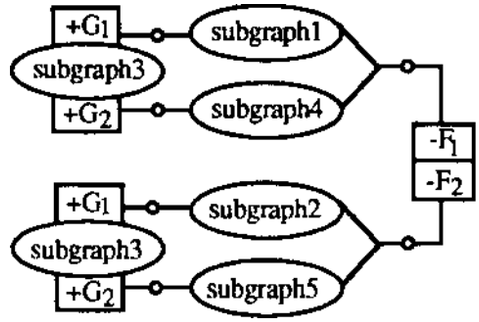


After cutting through both clauses $[-F_1\ -F_2]$ the graph contains refutation subgraphs for F_1 and F_2 sharing subgraph3. In both cases the proof can be done by cases after the lemma $G_1 \vee G_2$ has been introduced.

Already Shostak [1979] mentioned, that unsatisfiable ground clause sets exist, for which every refutation graph contains at least one of its clauses twice. But in this case one can inhibit the duplication of any specific clause.

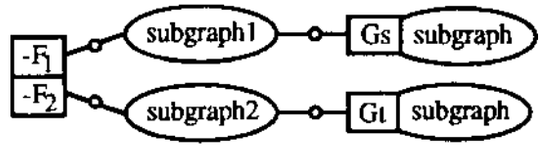
Lemma: For every unsatisfiable ground clause set S containing a clause C, one can construct a refutation graph, which contains C only once.

If one chooses the theorem clause $[-F_1\ -F_2]$ to appear only once, the graph of the last example takes the form shown below. Now the subgraph proving $G_1 \vee G_2$ is no longer shared, but two copies of it exist in the refutation graph.



So in general one has to search for isomorphic subgraphs that are complex enough to warrant the introduction of a lemma. In addition to an isomorphic graph structure corresponding literal nodes must represent identical literals and must be related to the same atom occurrences in the original formula. This condition may, however, be relaxed for term arguments of the free (lemma) literals, if they correspond to variables, that need not be instantiated in the subgraph. In this case the lemma becomes a quantified formula used more than once in different instantiations.

Such a lemma corresponds to a resolvent used more than once during the resolution proof. Thus, if the refutation graph was originally constructed from a resolution proof, one should keep this information in order to obviate the search for that kind of lemma. An example can easily be constructed by slightly altering the above graph.



3.3 Separating Links that Define Subgoals

In the previous chapter, the main incentive for the introduction of a lemma was to avoid unnecessary duplication in the proof. But this is not the only reason, why mathematicians use lemmata. Often they are used purely to structure the proof, so that its main ideas become better visible.

In an automatic proof transformation it is obviously difficult to find meaningful lemmata. And it is here again that the topological structure of the refutation graph may successfully be exploited. The task is to find parts of the refutation graph that are sufficiently complex in order to justify the introduction of a lemma, while they should at the same time be easily separable from the rest of the graph. Besides,

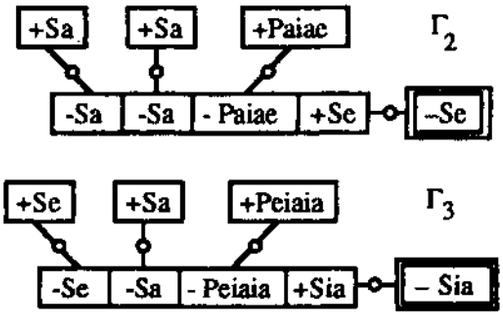
all the parts belonging to the proposed lemma must have become positively polarized before. If it were possible to find a link or a small set of links separating the refutation graph, and fulfilling these requirements, one might use the positively polarized part as a lemma. The search for such links is performed by the following algorithm:

Algorithm:

1. Starting with a non-trivial refutation graph Γ (cf. section 3.1), we want to compute a set Ψ of candidate links for a lemma. Start with $\Psi:=\{\}$.
2. Compute the set Ψ_S of the links separating the graph. The removal of a given $\Lambda\in\Psi_S$ from Γ results in two deduction graphs $\Gamma_+(\Lambda)$ and $\Gamma_-(\Lambda)$.
3. Compute $\Psi_0:=\{\Lambda\in\Psi_S \mid$ exactly one of $\Gamma_+(\Lambda)$ and $\Gamma_-(\Lambda)$ is trivial$\}$. These are useless, since they can only lead to trivial lemmata.
4. Let $\Psi:=\Psi\cup(\Psi_S\setminus\Psi_0)$.
5. For each $\Lambda\in\Psi_0$ duplicate the trivial subgraph $\Gamma_{tr}(\Lambda)$, if Λ is branching on the non-trivial side. Note that Γ is changed by this operation. In particular it may now have a set Ψ_S of additional separating links.
6. If Ψ_S is non-empty, go to 3, else continue with 7.
7. Now Ψ is the set of candidates for lemma purposes.

Main Example (cont'd): We will now resume our example. In constructing the current GNDP only transformation rules were applied not using, or changing, the refutation graph at all. As this is no longer possible at this stage, we try to detect separating links applying our algorithm to the graph Γ_0.

All the links except Ω are separating a trivial part from the rest of the graph. Therefore Π is now broken up and the clause [Sa] is triplicated. After that Ω is separating; the clause node [-Sia] in double bars is the only negatively polarized one. So one part of Γ_0 without Ω is all positive, and each part is trivial in itself. Using Davis' definition of obviousness, one has to check that the literal nodes of all clause nodes correspond to different atom occurrences of the original formula. In the complete graph the two clauses [-Sa -Sa -Paiae Se] and [-Se -Sa -Peiaia Sia] are different instances of the same formula occurrence. Thus Ω represents the lemma Se. Note that this lemma really has an interesting meaning, namely that the neutral element of the group is contained in the subset S. The two graphs Γ_2 and Γ_3 below result from cutting the graph at Ω, representing proofs for Se and Sia (using Se) respectively.



Now one proceeds to the next GNDP using the transformation rule E-Lemma:

(1)	1	⊢–	(∀uPuiue) ∧ (∀wPeww) ∧ (∀xyz Sx ∧ Sy ∧ Pxiyz ⇒ Sz)	Hyp

Let a be an arbitrary constant

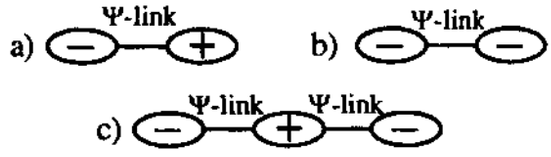
(2)	2	⊢–	Sa	Hyp
(8)	1, 2	⊢–	Se	Γ_2
(13)	1, 2	⊢–	Sia	Γ_3
(14)	1	⊢–	Sa ⇒ Sia	Ded(13)
(15)	1	⊢–	∀x Sx ⇒ Six	∀G(14)
(16)		⊢–	(∀uPuiue) ∧ (∀wPeww) ∧ (∀xyz Sx ∧ Sy ∧ Pxiyz ⇒ Sz) ⇒ (∀x Sx ⇒ Six)	Ded(15)

In order to prove Se we first work on Γ_2, then we continue with Γ_3 until the natural deduction proof is completed.

Unless the set Ψ of candidate links for the construction of a lemma is a singleton, as in the example above, we must define an actual lemma by selecting one or several of these links. In order to do this, we must try to isolate parts of the graph containing only positively polarized clause nodes, which are connected to the rest of the graph only via Ψ-links. The following algorithm describes, how this is done.

Algorithm:

1. For all negatively polarized clause nodes C^- compute $\Sigma_\Psi(C^-)$, the maximal connected subgraph, which includes C^- but does not contain any $\Lambda\in\Psi$. These graphs are deduction graphs, all of whose pure literal nodes were connected to the rest of Γ with Ψ-links.
2. For all positively polarized clause nodes C^+, not belonging to any of the $\Sigma_\Psi(C^-)$, compute $\Sigma_\Psi(C^+)$. Now the graph is divided into "positive" and "negative" subgraphs in one of three basically different ways. In all three cases, variations may occur due to separating links that are branching.



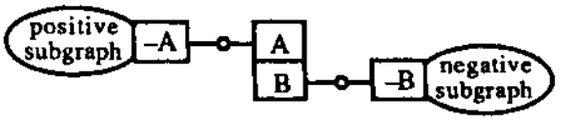
3a) If there is only one $\Sigma_\Psi(C^-)$, the set of Ψ-links attached to its pure literal nodes is used as a lemma, which can be derived from the positive rest of the graph, i.e. directly from axiom formulae and current assumptions. In this case the lemma is the conjunction of the literal nodes in the opposite shores of the pure literal nodes.

b) If two of the $\Sigma_\Psi(C^-)$ are adjacent, then the proof is separated into cases.

c) If there is a $\Sigma_\Psi(C^+)$ between two of the $\Sigma_\Psi(C^-)$, then one has to check, if the positive part is trivial, in which case one proceeds as in b). Otherwise the positive subgraph defines a disjunctive lemma, which will then be used to perform a proof by case analysis.

3.4 Structuring Proofs by Case Analysis

One of the transformation rules defined by Lingenfelder [1986], called M-Cases, leads to a division of the refutation graph. It can always be applied, when a disjunction has been derived earlier. An application is however undesirable in most cases, as can be seen from the following examples:



a) If only one of the resulting components contains negatively polarized literal nodes, then an extra and unnecessary proof by contradiction must be performed.
b) If the two resulting parts overlap widely, including negatively polarized literal nodes, then large parts of the proof will be duplicated in both cases.

A good case for the application of M-Cases appears, when both of the resulting components contain parts of the theorem, and their overlap is either small or restricted to positively polarized parts, in which case a lemma can be defined to avoid the duplication, cf. cases 3b and 3c above.

The most important case for the rule M-Cases comes up, when an existentially quantified formula cannot be proven constructively. In the refutation graph, this fact is reflected by the existence of several copies of the theorem clauses. M-Cases can now be applied, if all the resulting components contain just one of these copies.

4 Final Remarks

After the transformation process from a refutation graph into a natural deduction formalism the proof must now be ordered in accordance with its logical structure. A first algorithm to structure natural dedcution proofs has been proposed by Chester [1975]. He starts his transformation process from a given, completely unstructured natural deduction proof, having no information of how it was constructed. However, if an NDP was constructed by the method described above, one already knows about lemmata from the topological structure of the refutation graph as described before in section 3.

When the natural deduction graph has been ordered, some further steps are required to make the proof really understandable. The main drawback of natural deduction proofs is their length and the difficulty in seeing the important steps. One has to distinguish therefore between trivial proof steps and more important steps, which is not straightforward, as the answer depends on the context of the proof as well as on the intended reader. After all, a mathematician will consider a lot of proof steps trivial, that inexperienced readers might not find easy at all. This raises the question how this distinction can be made automatically.

A first approach will group several steps, especially when only propositional reasoning is involved. But it may also be indicated to combine propositional steps with an instantiation. If it is known, however, that the proof will appear in a text book immediately after the proof of some lemma, or that an expert will read the proof, a complete subproof may be omitted. In order to achieve this sort of reader dependent simplification of the proof it will be necessary to have a model. The development of user models is a well known research problem in AI, especially in the field of natural language processing and computer interfaces.

Summary: In this paper we have seen, that we can exploit the topological structure of computer generated proofs, in order to break them up into smaller lemmata. In particular this may avoid the need to prove a subformula more than once, when it is shared by different branches of the proof. In addition the information implicit in the topological properties of refutation graphs is used to structure the proof. This is done by dividing the graph into disjoint parts to be proved separately, either sequentially, as a lemma cited later in the proof, or as a proof by case analysis. In order to do this, the algorithm for the transformation of refutation graphs into natural deduction proofs had to be extended.

The same information also facilitates the process of ordering the natural deduction proof. The parts to be brought into a meaningful order are much smaller, thus reducing the number of arbitrary decisions that have to be made to choose an actual sequence of proof lines. In fact one has to solve several smaller linearization problems instead of a single large one; of course one also has to find a sequence of the lemmata in the end.

The final ordered version of the natural deduction proof is then used as a starting point for removing trivial steps from the proof. In general this can only be done with the help of a user model. When all of this has been done one can tackle the problem to state this formal proof in mathematical natural language, which is the topic of our current research interest, see [Huang, 1989].

References

[Andrews, 1980] Peter B. Andrews, *Transforming Matings into Natural Deduction Proofs,* Lecture Notes in Comp. Sci. 87 (CADE 1980), 281-292

[Chester, 1975] David Chester, *The Translation of Formal Proofs into English,* AI 7 (1976), 261-278

[Davis, 1981] Martin Davis, *Obvious Logical Inferences,* Proceedings of the 7th UCAI, Vancouver 1981

[Eisinger, 1988] Norbert Eisinger, *Completeness, Confluence, and Related Properties of Clause Graph Resolution,* PhD Thesis, Uni Kaiserslautern, 1988, SR-88-07

[Huang, 1989] Xiaorong Huang, *A Human Proof Presentation Model and Proof Transformation,* SEKI-Report, Uni Kaiserslautern, to appear

[Eisinger and Ohlbach, 1986] Norbert Eisinger, Hans J. Ohlbach, *The Markgraf Karl Refutation Procedure,* Lecture Notes in Comp. Sci. 230 (CADE 1986), 682f.

[Lingenfelder, 1986] Christoph Lingenfelder, *Transformation of Refutation Graphs into Natural Deduction Proofs,* SR-86-10, Uni Kaiserslautern

[Lingenfelder, 1988] Chr. Lingenfelder, *Structuring Computer Generated Proofs,* SR-88-19, Uni Kaiserslautern

[Lehr, 1988] Siegfried Lehr, *Transformation von Resolutionsbeweisen des MKRP,* Studienarbeit Uni Kaiserslautern, 1988

[Miller, 1983] Dale Miller, *Proofs in Higher Order Logic,* Ph.D. Thesis, Carnegie Mellon University (1983)

[Posegga, 1985] Joachim Posegga, *Using Deduction Graphs as a Representation for Resolution Proofs,* Diploma Thesis, Uni Kaiserslautern, 1986

[Pelletier and Rudnicki, 1986] Francis J. Pelletier, Piotr Rudnicki, *Non-Obviousness,* AAR Newsletter 6(1986)

[Pfenning, 1987] Frank Pfenning, *Proof Transformation in Higher-Order Logic,* PhD Thesis, Carnegie Mellon University, 1987

[Shostak, 1976] Robert E. Shostak, *Refutation Graphs,* A17(1976), 51-64

[Shostak, 1979] Robert E. Shostak, A *Graph Theoretic View of Resolution Theorem-Proving,* Report SRI International, Menlo Park, CA (1979)