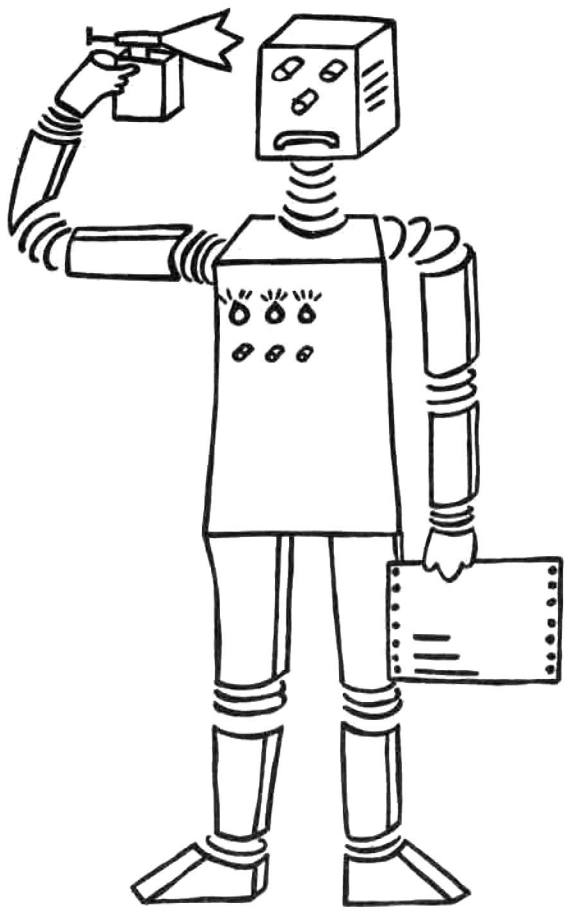
Fachbereich Informatik
Universität Kaiserslautern
Postfach 3049
D-6750 Kaiserslautern 1, W. Germany

Artificial
Intelligence
Laboratories

SEKI-REPORT



Transformation of Refutation Graphs into Natural Deduction Proofs

Christoph Lingenfelder

SEKI-Report SR-86-10 Dezember 1986

Abstract

Most computer generated proofs are stated in abstract representations not normally used by mathematicians. We describe a procedure to transform proofs represented as abstract refutation graphs into natural deduction proofs. The emphasis of this investigation is more on stylistic aspects rather than theoretical issues. In particular the topological properties of refutation graphs can be successfully exploited in order to obtain structured proofs.

Contents

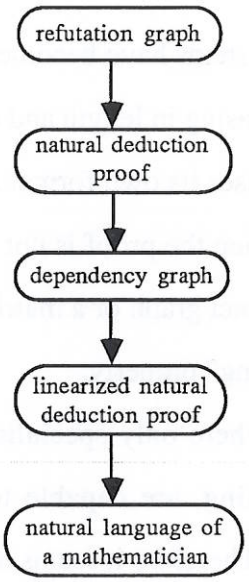
1. Introduction

Automated Deduction Systems have become more and more efficient in recent years, but the proofs they find are also increasing in length and complexity. To add to their incomprehensibility, almost every research group uses its own format and style of stating a proof. It may be given as a pure Resolution Proof, but when the proof is not actually found in distinct single steps, the result may be as complex as an abstract graph or a matrix with some additional conditions imposed on it such as acyclity or the "spanning" property.

This has led to a state where only specialists, and sometimes only specialists in the very method of automated reasoning, are capable to understand and check a proof found by an automated deduction system. Therefore it seems necessary to be able to represent proofs in a more comprehensible way.

Proof Transformation is an old problem of logic, but it has been neglected in a quest for automatically finding proofs in just any representation. Its main aspects used to be theoretical in nature, but now stylistic aspects begin to play a more important role. Peter Andrews was the first to take up these issues again, when he proposed a method to transform matrix proofs into natural deduction proofs [An80].

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. Then the proof lines are structured in a graph representing their mutual dependencies, which allows grouping of the single lines and a gradual linearization of the natural deduction proof. Finally this natural deduction proof is transformed into an intermediate representation, upon which simplification, structural, and stylistic procedures operate in order to find a "human like" proof style. This representation is then transformed into mathematical natural language.

2. General Definitions

This chapter contains the basic definitions of the logic used. Furthermore literals, clauses, and the notion of unifiability are defined. There are no important differences to the usual way of defining these concepts; similar definitions can for instance be found in [Lo78].

2.1 Terms and Substitutions

2.1-1 Definition:

We define a signature $\mathbb{F}$ as the union of the sets of constant symbols $\mathbb{F}_0$, and the sets $\mathbb{F}_n$ of n-ary function symbols (n = 1, 2, ...); $\mathbb{K}$ and all the $\mathbb{F}_n$ are finite. Let $\mathbb{V}$ be a denumerable set of variable symbols. Then the term set $\mathbb{T}$ is the least set with

(α) $\mathbb{V}, \mathbb{K} \subseteq \mathbb{T}$

(β) if $f \in \mathbb{F}_n$ and $t_1, t_2, \ldots, t_n \in \mathbb{T}$, then $ft_1t_2\ldots t_n \in \mathbb{T}$.

A term containing no variables is called a ground term. $\mathbb{T}_{gr}$ is the set of all ground terms. $\mathbb{V}(o)$ is an abbreviation for the set of variables occurring in an arbitrary object o, and the same convention is similarly used for $\mathbb{F}_n$, $\mathbb{F}$, $\mathbb{T}$, and $\mathbb{T}_{gr}$.

2.1-2 Definition:

A substitution is a mapping $\sigma: \mathbb{V} \rightarrow \mathbb{T}$ with finite domain $V=\{v \mid \sigma(v) \neq v\}$; $\sigma(V)$ is called the codomain of σ. A substitution σ with domain $\{x_1,x_2,\ldots,x_n\}$ and codomain $\{t_1,t_2,\ldots,t_n\}$ is represented as $\{x_1 \mapsto t_1,\ldots,x_n \mapsto t_n\}$. A substitution is extended to a mapping $\mathbb{T} \rightarrow \mathbb{T}$ by the usual homomorphism on terms. The application of a substitution to any other object containing terms is defined analogously.

A substitution σ is idempotent, if $\sigma\circ\sigma=\sigma$. This is equivalent to the requirement that none of the variables of its domain occurs in any of the terms of its codomain. In this report all substitutions will be idempotent. If a substitution maps into $\mathbb{T}_{gr}$, it is called a ground substitution.

2.1-3 Definition:

Let $s,t \in \mathbb{T}$. A matcher from s to t is a substitution μ with $\mu s=t$. A unifier of s and t is a substitution σ with $\sigma s=\sigma t$. If a unifier for s and t exists, then the two terms are said to be unifiable.

2.2 First Order Predicate Logic and Clauses

2.2-1 Definition:

We introduce the set $\mathbb{P}=\bigcup_{0\le n}\mathbb{P}_n$ consisting of finite sets of n-ary predicate symbols (n=0,1,...). There are two special zero-place predicate symbols, TRUE (written $\top$) and FALSE (written $\bot$). The objects of the form $Pt_1t_2\ldots t_n$ with $P\in\mathbb{P}$ and $t_1,t_2,\ldots,t_n\in\mathbb{T}$ constitute the set of atoms $\mathbb{A}$. If A is an atom, then +A and –A are (complementary) literals. The set of all literals is $\mathbb{L}$.

A finite set of literals is called a clause, $\mathbb{C}$ is the set of all clauses. A clause with literals $L_1,\ldots,L_n$ is written as $[L_1 \ldots L_n]$.

2.2-2 Definition:

Two literals are unifiable, if their signs are equal and their atoms are unifiable. They are called resolvable, whenever their signs are different and their atoms unifiable.

2.2-3 Definition:

To construct the formulas of First Order Predicate Logic, we use the following additional signs:

(a)	Unary connective	$-$	negation sign
(b)	Binary connectives	$\wedge$	conjunction sign
		$\vee$	disjunction sign
		$\Rightarrow$	implication sign
(c)	Quantors	$\forall$	universal quantifier
		$\exists$	existential quantifier

The set Φ of formulas of First Order Predicate Logic is now defined as usual:

(α) $\mathbb{L} \subseteq \Phi$

(β) If $A,B \in \Phi$, then $A \wedge B$, $A \vee B$, and $A \Rightarrow B$ are all in Φ.

(γ) If $A \in \Phi$, then $\forall x\, A \in \Phi$ and $\exists x\, A \in \Phi$.

(δ) All members of Φ can be described in this way.

$A \Leftrightarrow B$ is used as an abbreviation for $(A \Rightarrow B) \wedge (B \Rightarrow A)$. Furthermore we write $\forall x_1,x_2,\ldots,x_n A$ as an abbreviation for $\forall x_1 \forall x_2 \ldots \forall x_n A$ and similarly for the existential quantor. If $\mathcal{M}=\{M_1,M_2,\ldots,M_n\}$ is a finite set of formulas, we write $\bigwedge \mathcal{M}$ or $\bigwedge_{1\le i\le n} M_i$ instead of $M_1 \wedge M_2 \wedge \ldots \wedge M_n$ and likewise $\bigvee \mathcal{M}$ or $\bigvee_{1\le i\le n} M_i$ instead of $M_1 \vee M_2 \vee \ldots \vee M_n$.

Parentheses are used to indicate the range of the connectives, as in $((-A) \wedge (B \vee C))$. The outermost parentheses will be omitted most of the time, and we adopt the usual convention to define a binding order of the connectives. We assume that $-$ binds more strongly than $\wedge$ and $\vee$, these in turn bind more strongly than $\Rightarrow$ and $\Leftrightarrow$, and the quantors $\forall$ and $\exists$ are the weakest. Parentheses may be omitted according to this binding hierarchy, so that the above formula could be written $-A \wedge (B \vee C)$.

3. Proof Representations

The Kaiserslautern Theorem Prover works on a connection graph of clauses and uses many different deduction rules, not only simple resolution steps, for a deduction of the empty clause, see [MKRP84]. Some of the deduction rules, such as replacement resolution, cf.[Prä85], can easily be translated into simple resolution steps, while others directly rely upon graph properties, and it would be rather complicated to transform these into resolution steps. The output of this theorem prover is therefore not a resolution style proof but a graph representing the proof. One of the advantages of the graph representation is that it is possible to apply macro steps to the graph during the search for a proof without having to go through an equivalent sequence of single resolution steps.

The resulting graph, that finally represents a proof, is called a refutation graph; a detailed definition can be found in [Ei87]. In the next paragraph refutation graphs are defined, omitting any details not needed here, for in this report refutation graphs are only seen as abstract representations of proofs.

3.1 Refutation Graphs

3.1-1 Definition:

A clause graph is a quadruple $G = (\mathbb{N},[\mathbb{N}],£, \mathbb{\Pi})$, where

(a) $\mathbb{N}$ is a finite set. Its members are called the literal nodes of G.

(b) $[\mathbb{N}] \subset 2^{\mathbb{N}}$ is a partition of the set of literal nodes. The members of $[\mathbb{N}]$, which are classes of literal nodes are called the clause nodes of G. Contrary to the standard definition of a partition, $\emptyset \in [\mathbb{N}]$ is allowed. The clause node of a literal node L is denoted by [L].

(c) $£: \mathbb{N} \rightarrow \mathbb{L}$ is a mapping, which labels the literal nodes with literals, such that if $L,K \in \mathbb{N}$ belong to different clause nodes, then $\mathbb{V}(£L) \cap \mathbb{V}(£K) = \emptyset$.

(d) The set of polylinks $\mathbb{\Pi}$ is a partition of a subset of $\mathbb{N}$, such that for all $\Lambda \in \mathbb{\Pi}$ the following polylink condition holds:

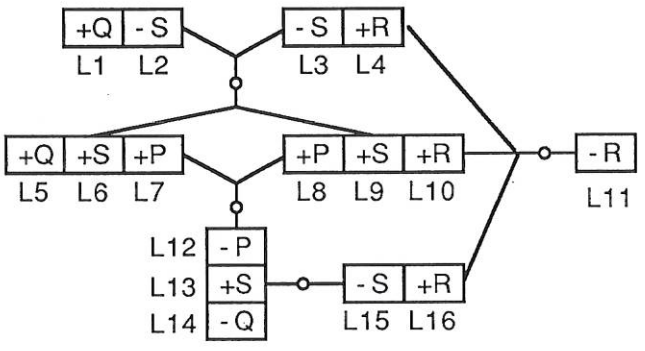
(π_1) All the literal nodes in one polylink are labelled with literals whose atoms are unifiable.

(π_2) There must be at least one positive and one negative literal in a polylink.

Literal nodes belonging to no polylink at all are called pure, $\mathbb{N}_p$ is the set of all pure literal nodes. Each polylink Λ has two opposite shores, a positive shore $S^+(\Lambda)$, and a negative shore $S^-(\Lambda)$, constituted by the literal nodes with positive and negative literals, respectively.

These clause graphs, developed in [Ei87] are a generalization of Kowalski's connection graphs, [Ko75], and R. Shostak's refutation graphs, [Sh79].

3.1-2 Example:



Here is 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 seven clause nodes, built up by bordering literal nodes. There are four polylinks, {L4, L10, L16, L11}, {L2, L3, L6, L9}, {L7, L8, L12}, and {L13, L15}. Polylinks 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 L1, L5, and L14 are pure.

It is often necessary to change a given clause graph G by adding or removing any parts. Since this involves several sets of nodes, one has to define carefully what the resulting graph will be. Adding a polylink Λ to a clause graph G means to change $\mathbb{\Pi}$ by adding to it a new set of literal nodes of the previously pure literal nodes; the polylink conditions π_1 and π_2 (cf. 3.1-1) must of course be obeyed.

Adding a literal node, means to add a new pure literal node to one of the existing clause nodes. And to add a clause node is to insert a new set of pure literal nodes to G making up a new clause node. Since there is normally no ambiguity, all these operations are written using the same + sign.

Similarly, to remove a polylink Λ from a clause graph G means to make its literal nodes pure, i.e. to add them to $\mathbb{N}_p$. Removing a literal node L from G is to remove it from its clause node and to change its polylink, unless it is pure. L is simply removed from its shore and, if the shore becomes empty, the whole polylink is removed. Note that, if L was the only literal node in a clause node, then the empty clause remains in the graph.

A clause node is removed by removing it from $[\mathbb{N}]$ and all of its literal nodes from $\mathbb{N}$. Removal of any part of a clause graph is written using the $-$ sign. We will now give a rigorous definition.

3.1-3 Definition:

Let $G=(\mathbb{N},[\mathbb{N}],\pounds,\mathbb{\Pi})$, and let $\Lambda\subseteq\mathbb{N}_p$ fulfil the polylink conditions π_1 and π_2.

Let $L\notin\mathbb{N}$, $D\in[\mathbb{N}]$, and $C\cap\mathbb{N}=\emptyset$.

Then $G+\Lambda = (\mathbb{N},[\mathbb{N}],\pounds, \mathbb{\Pi}\cup\{\Lambda\})$,

$G,D+L = (\mathbb{N}\cup\{L\}, [\mathbb{N}]\setminus\{D\} \cup \{D\cup\{L\}\},\pounds', \mathbb{\Pi})$,

where $\pounds'$ is an extension of $\pounds$ with $\pounds' L\in\mathbb{L}$.

$G+C = (\mathbb{N}\cup C, [\mathbb{N}]\cup\{C\},\pounds', \mathbb{\Pi})$,

where $\pounds'$ is an extension of $\pounds$ with $\pounds' L_i\in\mathbb{L}$ for all $L_i\in C$.

Let $G = (\mathbb{N},[\mathbb{N}],\pounds, \mathbb{\Pi})$ with $L\in C\in[\mathbb{N}]$ and $L\in\Lambda\in\mathbb{\Pi}$. Then

$G-\Lambda = (\mathbb{N},[\mathbb{N}],\pounds, \mathbb{\Pi}\setminus\{\Lambda\})$

$G,C-L = (\mathbb{N}\setminus\{L\}, [\mathbb{N}]\setminus\{C\} \cup \{C\setminus\{L\}\}, \pounds|_{\mathbb{N}\setminus\{L\}}, \mathbb{\Pi}')$

with $\mathbb{\Pi}' = \begin{cases} \mathbb{\Pi}, & \text{if L was pure} \\ \mathbb{\Pi}\setminus\{\Lambda\}, & \text{if } S^+(\Lambda)=\{L\} \text{ or } S^-(\Lambda)=\{L\} \\ \mathbb{\Pi}\setminus\{\Lambda\}\cup\{\Lambda\setminus\{L\}\}, & \text{else} \end{cases}$

$G-C = (\mathbb{N}\setminus C, [\mathbb{N}]\setminus\{C\}, \pounds|_{\mathbb{N}\setminus\{C\}}, \mathbb{\Pi}'')$,

where $\mathbb{\Pi}''$ is constructed similar to $\mathbb{\Pi}'$ by removing all literal nodes of C.

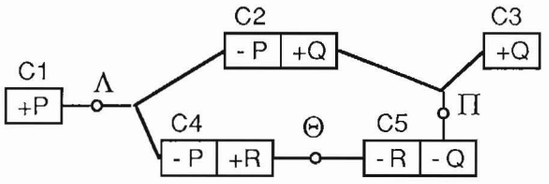
G' is a <u>subgraph</u> of a clause graph G, if it can be obtained from G by removing any number of clause nodes and polylinks.

3.1-4 Definition:

A <u>walk</u> in a clause graph G is an alternating sequence $C_0\Pi_0C_1\ldots C_{n-1}\Pi_nC_n$ ($n\geq1$) of clause nodes and polylinks such that for every pair of clauses C_j, C_{j+1} one contains a literal node of the positive shore of the connecting polylink Π_j and the other contains a literal node of its negative shore. Seeing clauses and polylinks as sets of literal nodes this means for all n either $C_{n-1}\cap S^+(\Pi_n)\neq\emptyset$ and $C_n\cap S^-(\Pi_n)\neq\emptyset$ or $C_{n-1}\cap S^-(\Pi_n)\neq\emptyset$ and $C_n\cap S^+(\Pi_n)\neq\emptyset$.

A set of links or a link Λ is <u>separating</u> G, if there exist two clause nodes C and D connected by a walk in G, that are no longer connected in $G-\Lambda$.

3.1-5 Example:



This is an example of a clause graph, where any two clause nodes are connected by a walk. Both Λ and Π are separating polylinks, but Θ is not.

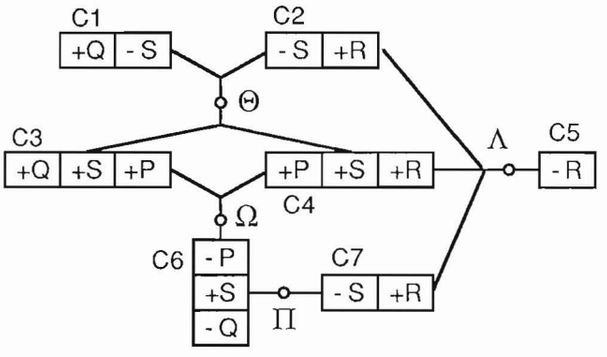
3.1-6 Definition:

A poly-trail in a clause graph G is a walk, where all the poly-links used are distinct. A poly-trail joins its start and end clause nodes C_0 and C_n.

A poly-cycle is a poly-trail joining a clause node to itself. If a clause graph G contains such a cycle it is called poly-cyclic, otherwise poly-acyclic. It is called poly-connected, if each pair of clause nodes is joined by a poly-trail.

A poly-component of a clause graph G is a maximal poly-connected subgraph of G.

3.1-7 Example:



This is the same example graph G as in 3.1-2 with the emphasis on clause nodes and polylinks. Two examples of poly-trails are $C_2\Theta C_4\Omega C_6$ and $C_5\Lambda C_7\Pi C_6\Omega C_3\Theta C_1$. There is no poly-cycle and since there exists no poly-trail between C_1 and C_2, the graph is not poly-connected, although any two clauses are connected by a walk.

G contains four poly-components, namely $G-\{C_1, C_3\}$, $G-\{C_1, C_4\}$, $G-\{C_2, C_3\}$ and $G-\{C_2, C_4\}$.

With the insertion of a further poly-link Φ, consisting of the three pure literal nodes, the graph becomes poly-cyclic; one poly-cycle is for instance $C_6\Omega C_3\Theta C_1\Phi C_6$.

3.1-8 Definition:

Let Λ and Π be polylinks in a clause graph. Λ is less nested than Π, $\Lambda \prec \Pi$, if there exist clause nodes C and D, containing literal nodes of the same shore of Λ, and joined by a poly-trail using Π.

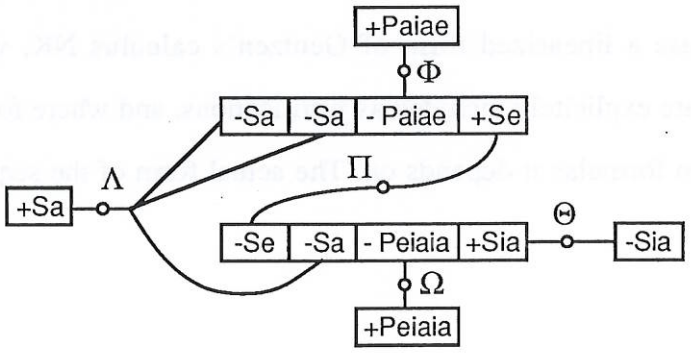
For example see 3.1-10 below, where Λ is less nested than Π.

3.1-9 Definition:

A deduction graph is a non-empty, ground, and poly-acyclic clause graph. A refutation graph is a deduction graph without pure literal nodes. We sometimes speak of deduction or refutation graphs even if they are not ground, but then the existence of a substitution is required that transforms them into ground graphs.

A minimal deduction (refutation) graph is one containing no proper subgraph, which is itself a deduction (refutation) graph.

3.1-10 Example: (refutation graph)



Later on we will often drop the distinction between clause nodes and literal nodes and the clauses and literals themselves. Furthermore we will sometimes omit the prefix "poly-" because we never use the simple objects analogously defined in [Ei87].

3.2 Natural Deduction Proofs

In 1933, Gerhard Gentzen developed a formal system for mathematical proofs with the intention to describe as closely as possible the actual logical inferences used in mathematical proofs („der möglichst genau das richtige logische Schließen bei mathematischen Beweisen wiedergibt" [Ge35]). The main difference between these natural deduction proofs (NDPs) and proofs in the earlier axiomatic systems by Frege, Russell, and Hilbert is that inferences are drawn from assumptions rather than from axioms.

Prawitz describes such systems of natural deduction in [Pr65]: "The inference rules of the systems of natural deduction correspond closely to procedures common in intuitive reasoning, and when informal proofs – such as are encountered in mathematics for example – are formalized within these systems, the main structure of the informal proofs can often be preserved".

We use a linearized form of Gentzen's calculus NK, where the dependencies between formulas are explicitely included as justifications, and where for every formula we give the set of assumption formulas it depends on. The actual form of the sequent rules is taken from Andrews [An80].

3.2-1 Definition:

A proof line of natural deduction consists of

(a) a finite, possibly empty set of formulas, called the assumptions

(b) a single formula, called conclusion

(c) a justification.

A proof line with assumptions $\mathcal{A}$, conclusion F and justification Rule X is written $\mathcal{A} \vdash$ F Rule X. Sometimes comments are given to make the proof easier to read, these

comments are then 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 premises of this last line is empty

(γ) Every line in S is justified by one of the rules below.

Hypothesis Rule (Hyp): Infer $\mathcal{A}, F \vdash F$,
this rule introduces a new assumption.

Deduction Rule (Ded): from $\mathcal{A}, F \vdash G$ infer $\mathcal{A} \vdash F \Rightarrow G$

Rule of Propositional Calculus (Tau):

from $\mathcal{A}_1 \vdash F_1, \ldots,$ and $\mathcal{A}_n \vdash F_n$
infer $\mathcal{A}_1, \ldots, \mathcal{A}_n \vdash G$,
provided that $F_1 \wedge \ldots \wedge F_n \Rightarrow G$ is tautologous

Negation Rule (Neg):

from $\mathcal{A} \vdash F$ infer $\mathcal{A} \vdash G$,

where F equals $\neg(\forall x\, H)$, $\neg(\exists x\, H)$, $\forall x \neg H$, or $\exists x \neg H$;
and G equals $\exists x \neg H$, $\forall x \neg H$, $\neg(\exists x\, H)$, or $\neg(\forall x\, H)$, respectively.

Rule of Indirect Proof (IP): from $\mathcal{A}, \neg F \vdash \bot$ infer $\mathcal{A} \vdash F$

Rule of Cases (Cas): from $\mathcal{A} \vdash F \vee G$
and $\mathcal{A}, F \vdash H$ and $\mathcal{A}, G \vdash H$
infer $\mathcal{A} \vdash H$

Universal Generalization ($\forall$G): from $\mathcal{A} \vdash G \vee F \vee H$,
infer $\mathcal{A} \vdash G \vee (\forall x\, F) \vee H$,
provided that x is not free in $\mathcal{A}$, G, or H.

Existential Generalization (∃G): Let F(x) be a formula, and let t be a term
from $\mathcal{A} \vdash G \vee F(t) \vee H$,
infer $\mathcal{A} \vdash G \vee \exists x A(x) \vee H$,

provided that x is not free in F(t).

Universal Instantiation (∀I): from $\mathcal{A} \vdash \forall x\, F(x)$ infer $\mathcal{A} \vdash F(t)$
for arbitrary terms t.

Rule of Choice (Sel): from $\mathcal{A} \vdash \exists x\, Fx$ and $\mathcal{A}, Fc \vdash G$
infer $\mathcal{A} \vdash G$, when $c \in F_0$ is not free in $\mathcal{A}$ or G.

Rule of alphabetic Change of bound Variables (αβ):

This rule allows to change the names of bound variables.

3.2-2 Example:

As an example let us prove that if $\forall x\, (Px \Rightarrow Qx)$ and Pa for a constant a, then $\exists y\, Qy$:

(1)	1	⊢	$\forall x\, (Px \Rightarrow Qx)$	Hyp
(2)	2	⊢	Pa	Hyp
(3)	1	⊢	$Pa \Rightarrow Qa$	∀I(1)
(4)	1, 2	⊢	Qa	Tau(2,3)
(5)	1, 2	⊢	$\exists y\, Qy$	∃G(4)
(6)	1	⊢	$Pa \Rightarrow \exists y\, Qy$	Ded(5)
(7)		⊢	$(\forall x\, Px \Rightarrow Qx) \Rightarrow (Pa \Rightarrow \exists y\, Qy)$	Ded(6)
(8)		⊢	$(\forall x\, Px \Rightarrow Qx) \wedge Pa \Rightarrow \exists y\, Qy)$	Tau(7)

The proof lines have numbers, which are used for two purposes:
(a) in the justification, to indicate which other lines a given line depends on, and
(b) to abbreviate an assumption formula; a number in the place of an assumption formula stands for the formula introduced by the hypothesis rule in the line with this number.

Note that the reasoning is done exclusively with the conclusion formulae, while the assumptions are only carried along to emphasize the interdependencies between the formulae. This is characteristic of Gentzen's natural deduction system NK, whereas the calculi of sequents, as for example Gentzen's LK also change the formulae of the antecedent.

4. Proof Transformation

4.1 Generalized Natural Deduction Proofs

The construction of natural deduction proofs (NDPs), by humans and computers alike, is conducted in single steps. To prove any valid formula F one always starts with a line $\vdash F$. 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 in [An80], one may find completed subproofs, but also others that are not yet done. To formalize this procedure we introduce generalized Natural Deduction Proofs.

4.1-1 Definition:

A finite sequence S of proof lines is called a Generalized Natural Deduction Proof (GNDP) of a formula F, if

(a) F is the conclusion of the last line of S

(b) the last line of S has no assumption

(c) every line is either justified by a rule of the calculus (see above), or it is justified by a proof (possibly in a different calculus) of its conclusion from its premises.

This allows lines not correctly justified within the calculus, but it is assumed that these lines are "correct", in the sense that a proof for $(\bigwedge \text{premises} \Rightarrow \text{conclusion})$ exists. 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 a normal 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.

4.1-2 Example:

In this example we give one possible generalized NDP for the formula

F := (∀uPuiue)∧(∀wPeww)∧(∀xyz Sx∧ Sy ∧ Pxiyz ⇒ Sz) ⇒ (∀x Sx ⇒ Six).

This is a formulation of part of the subgroup criterion:

Let G be a group, S⊆G; if for all x,y in S, $y^{-1}\circ x$ is also in S, then for every x in S its inverse is also in S.

(1) 1	⊢	(∀uPuiue) ∧ (∀wPeww) ∧ (∀xyz Sx∧ Sy ∧ Pxiyz ⇒ Sz)	Hyp
Let a be an arbitrary constant			
(2) 2	⊢	Sa	Hyp
(3) 1, 2	⊢	Sia	π
(4) 1	⊢	Sa ⇒ Sia	Ded(3)
(5) 1	⊢	∀x Sx ⇒ Six	∀G(4)
(6)	⊢	(∀uPuiue)∧(∀wPeww)∧(∀xyz Sx∧ Sy ∧ Pxiyz ⇒ Sz) ⇒ (∀x Sx ⇒ Six)	Ded(5)

As a proof π see the refutation graph in example 3.1-10.

In order to find a natural deduction proof for a formula F, for which a proof π has been found, a finite sequence of generalized NDPs 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 the next chapter.

4.2 Transformation Rules for Generalized NDPs

In order to generate a sequence of GNDPs ending in a natural deduction proof for a valid formula F, it is necessary to describe the rules by which a GNDP is constructed from its predecessor in the sequence. In the following example such a sequence is shown for the NDP of example 3.2-2. This example should throw some light on the nature of the transition rules.

4.2-1 Example:

We start with the trivial GNDP for the formula to be shown. During this example it is assumed that the proofs π_i are always known.

$GNDP_1$:	(7)		$\vdash (\forall x\, Px \Rightarrow Qx) \wedge Pa \Rightarrow \exists y\, Qy$	π_1

To prove this implication, we may additionally assume its left hand side. The proof of the right hand side together with the deduction rule will then complete the proof.

$GNDP_2$:	(1)	1	$\vdash (\forall x\, Px \Rightarrow Qx) \wedge Pa$	Hyp
	(6)	1	$\vdash \exists y\, Qy$	π_2
	(7)		$\vdash (\forall x\, Px \Rightarrow Qx) \wedge Pa \Rightarrow \exists y\, Qy$	Ded(6)

To show the existence of Qy, a representative must be found.

$GNDP_3$:	(1)	1	$\vdash (\forall x\, Px \Rightarrow Qx) \wedge Pa$	Hyp
	(5)	1	$\vdash Qa$	π_3
	(6)	1	$\vdash \exists y\, Qy$	$\exists G(5)$
	(7)		$\vdash (\forall x\, Px \Rightarrow Qx) \wedge Pa \Rightarrow \exists y\, Qy$	Ded(6)

Now a subformula containing Q is isolated from an internal formula by applying propositional rules.

$GNDP_4$:	(1)	1	$\vdash (\forall x\, Px \Rightarrow Qx) \wedge Pa$	Hyp
	(2)	1	$\vdash (\forall x\, Px \Rightarrow Qx)$	Tau(1)
	(5)	1	$\vdash Qa$	π_4
	(6)	1	$\vdash \exists y\, Qy$	$\exists G(5)$
	(7)		$\vdash (\forall x\, Px \vee Qx) \wedge Pa \Rightarrow \exists y\, Qy$	Ded(6)

The next step is to instantiate the universally quanified formula, such that Qa appears in it.

$GNDP_5$:	(1)	1	$\vdash (\forall x\, Px \Rightarrow Qx) \wedge Pa$	Hyp
	(2)	1	$\vdash (\forall x\, Px \Rightarrow Qx)$	Tau(1)
	(3)	1	$\vdash Pa \Rightarrow Qa$	$\forall$I(2)
	(5)	1	$\vdash Qa$	π_5
	(6)	1	$\vdash \exists y\, Qy$	$\exists$G(5)
	(7)		$\vdash (\forall x\, Px \Rightarrow Qx) \wedge Pa \Rightarrow \exists y\, Qy$	Ded(6)

Now Qa holds, if Pa can be shown.

$GNDP_6$:	(1)	1	$\vdash (\forall x\, Px \Rightarrow Qx) \wedge Pa$	Hyp
	(2)	1	$\vdash (\forall x\, Px \Rightarrow Qx)$	Tau(1)
	(3)	1	$\vdash Pa \Rightarrow Qa$	$\forall$I(2)
	(4)	1	$\vdash Pa$	π_6
	(5)	1	$\vdash Qa$	Tau(3,4)
	(6)	1	$\vdash \exists y\, Qy$	$\exists$G(5)
	(7)		$\vdash (\forall x\, Px \Rightarrow Qx) \wedge Pa \Rightarrow \exists y\, Qy$	Ded(6)

And Pa follows from 1 in propositional logic, which leads to the desired natural deduction proof.

$GNDP_7$ (NDP):	(1)	1	$\vdash (\forall x\, Px \Rightarrow Qx) \wedge Pa$	Hyp
	(2)	1	$\vdash (\forall x\, Px \Rightarrow Qx)$	Tau(1)
	(3)	1	$\vdash Pa \Rightarrow Qa$	$\forall$I(2)
	(4)	1	$\vdash Pa$	Tau(1)
	(5)	1	$\vdash Qa$	Tau(3,4)
	(6)	1	$\vdash \exists y\, Qy$	$\exists$G(5)
	(7)		$\vdash (\forall x\, Px \Rightarrow Qx) \wedge Pa \Rightarrow \exists y\, Qy$	Ded(6)

There are three largely different groups of rules. Rules of the first group, external rules, insert a justification within the calculus for a previously external line, and insert other external lines. It is in fact a form of backward reasoning. Examples for this type are the transitions between $GNDP_1$ and $GNDP_2$ or between $GNDP_2$ and $GNDP_3$.

For the second type, mixed rules, both internal and external lines are used to change the GNDP, which type has been used in the construction of $GNDP_5$ from $GNDP_6$.

Rules of the third group, internal rules, only apply rules of the calculus to internal lines, deducing further internal lines, but do not affect any external line. This last type of rules uses previously proved formulas or axioms to derive new ones by means of forward reasoning. With the rules of this type reasoning can be done in propositional logic, but also instances of universally quantified formulae can be built.

In the following sections we shall give a formal account of these transition rules. In their description, $\mathcal{A}$ is a list of assumption formulas, capital letters indicate single formulae, small greek letters are used as labels for the lines, the justification Rule $\mathfrak{R}$ stands for an arbitrary rule of the natural deduction calculus, and the justifications π, π',π_1, and π_2 represent proofs of the respective lines. For all these rules one must make sure that the proofs π',π_1, or π_2 can be constructed from π or are otherwise known.

4.2.1 External Rules:

The lines on the left hand side of the arrow ($\longrightarrow$) are replaced by those on the right hand side in the next generalized NDP of the sequence.

E⇒:

$$(\gamma)\ \mathcal{A} \vdash F \Rightarrow G\ \pi \longrightarrow \begin{cases} (\alpha)\ \mathcal{A}, F \vdash F & \text{Hyp} \\ (\beta)\ \mathcal{A}, F \vdash G & \pi' \\ (\gamma)\ \mathcal{A} \vdash F \Rightarrow G & \text{Ded}(\beta) \end{cases}$$

E∧:

$$(\gamma)\ \mathcal{A} \vdash F \wedge G\ \pi \longrightarrow \begin{cases} (\alpha)\ \mathcal{A} \vdash F & \pi_1 \\ (\beta)\ \mathcal{A} \vdash G & \pi_2 \\ (\gamma)\ \mathcal{A} \vdash F \wedge G & \text{Tau}(\alpha,\beta) \end{cases}$$

E∨1:

$$(\delta)\ \mathcal{A} \vdash F \vee G\ \pi \longrightarrow \begin{cases} (\alpha)\ \mathcal{A},\neg F \vdash \neg F & \text{Hyp} \\ (\beta)\ \mathcal{A},\neg F \vdash G & \pi' \\ (\gamma)\ \mathcal{A} \vdash \neg F \Rightarrow G & \text{Ded}(\beta) \\ (\delta)\ \mathcal{A} \vdash F \vee G & \text{Tau}(\gamma) \end{cases}$$

E∨2:

$(\delta)\ \mathcal{A} \vdash F \vee G\ \ \pi \longrightarrow$

(α)	$\mathcal{A}, \neg G$	$\vdash$	$\neg G$	Hyp
(β)	$\mathcal{A}, \neg G$	$\vdash$	F	π'
(γ)	$\mathcal{A}$	$\vdash$	$\neg G \Rightarrow F$	Ded(β)
(δ)	$\mathcal{A}$	$\vdash$	$F \vee G$	Tau(γ)

E∨3:

$(\beta)\ \mathcal{A} \vdash F \vee G\ \ \pi \longrightarrow$

(α)	$\mathcal{A}$	$\vdash$	F	π'
(β)	$\mathcal{A}$	$\vdash$	$F \vee G$	Tau(α)

E∨4:

$(\beta)\ \mathcal{A} \vdash F \vee G\ \ \pi \longrightarrow$

(α)	$\mathcal{A}$	$\vdash$	G	π'
(β)	$\mathcal{A}$	$\vdash$	$F \vee G$	Tau(α)

E¬:

$(\beta)\ \mathcal{A} \vdash \neg F\ \ \pi \longrightarrow$

(α)	$\mathcal{A}$	$\vdash$	G	π'
(β)	$\mathcal{A}$	$\vdash$	$\neg F$	Rule $\Re$

provided that F and G are one of the pairs (¬A, A), (A∧B, ¬A∨¬B), (A∨B, ¬A∧¬B), (A⇒B, A∧¬B), (∀xA, ∃x¬A), (∃xA, ∀x¬A), and Rule $\Re$ is the appropriate rule of inference.

E∀:

$(\beta)\ \mathcal{A} \vdash \forall x Fx\ \ \pi \longrightarrow$

Let c be an arbitrary object				
(α)	$\mathcal{A}$	$\vdash$	Fc	π'
(β)	$\mathcal{A}$	$\vdash$	$\forall x Fx$	$\forall G(\alpha)$

E∃:

$(\beta)\ \mathcal{A} \vdash \exists x Fx\ \ \pi \longrightarrow$

(α)	$\mathcal{A}$	$\vdash$	Ft	π'
(β)	$\mathcal{A}$	$\vdash$	$\exists x Fx$	$\exists G(\alpha)$

E-Divide:

$(\zeta)\ \mathcal{A} \vdash F\ \ \pi \longrightarrow$

(α)	$\mathcal{A}$	$\vdash$	$G \vee \neg G$	Tau
We consider separately the cases of (α)				
Case 1:				
(β)	$\mathcal{A}, G$	$\vdash$	G	Hyp
(γ)	$\mathcal{A}, G$	$\vdash$	F	π_1
Case 2:				
(δ)	$\mathcal{A}, \neg G$	$\vdash$	$\neg G$	Hyp
(ε)	$\mathcal{A}, \neg G$	$\vdash$	F	π_2
End of cases (1, 2) of (α)				
(ζ)	$\mathcal{A}$	$\vdash$	F	Cas($\alpha,\gamma,\varepsilon$)

E⊥:

$(\gamma)\ \mathcal{A} \vdash F\ \ \pi \longrightarrow$

(α)	$\mathcal{A}, \neg F$	$\vdash$	$\neg F$	Hyp
(β)	$\mathcal{A}, \neg F$	$\vdash$	$\bot$	π'
(γ)	$\mathcal{A}$	$\vdash$	F	IP(β)

All these external rules fall into one of two categories. E⇒, E∧, E–, and E∀ are called automatic, because no further information is needed for their application. The four variations of the rule E∨, as well as E∃, and E⊥ may only be applied with the permission of the user or with additional information, for example if the term t in the case of E∃ is explicitely given. These external rules are therefore called user-guided.

4.2.2 Mixed Rules

M-Cases:

$$\left.\begin{array}{llll}(\alpha) & \mathcal{A} \vdash & F \vee G & \text{Rule } \mathfrak{R} \\ (\zeta) & \mathcal{A} \vdash & H & \pi\end{array}\right\} \longrightarrow$$

$$\left\{\begin{array}{llll}(\alpha) & \mathcal{A} \vdash & F \vee G & \text{Rule } \mathfrak{R} \\ \text{We consider separately the cases of } (\alpha) & & & \\ \underline{\text{Case 1:}} & & & \\ (\beta) & \mathcal{A}, F \vdash & F & \text{Hyp} \\ (\gamma) & \mathcal{A}, F \vdash & H & \pi_1 \\ \underline{\text{Case 2:}} & & & \\ (\delta) & \mathcal{A}, G \vdash & G & \text{Hyp} \\ (\varepsilon) & \mathcal{A}, G \vdash & H & \pi_2 \\ \text{End of cases (1, 2) of } (\alpha) & & & \\ (\zeta) & \mathcal{A} \vdash & H & \text{Cas}(\alpha,\gamma,\varepsilon)\end{array}\right.$$

M-Unless:

$$\left.\begin{array}{llll}(\alpha) & \mathcal{A} \vdash & F \vee G & \text{Rule } \mathfrak{R} \\ (\zeta) & \mathcal{A} \vdash & G & \pi\end{array}\right\} \longrightarrow$$

$$\left\{\begin{array}{llll}(\alpha) & \mathcal{A} \vdash & F \vee G & \text{Rule } \mathfrak{R} \\ (\beta) & \mathcal{A}, F \vdash & F & \text{Hyp} \\ (\gamma) & \mathcal{A}, F \vdash & G & \pi' \\ (\varepsilon) & \mathcal{A} \vdash & F \Rightarrow G & \text{Ded}(\gamma) \\ (\zeta) & \mathcal{A} \vdash & G & \text{Tau}(\alpha,\varepsilon)\end{array}\right.$$

M-Choose:

$$\left.\begin{array}{llll}(\alpha) & \mathcal{A} \vdash & \exists x F x & \text{Rule } \mathfrak{R} \\ (\delta) & \mathcal{A} \vdash & G & \pi\end{array}\right\} \longrightarrow$$

$$\left\{\begin{array}{llll}(\alpha) & \mathcal{A} \vdash & \exists x F x & \text{Rule } \mathfrak{R} \\ (\beta) & \mathcal{A}, Fc \vdash & Fc & \text{Hyp} \\ & & & \\ (\gamma) & \mathcal{A}, Fc \vdash & G & \pi' \\ (\delta) & \mathcal{A} \vdash & G & \text{Sel}(\alpha,\gamma)\end{array}\right.$$

c may not occur free in $\mathcal{A}$, F, or G.

M-Inf:

$$\left.\begin{array}{llll}(\alpha) & \mathcal{A} & \vdash F \Rightarrow G & \text{Rule } \mathfrak{R} \\ (\gamma) & \mathcal{A} & \vdash G & \pi\end{array}\right\} \longrightarrow \left\{\begin{array}{llll}(\alpha) & \mathcal{A} & \vdash F \Rightarrow G & \text{Rule } \mathfrak{R} \\ (\beta) & \mathcal{A} & \vdash F & \pi' \\ (\gamma) & \mathcal{A} & \vdash G & \text{Tau}(\alpha,\beta)\end{array}\right.$$

4.2.3 Internal Rules

All of these deducing rules use internal lines and add a new internal line to the generalized proof. Here the new lines are marked with an arrow, "→", and written below their parent lines.

I–:

$$\begin{array}{llll} & (\alpha) & \mathcal{A} \vdash -F & \text{Rule } \mathfrak{R} \\ \rightarrow & (\beta) & \mathcal{A} \vdash G & \text{Neg}(\alpha)\end{array}$$

provided that F and G are one of the pairs ($\forall$xA, $\exists$x–A) and ($\exists$xA, $\forall$x–A).

I$\forall$:

$$\begin{array}{llll} & (\alpha) & \mathcal{A} \vdash \forall x F x & \text{Rule } \mathfrak{R} \\ \rightarrow & (\beta) & \mathcal{A} \vdash Ft & \forall I(\alpha)\end{array}$$

for an arbitrary term t, that can be inserted for x.

I-Tau:

$$\begin{array}{llll} & (\alpha_1) & \mathcal{A}_1 \vdash F_1 & \text{Rule } X_1 \\ & (\alpha_2) & \mathcal{A}_2 \vdash F_2 & \text{Rule } X_2 \\ & & \cdots & \\ & (\alpha_n) & \mathcal{A}_n \vdash F_n & \text{Rule } X_n \\ \rightarrow & (\beta) & \bigcup \mathcal{A}_i \vdash F & \text{Tau}(\alpha_1,\ldots,\alpha_n)\end{array}$$

provided that F is a consequence of F_1 through F_n in propositional logic.

For practical purposes this rule must be further divided into smaller rules. Three types are possible, namely analytic rules, breaking up formulas, synthetic rules, constructing formulas from others, and finally converting rules, that change a formula to an equivalent form.

a) analytic propositional rules:

Ia∧1:		(α)	$\mathcal{A}$	⊢	F ∧ G	Rule ℜ
	→	(β)	$\mathcal{A}$	⊢	F	Tau(α)
Ia∧2:		(α)	$\mathcal{A}$	⊢	F ∧ G	Rule ℜ
	→	(β)	$\mathcal{A}$	⊢	G	Tau(α)
Ia—:		(α)	$\mathcal{A}$	⊢	–(–F)	Rule ℜ
	→	(β)	$\mathcal{A}$	⊢	F	Tau(α)

b) synthetic propositional rules:

Is∧:		(α)	$\mathcal{A}_1$	⊢	F	Rule X_1
		(β)	$\mathcal{A}_2$	⊢	G	Rule X_2
	→	(γ)	$\mathcal{A}_1,\mathcal{A}_2$	⊢	F ∧ G	Tau(α, β)
Is∨1:		(α)	$\mathcal{A}$	⊢	F	Rule ℜ
	→	(β)	$\mathcal{A}$	⊢	F ∨ G	Tau(α)
Is∨2:		(α)	$\mathcal{A}$	⊢	G	Rule ℜ
	→	(β)	$\mathcal{A}$	⊢	F ∨ G	Tau(α)
Is⇒:		(α)	$\mathcal{A}$	⊢	G	Rule ℜ
	→	(β)	$\mathcal{A}$	⊢	F ⇒ G	Tau(α)
Is—:		(α)	$\mathcal{A}$	⊢	F	Rule ℜ
	→	(β)	$\mathcal{A}$	⊢	–(–F)	Tau(α)

c) converting propositional rules:

Ic–∧:		(α)	$\mathcal{A}$	⊢	– (F ∧ G)	Rule $\Re$
	→	(β)	$\mathcal{A}$	⊢	–F ∨ –G	Tau(α)
Ic–∨:		(α)	$\mathcal{A}$	⊢	– (F ∨ G)	Rule $\Re$
	→	(β)	$\mathcal{A}$	⊢	–F ∧ –G	Tau(α)
Ic–⇒:		(α)	$\mathcal{A}$	⊢	– (F ⇒ G)	Rule $\Re$
	→	(β)	$\mathcal{A}$	⊢	F ∧ –G	Tau(α)
Ic⇒:		(α)	$\mathcal{A}$	⊢	F ⇒ G	Rule $\Re$
	→	(β)	$\mathcal{A}$	⊢	–F ∨ G	Tau(α)
Ic∨1:		(α)	$\mathcal{A}$	⊢	F ∨ G	Rule $\Re$
	→	(β)	$\mathcal{A}$	⊢	–F ⇒ G	Tau(α)
Ic∨2:		(α)	$\mathcal{A}$	⊢	F ∨ G	Rule $\Re$
	→	(β)	$\mathcal{A}$	⊢	–G ⇒ F	Tau(α)
Ic∨∧:		(α)	$\mathcal{A}$	⊢	E∨(F∧G)	Rule $\Re$
	→	(β)	$\mathcal{A}$	⊢	(E∨F)∧(E∨G)	Tau(α)
Ic-∧∨:		(α)	$\mathcal{A}$	⊢	E∧(F∨G)	Rule $\Re$
	→	(β)	$\mathcal{A}$	⊢	(E∧F)∨(E∧G)	Tau(α)

4.3 A Semiautomatic Proof System

The set of transformation rules defined in the previous section constitutes a proof system for natural deduction proofs. This means that for any valid formula F, there is a finite sequence of GNDPs starting with $\vdash F$ and ending with an NDP for F. Every element in this sequence follows from its predecessor by application of one of the transition rules (Completeness of the set of transition rules).

This system could be used as a proof checker, the user choosing from a menu of applicable rules, and the system correctly applying them. But the system can actually do a lot more by preselecting the trafo rules and giving the user a much smaller choice of rules.

Starting from the trivial GNDP, external rules are applied – if necessary with the help of the user – until all of the conclusion formulae of external lines are either literals or suitable subformulas of internal conclusions. In this way it is avoided, always to break down the formula to prove into its literals and build it up again later. If a complex formula is contained as a whole in some axiom, then it can be hoped, that a shorter proof is available. This is detected by using the notion of axiomatic formulae defined below.

Then mixed rules are used, and only if no more external or mixed rule can be applied, the system starts using internal rules.

4.3-1 Definition:

In the context of (generalized) natural deduction proofs, axiomatic formulas are defined as follows, where we distinguish between strongly axiomatic and weakly axiomatic:

(a) Any conclusion F of an internal line is a strongly axiomatic formula.

(b) Any strongly axiomatic formula is also weakly axiomatic.

(c) If F is a strongly (weakly) axiomatic formula, then with

$F = A \wedge B$ A and B are strongly (weakly) axiomatic,

$F = A \vee B$ A and B are weakly axiomatic,

F = $A \Rightarrow B$ B is weakly axiomatic, strongly , if a proof for A is known and F is strongly axiomatic,

F = $-(-A)$ A is strongly (weakly) axiomatic,

F = $\forall x\ Ax$ At becomes strongly (weakly) axiomatic, if it is known, that t must be inserted for x during the proof.

F = $\exists xFx$ Fc becomes strongly (weakly) axiomatic, if it is known, that c is inserted for x during the proof.

F = $-(A \wedge B)$, $-(A \vee B)$, $-(A \Rightarrow B)$, or $-\exists xAx$, then $-A \vee -B$, $-A \wedge -B$, $A \wedge -B$, or $\forall x-Ax$ are strongly (weakly) axiomatic, respectively.

In the following, for each of the external lines in a GNDP there are sets of strongly and weakly axiomatic formulas. E-Rules must not be applied, if the conclusion of the external line is among its strongly axiomatic formulas.

The strategy for a semiautomatic proof system is the following:

4.3-2 Algorithm:

1. Start with GNDP = $\varnothing \vdash F$.
   Initialize strongly and weakly axiomatic formulas for this external line as empty sets.

2. Apply automatic E-Rules as long as possible, until all of the external lines conclude in weakly axiomatic formulas. Whenever new internal lines are added, their conclusions become strongly axiomatic formulas for the external line in case. As always the complete sets of axiomatic formulas are computed.

3. If possible, ask the user, whether one of the other user-guided external rules E$\vee$, E$\exists$, or E-Divide should be applied. If so, do it and go to 2.

4. Now apply mixed rules at the user's discretion until no longer possible. After every application the sets of axiomatic formulas have to be updated. If M-Inf was applied, go to 2.

5. Check, whether the proof is already completed. Then return GNDP as final proof.

6. Reluctantly apply E⊥, if the user really insists, then update the sets of axiomatic formulas.

7. Let the user choose, which of the internal rules shall be applied. Then go to 5.

Of course in our actual implementation, "the user" may be the computer itself selecting according to appropriate heuristics by making use of the information in a refutation graph which was previously computed.

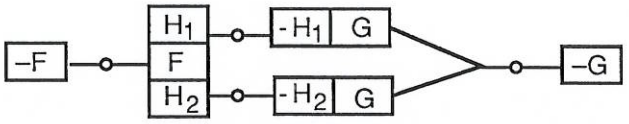
4.4 Using a Refutation Graph to construct an NDP

In this chapter we show, how a proof represented as a refutation graph can guide the "search" for a natural deduction proof. In this context, search does not mean to find an original proof, but to transform the given, graph represented proof into the natural deduction calculus.

To that end, we use the above rule system and gradually change generalized NDPs in order to complete the natural deduction proof. To achieve this, all the tasks formerly done by the user have to be taken over. This includes the choice between several applicable transformation rules and the update of the deduction graph during the whole process. The information about the automatically generated proof consists of a refutation graph, a ground substitution for all the formulae needed, which includes the information about skolemization and duplication of clauses, and a relation Δ between the literal nodes of the refutation graph and the atom occurrences of the input formulae, indicating where the literal nodes stem from.

In this section, external rules will only be considered applicable, if their conclusions are not axiomatic.The first point of choice in algorithm 4.3-2 comes up, when rule E∨ is applicable. In this case one of its four versions can always be used, so the graph is needed to decide which one. Whenever E∨3 or E∨4 is necessary, that is when one part of the disjunction is directly provable, then the other part does not appear in the refutation graph, which is a property easy to check. The choice between E∨1 and E∨2 is only a stylistic one; in this case there are always two clause nodes in the graph representing the two parts of the disjunction, so the problem appears to be symmetric. In fact both of E∨1and E∨2 always work. But the structure of the refutation graph may help to make a good choice as can be seen in the following example:

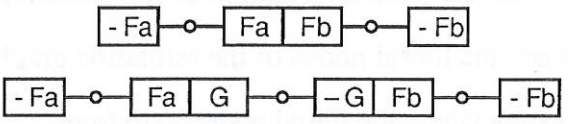
4.4-1 Example Graph:



A good heuristic is to choose the one with the smaller number of literal nodes in its shore, –F in this example, as an additional assumption. In this way the rest of the proof can be divided into two independent parts, while choosing –G would lead to a subsequent application of the Rule of Cases. This heuristic can also be generalized to cases where the formulae F and G are not literals.

When E∃ can be applied, one has to consult the total unifier of the refutation graph. If the variable has only been instantiated once, that is no copy of the formula has been needed, then the rule can immediately be set to work. If copies have been made, one way to continue is always to construct a proof by contradiction starting with an application of E⊥. But there are certain cases, when this is not necessary.

4.4-2 Example Graphs:



In the upper graph the proof can be done in two cases with Fa and Fb as assumptions, while in the lower one E-Divide can be applied for G and –G and so the refutation graph is cut in two

parts. Another possibility in the second case is to derive FavFb first, and continue as in case one.

More complex situations arise, when a decision about mixed rules has to made. After all, it is certainly not useful to divide the proof into cases, whenever a disjunction has been derived. And we have seen earlier, that some of these rules may lead into dead ends when applied incautiously. On the other hand none of these rules is indispensable, for one can always fall back on a proof by contradiction. This is not desired, however, and large classes of graphs have to be found where an application of such rules is safe and leads to nice proofs.

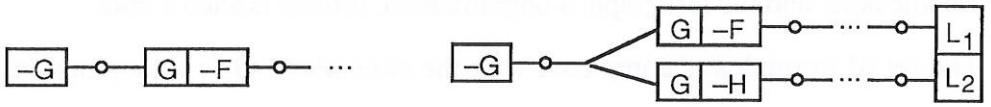
In the following section we use the rule M-Inf as an example to show how certain structural properties of the refutation graph can be sufficient to guarantee the safety of the application of such rules. The rule M-Inf is repeated below as a reminder:

M-Inf:

$$\left.\begin{array}{llll}(\alpha) & \mathcal{A} \vdash F \Rightarrow G & \text{Rule X} \\ (\gamma) & \mathcal{A} \vdash G & \pi\end{array}\right\} \longrightarrow \left\{\begin{array}{llll}(\alpha) & \mathcal{A} \vdash F \Rightarrow G & \text{Rule X} \\ (\beta) & \mathcal{A} \vdash F & \pi' \\ (\gamma) & \mathcal{A} \vdash G & \text{Tau}(\alpha,\beta)\end{array}\right.$$

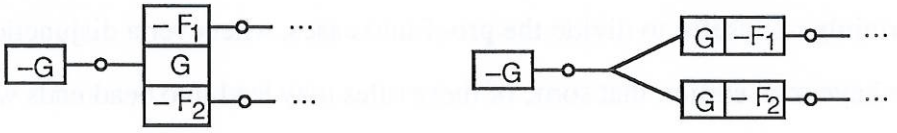
Let as assume for the moment, that we have to decide about the application of M-Inf. The danger is to apply it, when F is not provable. So we have to make sure that F is valid, and this can be seen from the graph, provided the automatically found proof has used the formula F⇒G. The simplest case is, when the formulae F and G involved are both literals.

4.4-3 Example Graphs:



Whenever the graph has the first structure, the rule may be applied. Should the clause [–F G] not appear in the graph, the rule can obviously not be applied, but there are also situations, where [–F G] is part of the graph, but F is still not derivable, as in the second example graph. In this case one could for instance break up the proof into cases with assumptions L_1 and L_2.

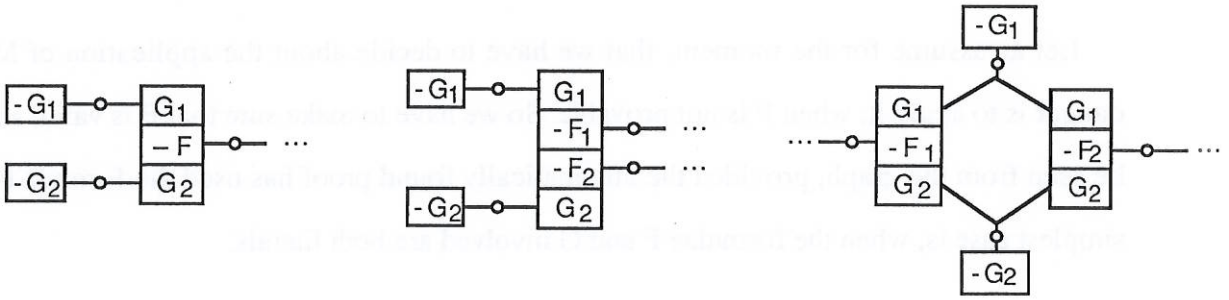
4.4-4 Example Graphs:



The two refutation graphs above cover the cases, where F is a conjunction or disjunction. In the first case there must not be a trail between $-F_1$ and $-F_2$, since then the graph would be cyclic, and therefore no refutation graph. If in the second case there were no such a trail, then the refutation graph would not be minimal, since any one of the branches could be omitted. This leads to a situation, where either F_1 or F_2 can be derived independently as in E∨3 or E∨4. If, on the other hand, there is such a trail, then E∨1 or E∨2 can be applied after M-Inf.

When $G=G_1 \vee G_2$ is a disjunction and F either literal, conjunction, or disjunction, then the following refutation graphs allow an application of M-Inf:

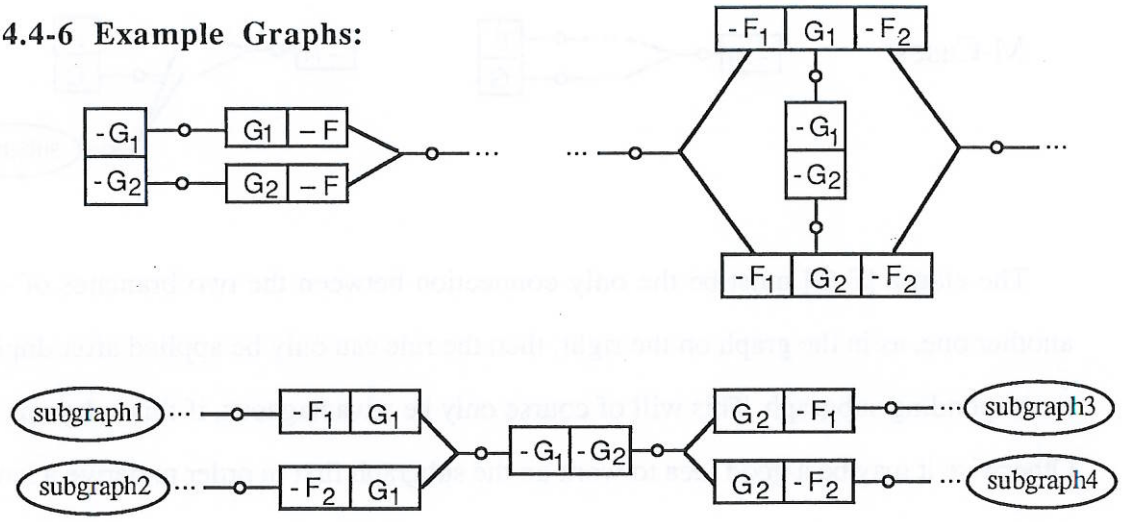
4.4-5 Example Graphs:



As in the first case above there must be no trail between –F1 and –F2 in the refutation graph in the middle here; and the last graph is only minimal, if there is such a trail.

The set of examples is completed with the case where G is a conjunction. Again F may be either literal, conjunction, or disjunction.

4.4-6 Example Graphs:



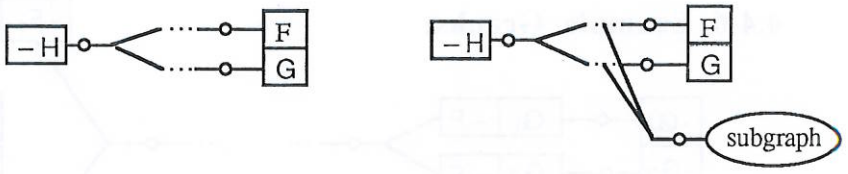
When F is a conjunction, there are two essentially different possible graphs. They correspond to the cases of the subsequent applications of different versions of rule E∨. The first one – subsequent application of E∨3 or E∨4 – is of course isomorphic to the case, where F is a literal. The second case is illustrated above; for this graph to become a refutation graph, trails must exist both between subgraph1, subgraph2 and subgraph3, subgraph4. Otherwise the graph would be either cyclic or not minimal.

The cases where F or G are implications can always be seen as one of the disjunction cases above . And also when F or G are quantified formulae, there is no essential difference.

So rule M-Inf can be applied with advantage when the graph has the form described above, it must not be used, when the clause [–F G] does not appear in the graph at all. The remaining negative cases can be summed up by the condition, that G may not be present in the refutation graph in several copies. The negative example above is really only a special case of this, because the graph could be rewritten using two copies of G. But not all is lost in this case; most of the time the rule M-Inf can then be used to prove one part after division of the proof into cases.

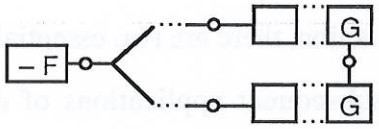
Similarly graph situations can be given for the application of the rules M-Cases, E-Divide, and M-Unless. The cases below cover only the case, where F, G, and H are literals, but they can be generalized just as those for rule M-Inf.

M-Cases:



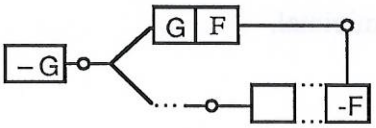
The clause [F G] must be the only connection between the two branches of –H. If there is another one, as in the graph on the right, then the rule can only be applied after duplication of the corresponding subgraph. This will of course only be advantageous, if this subgraph is very small. Otherwise it may be a good idea to work an the subgraph first in order to derive a lemma.

E-Divide:



The link between G and –G must separate the two branches leading to –F. This rule is actually only a variation of the rule M-Cases, it is not usually applied unless especially called for by a heuristic.

M-Unless:



This is also a special case of M-Cases, where one of the cases is trivial. Here a cut through F and G in [F G] leads to the desired linearization of the proof.

In the rest of this chapter, the process of updating the refutation graph after each application of a rule is described. There are essentially four tasks to be done:

1. Updating the total substitution after using an instantiation.
2. Updating the sets of axiomatic formulae for the external line currently worked with.
3. Updating the sets of negatively polarized literals in the graph, i.e. those literals that were constructed from the falsified parts of the theorem to prove.
4. Removing parts of the graph or dividing the graph into parts.

For all these tasks we will give examples.

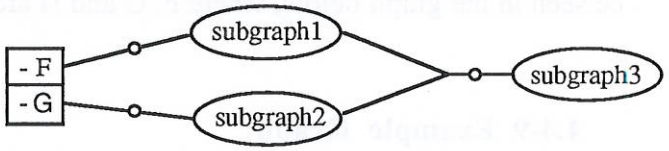
The substitution has to be altered, whenever one of its components has been used. This happens for instance in the case of rule E∃, when the component x↦t can be removed from the substitution.

The rule E⇒ is an example, where the set of axiomatic formulae is changed. The formula F and its axiomatic closure is added to the set of axiomatic formulas.

When E∨1 is applied, then the formerly negated literal –F is considered an axiom now, which means that it may be positively used in deduction steps.

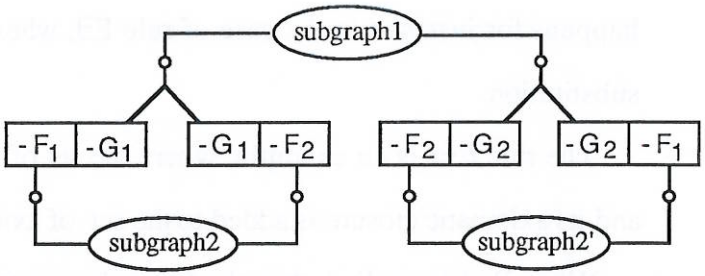
The most difficult part of the updating process is the division of the graph, when the proof can be done in two independent parts. This is the case after application of E∧, E-Divide, or M-Cases. With literals F and G a graph might have the following form before application of E∧:

4.4-7 Example Graph:



The refutation graph must now be split into two parts. This is done by dividing the clause [–F –G] into two clauses [–F] and [–G]. The remaining graph is obviously still a refutation graph, since no literal becomes pure and no cycle can be introduced, but the graph is no longer minimal. It can be seen, that its two components are independent refutation graphs for F and G. If these components have a non-empty intersection (subgraph3), then this subgraph must be duplicated. This is not desirable, however, when it is too large. In this case the subgraph has to be worked with first in order to generate a lemma, which can later be used in both the proofs of F and G.

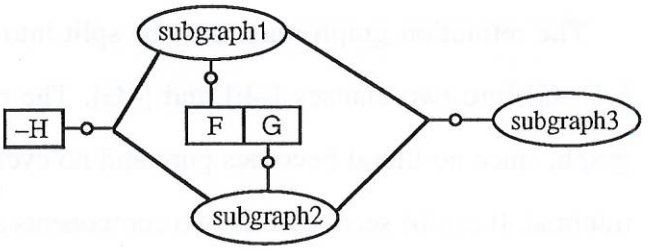
4.4-8 Example Graph:



The structure of the graph becomes more complex, when F and G are disjunctions (see above). The cut must now be fourfold, separating –F from –G in four different clauses. In this case one has a choice whether to adopt subgraph1 or subgraph2. Again any two subgraphs may intersect.

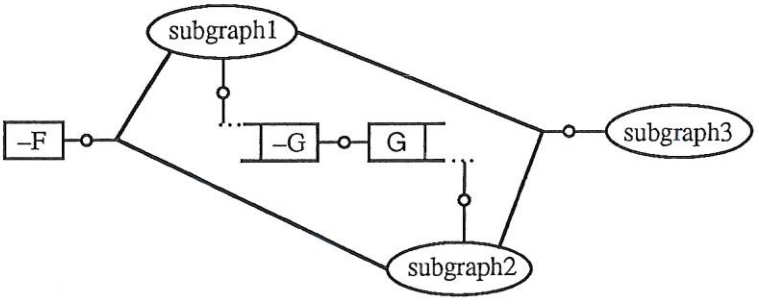
The structures of the refutation graphs for the rule M-Cases is „dual" to those for E∧. This can be seen in the graph below, where F, G and H are assumed to be literals.

4.4-9 Example Graph:



Again it is cut through [F G], and the resulting two graph components are both proofs for H, assuming F or G as an additional assumption.

4.4-10 Example Graph:



In this case the cut must be through the two clauses containing G and –G. These literals are therfore duplicated in the process. Actually this rule corresponds to an application of a resolution step with G and –G as complementary literals and a subsequent division of the proof into cases by breaking up the resolvent.

The problem of selecting the correct internal rules remains to be dealt with. When all the external formulae are axiomatic and none of the mixed rules can be applied, then the task is either to derive the external formulae by forward reasoning, using internal rules, or to derive new internal lines in order to apply mixed rules later.

In case of strongly axiomatic formulae one only has to apply I–, I∀, or analytic propositional rules in order to derive the desired formula. When a weakly axiomatic formula is to be proved, then one starts just as for strongly axiomatic formulae, but stops, when the subformula is reached, where further subformulae become only weakly axiomatic. This must be a disjunction, an implication, or an existentially quantified formula, containing the original formula as a subformula. Now the appropriate mixed rule must be applied and the process repeated.

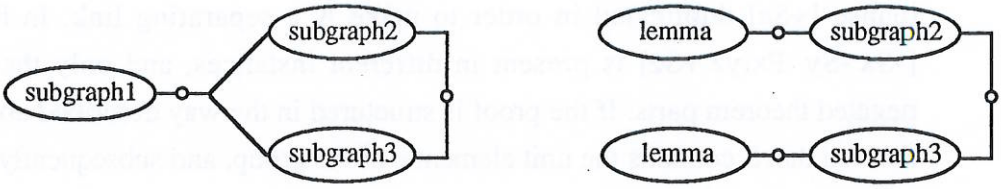
5. Outlook

The system described in the previous chapter represents a first step towards proofs in natural language. But on this basis alone it is not possible to perform an immediate translation of the resulting natural deduction proof into natural language. For one thing the proof has no internal structure, all the single proof steps, whether trivial or important, seem to have the same weight in the proof. Moreover, there is no easy way to determine which of the steps depend on one another and can therefore only be executed in a certain order, and which ones are totally independent. To solve the second problem one must construct a linearized version of the natural deduction proof, that is one where an exact order of the single steps is given. And the natural way to solve the first problem is to formulate subgoals or lemmas within the proof, that can be derived seperately and will then later be used in the proof. In the rest of this chapter we will describe some ideas that might lead to better answers to these two questions of proof transformation.

There are three main reasons for a mathematician to formulate subgoals or lemmas in a proof. A lemma is generated, whenever a certain formula, that must be derived in a number of proof steps from the axioms, is later used several times. A lemma is also useful, when more than one derived formula is used in a single proof step, in this case all but one of these formulae are derived as lemmas before the main chain of reasoning is followed. The last case comes up in relatively long proofs, that can be divided into several independent non-trivial parts. These parts will then be considered as subgoals, especially when the formulae in case formulate interesting propositions.

It seems to be possible to detect some of these situations by using the information hidden in the refutation graph and the natural deduction proof. A schematic graph for the case, where a lemma is used more than once is given on the next page.

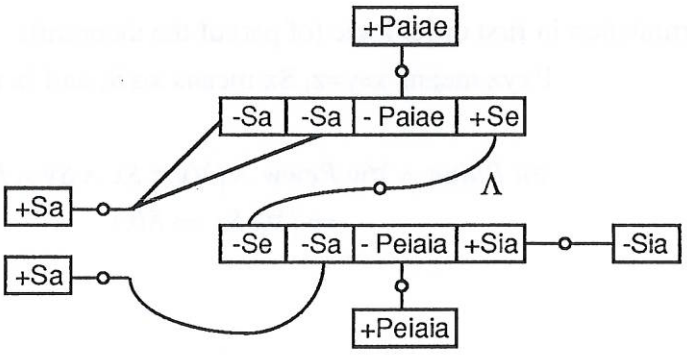
Example 5-1:



Of course this can only work, if subgraph 1 contains no part of the negated theorem; and it makes only sense, if the lemma represents a sufficiently small formula compared to subgraph 1. The way to find similar situations is by searching for (sets of) links separating the refutation graph such that one part is free of negated theorem parts. This subgraph will then be a deduction graph for the lemma formula.

The case of subgoals, that is lemmas which are only used once, differs only slightly from the above case. Again a separating link or a set of separating links have to be found, such that all the negated theorem parts are contained in the same subgraph. But now, to make the formulation of a subgoal worthwile, it is important that the lemma should represent an important step in the proof. As a rule, both partial proofs should be "easier" than the original theorem. In fact, mathematicians often use "interesting propositions" as lemmas in such cases, but it seems hard to find those using purely syntactic information. In [Da81] Martin Davis proposes that complicated inferences require multiple substitutions in the same clause. In this sense one might argue that a subgoal makes the proof more obvious, when the different parts of the refuatation graph contain different copies of the same clause. This idea is illustrated in the example below.

Example 5-2:



This is the refutation graph of examples 3.1-10 and 4.1-2 (subgroup criterion), with the unit clause [+Sa] duplicated in order to make Λ a separating link. In both subgraphs the clause [-Sx -Sy -Pxiyz +Sz] is present in different instances, and only the lower subgraph contains negated theorem parts. If the proof is structured in the way described above, then in a first step it is derived that S contains the unit element e of the group, and subsequently that for every element x in S, its inverse is also in S.

For the linearization of natural deduction proofs Daniel Chester has formulated a procedure in [Ch76]. He uses dependency graphs, where the nodes correspond to (sets of) proof lines of the natural deduction proof and the edges represent the interdependencies between these lines. In this representation it is easier to group sets of proof lines belonging to one another, and it is also possible to formulate lemmas, whenever a derived formula depends on too many different formulae previously derived.

All these considerations are only very briefly stated in this outlook, and the following example shall now show the complete process from a formulation of a theorem in mathematical language to a proof also readable by a mathematician.

Example 5-3: (subgroup criterion)

1. Informal (mathematical) representation of the theorem:

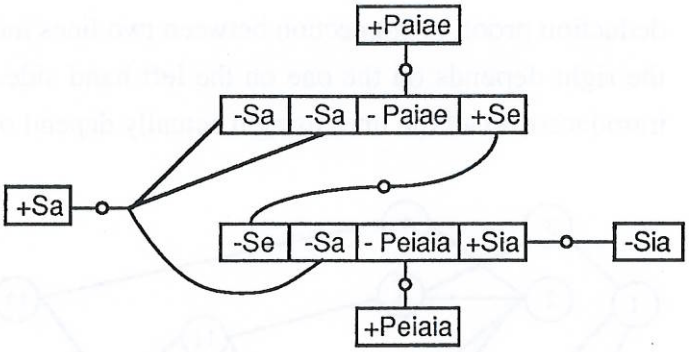
   Let G be a group, $S \subseteq G$; if for all x,y in S, $x \circ y^{-1}$ is also in S,
   then for every x in S its inverse is also in S.

2. Formulation in first order logic (of part of the theorem):

   Pxyz means $x \circ y = z$, Sx means $x \in S$, and ix means x^{-1}, the inverse of x.

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

3. Proof of the theorem as a refutation graph:

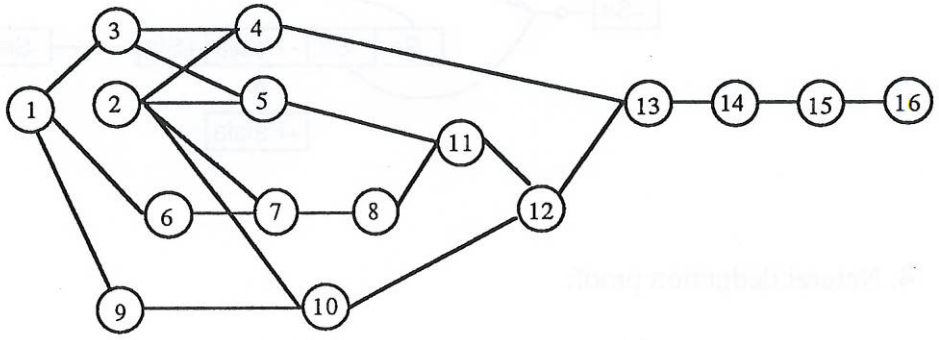


4. Natural deduction proof:

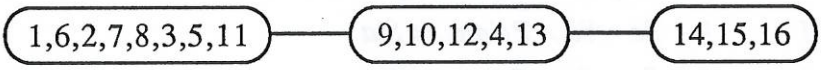
(1) 1	$\vdash$	$(\forall u Puiue) \wedge (\forall w Peww) \wedge (\forall xyz\ Sx \wedge Sy \wedge Pxiyz \Rightarrow Sz)$	*Hyp*
Let a be an arbitrary constant			
(2) 2	$\vdash$	Sa	*Hyp*
(3) 1	$\vdash$	$\forall xyz\ Sx \wedge Sy \wedge Pxiyz \Rightarrow Sz$	*Tau(1)*
(4) 1	$\vdash$	$Se \wedge Sa \wedge Peiaia \Rightarrow Sia$	*∀I(3)*
(5) 1	$\vdash$	$Sa \wedge Sa \wedge Paiae \Rightarrow Se$	*∀I(3)*
(6) 1	$\vdash$	$\forall u\ Puiue$	*Tau(1)*
(7) 1	$\vdash$	$Paiae$	*∀I(6)*
(8) 1, 2	$\vdash$	$Sa \wedge Sa \wedge Paiae$	*Tau(2,7)*
(9) 1	$\vdash$	$\forall w\ Peww$	*Tau(1)*
(10) 1	$\vdash$	$Peiaia$	*∀I(9)*
(11) 1	$\vdash$	Se	*Tau(5,8)*
(12) 1	$\vdash$	$Se \wedge Sa \wedge Peiaia$	*Tau(2,10,11)*
(13) 1, 2	$\vdash$	Sia	*Tau(4,12)*
(14) 1	$\vdash$	$Sa \Rightarrow Sia$	*Ded(13)*
(15) 1	$\vdash$	$\forall x\ Sx \Rightarrow Six$	*∀G(14)*
(16)	$\vdash$	$(\forall u Puiue) \wedge (\forall w Peww) \wedge (\forall xyz\ Sx \wedge Sy \wedge Pxiyz \Rightarrow Sz) \Rightarrow (\forall x\ Sx \Rightarrow Six)$	*Ded(15)*

5. Dependency graph:

This graph represents the dependency relation between lines of the natural deduction proof. A connection between two lines indicates that the line further to the right depends on the one on the left hand side. In line 2 a new constant is introduced, so all the lines using it actually depend on line 2.



6. Linearized proof schema:



7. Natural language:

For this formulation, the proof lines were "literally translated" into natural language. The numbers in brackets indicate, where the proposition comes from. Trivial lines, only building up conjunctions from previously shown facts, were omitted, and the fact $e \in S$ was used as a subgoal.

In a group $u \circ u^{-1}=e$ holds for all u [6]. Let a be an arbitrary element of $S \subseteq G$ [2], then $a \circ a^{-1}=e$ [7]. Since it is given, that for all $x,y \in S$, also $x \circ y^{-1} \in S$ [3], this also holds for $x=y=a$ [5], hence $a \circ a^{-1}=e \in S$ [11].
Now $e \circ w=w$ for all w [9], therefore $e \circ a^{-1}=a^{-1}$ [10]. With $x=e$ and $y=a$ [4], both elements of S, also $e \circ a^{-1}=a^{-1} \in S$ [13].
Now $a^{-1} \in S$, if $a \in S$ [14]. Since a was arbitrarily chosen, this holds for all $x \in S$ [15], which proves the theorem [16].

6. Literature

[An80] P. B. Andrews
Transforming Matings into Natural Deduction Proofs
Lecture Notes in Comp. Sci. 87, pp. 281-292 (1980)

[Che76] D. Chester
The Translation of Formal Proofs into English
AI 7, pp. 261-278 (1976)

[Da81] M. Davis
Obvious Logical Inferences
Proc. of the 7th IJCAI, Vancouver, 1981

[Ei87] N. Eisinger
Completeness, Confluence, and Related
Problems of Clause Graph Resolution
Dissertation, Universität Kaiserslautern, 1987

[Ge35] G. Gentzen
Untersuchungen über das logische Schließen I
Math. Zeitschrift 39, pp. 176-210 (1935)

[Ko75] R. Kowalski
A Proof Procedure Using Connection Graphs
JACM, vol. 22, No. 4, 1975, pp. 572-595

[Lo78] D. W. Loveland
Automated Theorem Proving: A Logical Basis
North Holland, 1978

[MKRP84] Karl Mark G Raph
The Markgraf Karl Refutation Procedure
Memo-SEKI-Mk-84-01, Universität Kaiserslautern, 1984

[Prä85] A. Präcklein
Ein Reduktionsmodul für einen Automatischen Beweiser
Diplomarbeit, Universität Karlsruhe, 1985

[Pr65] D. Prawitz
Natural Deduction, A Proof-Theoretical Study
Almqvist & Wiksell, Stockholm, 1965

[Sh79] R. E. Shostak
A Graph-Theoretic View of Resolution Theorem-Proving
Report SRI International, Menlo Park, CA 1979