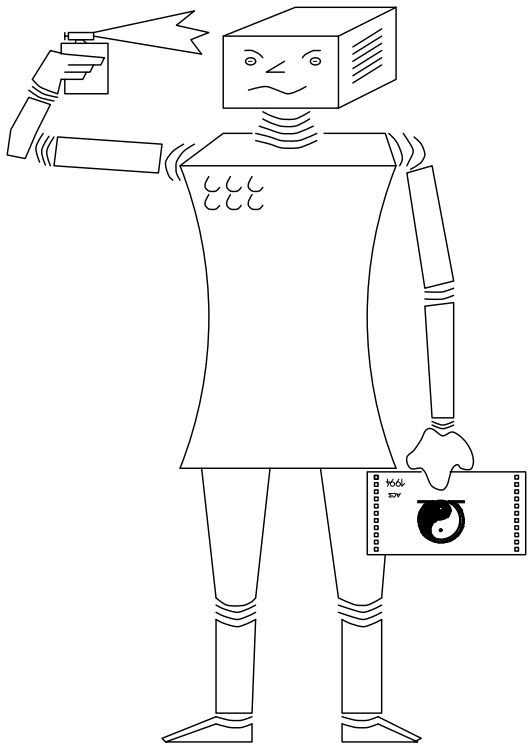

Translating Machine-Generated Resolution Proofs into ND-Proofs at the Assertion Level

Xiaorong Huang

Published as: Proc. of 4th Pacific Rim Int. Conf. on AI, Springer, a slightly extended version

Translating Machine-Generated Resolution Proofs into ND-Proofs at the Assertion Level

Xiaorong Huang

Fachbereich Informatik, Universität des Saarlandes
Postfach 15 11 50, D-66041 Saarbrücken, Germany
e-mail: huang@cs.uni-sb.de

Abstract. Most automated theorem provers suffer from the problem that the resulting proofs are difficult to understand even for experienced mathematicians. An effective communication between the system and its users, however, is crucial for many applications, such as in a mathematical assistant system. Therefore, efforts have been made to transform machine generated proofs (e.g. resolution proofs) into natural deduction (ND) proofs. The state-of-the-art procedure of proof transformation follows basically its completeness proof: the premises and the conclusion are decomposed into unit literals, then the theorem is derived by multiple levels of proofs by contradiction. Indeterminism is introduced by heuristics that aim at the production of more elegant results. This indeterministic character entails not only a complex search, but also leads to unpredictable results.
In this paper we first study resolution proofs in terms of meaningful operations employed by human mathematicians, and thereby establish a correspondence between resolution proofs and ND proofs at a more abstract level. Concretely, we show that if its unit initial clauses are CNFs of literal premises of a problem, a unit resolution corresponds directly to a well-structured ND proof segment that mathematicians intuitively understand as the application of a definition or a theorem. The consequence is twofold: First it enhances our intuitive understanding of resolution proofs in terms of the vocabulary with which mathematicians talk about proofs. Second, the transformation process is now largely deterministic and therefore efficient. This determinism also guarantees the quality of resulting proofs.

1 Introduction

Most automated theorem provers suffer from the problem that they can produce proofs only in formalisms difficult to understand even for experienced mathematicians. In many applications, in particular if it is used as a mathematical assistant, it is crucial that the system and a user can communicate in an effective way. Only if a system also talks his language, a user will be convinced by machine-found proofs and feel his understanding of the topic improved. Since no current system can solve a wide range of challenging problems efficiently and the situation will not change in the near future, a user has to understand intermediate results in order to provide further guidance. Therefore, various efforts

have been made to *reconstruct* natural deduction (ND) proofs [Gen35] from such machine-generated proofs [And80, Mil83, Pfe87, Lin90, SK95].

Current procedures that transforms resolution proofs follow basically their completeness proof. Starting from a problem in a ND framework that contains the hypotheses and the conclusion, it first recursively decomposes the premises and the conclusion until literals are reached. Then the conclusion is proved by multiple levels of proofs by contradiction. To come up with more elegant proofs, this basic procedure was enriched with heuristics. Up to now there are only isolated heuristics that cover some specific proof structures, and they are often formulated as vague guide lines [Lin90, PN90]. When no heuristics are applicable, ND proofs thus constructed tend to be very awkward. Below is an artificial example to illustrate the worst case. The proof is encoded in the linearized version of natural deduction first used in [And80], which we will use throughout this paper. The numbers after the line number represent the logical hypotheses a line depends on. The inference rule that justifies a line is given after the conclusion formula, followed by the premise lines. Considering that this problem can be proved with one step of modus-ponens, the proof below is indeed awkward enough.

No	Hyp		Formula	Reason
1.	1	$\vdash$	A	(Hyp)
2.	2	$\vdash$	$A \Rightarrow B$	(Hyp)
3.	3	$\vdash$	$\neg B$	(Hyp)
4.	2	$\vdash$	$\neg A \vee B$	(Tau 2)
5.	5	$\vdash$	$\neg A$	(Hyp)
6.	1,5	$\vdash$	$\bot$	($\neg$E 1 5)
7.	7	$\vdash$	B	(Hyp)
8.	3,7	$\vdash$	$\bot$	($\neg$E 3 7)
9.	1,2,3	$\vdash$	$\bot$	(Case 4 6 8)
10.	1,2	$\vdash$	B	(Ind 9)

Not only the quality is not predicable, such heuristics introduces a complex search space. Take the proof above again as an example, the system must choose between modus-ponens and case analysis. While it is trivial in this case, these kind of decision involves a complex search in general. In some sense, the previous transformation procedures involve a search anew for a proof in the ND framework, utilizing some information of a proof found in another formalism. It is therefore not very surprising that the transform is sometimes more expensive than the original problem solving.

Another problem with the current approach is its target representation itself. Although each single step in an ND proof is easy to understand, the entire proof is usually at the level of a logic calculus and contain too many tedious steps. The resulting proofs are composed of derivations familiar from elementary logic, where the focus of attention is on syntactic manipulations rather than on the underlying semantic ideas. In contrast, informal proofs found in standard mathematical textbooks are primarily justified by applications of definitions or theorems. For instance, the derivation of $a \in F$ from $U \subset F$ and $a \in U$ is usually

justified by applying the definition of a subset encoded as

$$\forall S_1, S_2. S_1 \subset S_2 \Leftrightarrow \forall x. x \in S_1 \Rightarrow x \in S_2 \tag{1}$$

In [Hua94b], the author formalized the intuitive notion of the application of a definition or a theorem (collectively called *assertions*), as well as a procedure that substantially shortens ND proofs by abstracting them to the assertion level.

This paper attempts to show certain resolution proofs can be understood intuitively in the same way. Concretely, we will consider unit resolution proofs where the initial unit clauses are produced from literal hypotheses lines of the problem formulated in ND. We call them *SSPU*-resolutions (unit resolutions for simple-structured problems). Let us first examine the resolution proof below, obtained by restructuring a machine-found proof (compare Section 3.3). The numbering of the clauses are quite unnatural, but their meaning will become clear, once we show how they are derived from the original ones.

Example 1

The set of initial clauses:

$C1 = \{+(a * a^{-1} = e)\}$ $\quad$ $C2 = \{+(e * a^{-1} = a^{-1})\}$

$C3 = \{-(x \in S), -(y \in S), -(x * y^{-1} = z), +(z \in S)\}$

$C4 = \{+(a \in S)\}$ $\quad$ $C5 = \{-(a^{-1} \in S)\}$

The resolution steps:

C3,1 & C4,1 : add R2': $\{-(y \in S), -(a * y^{-1} = z), +(z \in S)\}$

R2',1 & C4,1 : add R3': $\{-(a * a^{-1} = z), +(z \in S)\}$

C3,2 & C4,1 : add R4': $\{-(x \in S), -(x * a^{-1} = z), +(z \in S)\}$

R3',1 & C1,1 : add R5': $\{+(e \in S)\}$

R4',2 & C2,1 : add R6': $\{-(e \in S), +(a^{-1} \in S)\}$

R5',1& R6',1 : add R1': $\{+(a^{-1} \in S)\}$

R1',1& C5,1 : add R7: $\square$

Fig. 1. An *SSPU*-resolution Proof

Note that $C3$ is the CNF of the subgroup criterion given below:

$$\forall x. \forall y. x \in S \wedge y \in S \Rightarrow y * x^{-1} \in S \tag{2}$$

This resolution proof basically consists of two applications of (2). The first one is the subproof rooted at $R5'$ that derives $e \in S$ from $a \in S$ and $a * a^{-1} \in S$. The second one is the subproof rooted at $R1$', which derives $a^{-1} \in S$ from $e \in S$, $a \in S$ and $e * a^{-1} \in S$.

Section 2 characterizes *SSPU*-resolutions, which can be understood as a sequence of applications of assertions. That is this class of resolution proofs can be understood in the same way as we understand proofs in mathematical textbooks. Based on this correlation Section 3 specifies a deterministic procedure that transforms *SSPU*-resolutions into neatly structured ND proofs with assertion level justifications. This section also describes how *SSPU-refutable* proofs can be restructured into *SSPU*-resolution proofs. Section 4 contains techniques that split an arbitrary resolution into interrelated *SSPU*-resolution segments.

The transformation process as a whole is largely deterministic and therefore efficient, reducing heuristic search only to the strategies that split nonunit-refutable proofs to unit-refutable proofs. This determinism also guarantees the quality of resulting proofs. Finally, we conclude this paper with a discussion of future improvements.

2 Application of an Assertion

To obtain proofs similar to those found in mathematical textbooks, the author has proposed a more abstract level of justifications for ND-style proofs, called *assertion level* [Hua92, Hua94b], where derivations are justified by the application of definitions or theorems (collectively called *assertions*).

Example 2 The application of the definition of subset (1) discussed in the introduction is logically equivalent to the compound proof segment below[1]:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\mathcal{A}:\ \ \forall S_1, S_2 .\ S_1 \subset S_2 \Leftrightarrow (\forall x .\ x \in S_1 \Rightarrow x \in S_2)}{U \subset F \Leftrightarrow (\forall x .\ x \in U \Rightarrow x \in F)}\forall E}{U \subset F \Rightarrow (\forall x .\ x \in U \Rightarrow x \in F)}\Leftrightarrow E \qquad U \subset F}{\forall x .\ x \in U \Rightarrow x \in F}\Rightarrow E}{a \in U \Rightarrow a \in F}\forall E \qquad a \in U}{a \in F}\Rightarrow E$$

Fig. 2. An ND proof applying the subset definition

Actually, the notion of the application of an assertion is specified in terms of a so-called *decomposition-composition* constraint imposed on such proof segments [Hua92, Hua94b]. The following two definitions are necessary for the discussion of this constraint.

Definition (Decomposition Rule) An inference rule of the form $\frac{\triangle\vdash F, \triangle\vdash P_1, \ldots, \triangle\vdash P_n}{\triangle\vdash Q}$ is a *decomposition* rule with respect to the formula schema F, if all applications of it, written as $\frac{\triangle\vdash F', \triangle\vdash P'_1, \ldots, \triangle\vdash P'_n}{\triangle\vdash Q'}$, satisfy the following condition: each $P'_1, \ldots, P'_n$ and Q' is

- (the negation of) a proper subformula of F', or
- (the negation of) a specialization of F' or of one of its proper subformulas.

Intuitively, a decomposition rule derives a conclusion that is part of one of its premises. Furthermore, other premises are also part of this special premise. Under this definition, $\frac{\triangle\vdash A\wedge B}{\triangle\vdash A}\wedge\mathrm{E}$ and its dual, $\frac{\triangle\vdash A\Rightarrow B, \triangle\vdash A}{\triangle\vdash B}\Rightarrow\mathrm{E}$, as well as $\frac{\triangle\vdash \forall x .\ P[x]}{\triangle\vdash P[a]}\forall\mathrm{E}$ are the only decomposition rules in the natural deduction calculus $\mathcal{NK}$ [Gen35]. To emphasize that the variable x occurs in P, we represent P by $P[x]$. Furthermore, we added the rule $\frac{\triangle\vdash A\vee B, \triangle\vdash \neg B}{\triangle\vdash A}\vee\mathrm{E}$ and its dual to decompose disjunctions.

Definition (Composition Rule)

[1] Only in this example, we present an ND proof as a tree to discuss the constraints.

An inference rule of the form $\frac{\triangle\vdash P_1,\ldots,\triangle\vdash P_n}{\triangle\vdash Q}$ is called a *composition* rule if all applications of it, written as $\frac{\triangle\vdash P'_1,\ldots,\triangle\vdash P'_n}{\triangle\vdash Q'}$, satisfy the following condition: each $P'_1,\ldots P'_n$ is

- (the negation of) a proper subformula of Q', or
- (the negation of) a specialization of Q' or of one of its proper subformulas.

Several examples of composition rules are $\frac{\triangle\vdash A,\triangle\vdash B}{\triangle\vdash A\wedge B}\wedge\mathrm{I}$, $\frac{\triangle\vdash A}{\triangle\vdash A\vee B}\vee\mathrm{I}$ and its dual, and $\frac{\triangle\vdash P[a]}{\triangle\vdash\exists x_{\blacksquare}\,P[x]}\exists\mathrm{I}$.

The decomposition-composition constraint can now be stated in a fairly simple way with the help of proof tree in Fig. 2. It requires that derivation along the branch from the assertion $\mathcal{A}$, which is always a leaf, to the root are all justified by a decomposition rule. This branch is called the *main branch*, which consists exclusively of a sequence of decompositions. In other words, the conclusion of a step in this branch is always (the negation of) a subformula of its predecessor. Other premises needed in the series of decompositions along the main branch (the leaves $U \subset F$ and $a \in U$ in the proof above) can be obtained by compositions. This actually guarantees that each intermediate node in such a proof tree is (the negation of) a subformula of the assertion applied, which explains why we call it an application of an assertion. For a formal treatment of this constraint, the readers are referred to [Hua92, Hua94b, Hua96].

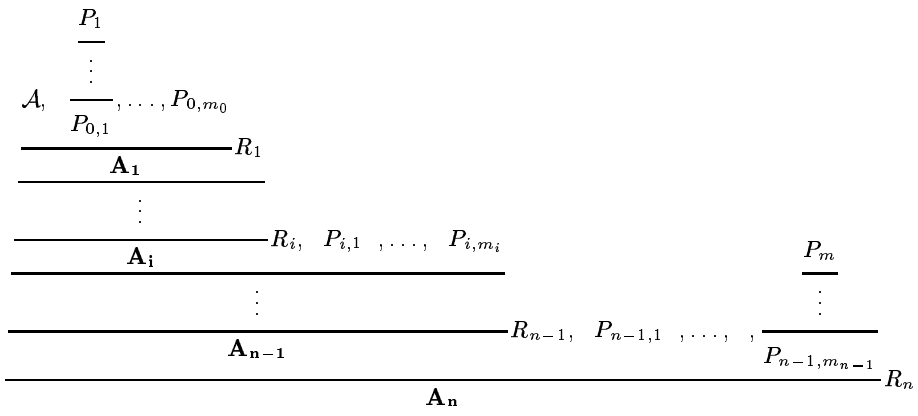

Fig. 3.

2.1 *SSPU*-Resolution

This section characterizes a class of resolution proof segments that can be seen as a sequence of applications of certain assertions. Since this concept is first defined in terms of ND proofs, we thereby establish a correspondence between these seemingly very different formalisms at a more abstract level. We begin

Composition and Decomposition Constraints

A logic level proof tree is called an application of an assertion $\mathcal{A}$, if it satisfies the following constraints:

1. *quasi-linearity property*: At every proof step, there is at most one premise depending on $\mathcal{A}$.
   It can easily be concluded that all proof nodes depending on $\mathcal{A}$ together form a branch in the proof tree, from the assertion $\mathcal{A}$ to the root. The branch is called the *main branch*. Nodes along this branch are now called the *main intermediate conclusions*. The general structure of a proof tree is illustrated in Figure 3, where the main branch is the branch from $\mathcal{A}$ to A_n. The formula $\mathcal{A}, A_1, ..., A_n$ denote the main intermediate conclusions, and $P_{i,1}, ..., P_{i,m_i}$ are the main preconditions for the decomposition step from A_i to A_{i+1}. In other words, the main intermediate conclusions are the only intermediate conclusions depending on $\mathcal{A}$. Furthermore, exactly one of their premises depends on $\mathcal{A}$. We are referring to this *linear order* when we later talk about the previous or subsequent main intermediate conclusions.
2. *decomposition property*: Main intermediate conclusions are justified by decomposition rules $R_1, \ldots, R_n$ with respect to the previous main intermediate conclusion. The inference along the main branch is therefore a linear process of decomposition of the assertion $\mathcal{A}$.
3. *composition property:* Other intermediate conclusions are justified by composition rules with respect to the corresponding intermediate conclusion.

Table 1. Composition and Decomposition Constraints

with the characterization of one step of application of an assertion in terms of resolution.

Theorem 1 (R-Application of an assertion)

Suppose there is a resolution with one non-unit initial clause A, n unit initial clauses $P_1, P_2, \ldots, P_n$, and a unit final resolvent Q. $\frac{P_1, P_2, \ldots, P_n}{Q}$ can be justified as an application of A', if $P_1, P_2, \ldots, P_n$ and Q are ground and contain no skolem constant, and A is the CNF of A'.

We call such a resolution segment an *R-application*. Instead of a formal inductive proof (see [HM96]), we provide an intuitive explanation. To do so, let us reexamine the decomposition-composition constraint. Basically it says, if an assertion is universally quantified, instantiate it only with one constant; if it is a conjunction, use only one branch of it; if it is a disjunction, try to negate the other branches and then use the remaining branch. ND inference rules handling hypotheses and thereby causing branching of a proof, like the case analysis and choice rule, are forbidden. Translated into resolution, this constraint means that only one clause normalized from an assertion is involved (hence only one non-unit clause, actually a condition stronger than necessary), and variables can be instantiated only once (hence ground premises). The unit premises and the unit conclusion condition can be understood in light of the $\vee$E rule. The restriction on skolem constants is due to the lack of decomposition rules for instantiating

existentially quantified formulas.

Example 2 (Continued)

Let us return to the example used in the last section, and examine the corresponding *R-application* illustrated below:

The set of initial clauses:
$C1 = \{+(a \in U)\}$ $C2 = \{+(U \subset V)\}$
$C3 = \{-(S_1 \subset S_2), -(x \in S_1), +(x \in S_2)\}$
The resolution steps:
C2,1 & C3,1: add R1: $\{-(x \in U), +(x \in V)\}$
R1,1 & C1,1: add R2: $+(a \in V)$

Apparently different formulations of the same problem may lead to the same clause form. Since the transformation of a resolution proof starts with a original problem formulation, it is sensitive to the structure of such formulations. Below we first consider an R-application with a simple problem structure, which can be transformed into a sequence of applications of assertions in a schematic way.

Definition (Simple-Structured Problem)

To distinguish atomic and no-atomic premises, we structure our problems into so-called simple-structured problems (*SSP*) using the following structure:

$$(A_1 \wedge \cdots \wedge A_m) \wedge (P_1 \wedge \ldots \wedge P_n) \Rightarrow Q$$

where $(A_1 \wedge \ldots \wedge A_m)$ is a conjunction of definitions or theorems, $(P_1 \wedge \ldots \wedge P_n)$ is a conjunction of (quantified) literals serving as the premises of the current problem, and the (quantified) literal Q is the conclusion.

Definition (*SSPU*-resolution)

A unit resolution is called an *SSPU*-resolution (unit resolution for simple-structured problems), if the initial unit clauses are normalized from the premises and the conclusion of a *SSP* problem.

An *SSPU*-resolution is called *ground*, if all its unit clauses are ground and contain no skolem constants. This implies that the premises and the conclusion are all ground literals. An *SSPU*-resolution is called *degenerated* if it contains the empty clause $\square$ as its only resolvent. This is the case when two initial unit clauses are contradictory. Apparently *SSPU*-resolution covers all binary resolutions without factorization, if we do not take (*SSP*) constraints into account. Although it is not complete, it does cover a wide range of mathematical problem we uncounted. We will discuss it in Section 3.4.

3 From *SSPU*-Resolution to Assertion Level ND-Proofs

3.1 The Basic Procedure for Ground *SSPU*-Resolution

This subsection first presents a deterministic algorithm that transforms *SSPU*-resolution proofs into ND proofs at the assertion level. We start with a procedure for ground *SSPU*-resolutions. It is based on the observation that a ground *SSPU*-resolution is a sequence of *R-application*s according to Theorem 1. Apart from the resolution proof (with a ground substitution for all variables in the resolution)

and the ND proof under construction, our algorithm maintains a relation δ that associates (among others) every unit initial clause and unit resolvent r with an ND proof line l, where r is the CNF of the formula in line l. This is denoted by $\langle r, l\rangle \in \delta$, and is an extension to the δ relation used in [Lin89].

Basic Procedure:

1. Initialization: Introduce the assertions, the premises and the negation of the conclusion as hypothesis lines, and initialize the δ relation to establish the correspondence between the initial clauses and the initial ND lines.
2. Translation: For every unit resolvent r do the following (suppose the subtree in the resolution proof rooted by r has leaves $(r_1, \ldots, r_n)$, $\langle r_i, l_i\rangle \in \delta$. C is the unique non-unit initial clause in this subtree, being the CNF of an ND line L): add a line l to the ND proof, which derives r from lines $l_1, \ldots, l_n$ by applying the assertion L, add $\langle r, l\rangle$ to δ.
3. Derive the conclusion by contradiction.

We will illustrate algorithm using example 1 throughout the paper. The development of the example is reverse to the order of actual processing: each time the input proof is produced by the operation described in the next session. The original proof found by the MKRP system [EO86] will be given in section 4.

Example 1(Continued)

The input resolution proof for this session can be found in Fig. 1. Together with the problem formulation below, it forms an *SSPU*-resolution proof:

- Premises: $a \in S \wedge a * a^{-1} = e \wedge e * a^{-1} = a^{-1}$
- Assertion: $\forall x_{\blacksquare} \forall y_{\blacksquare} x \in S \wedge y \in S \Rightarrow y * x^{-1} \in S$
- Conclusion: $a^{-1} \in S$

Both unit resolvents R5' and R1' in the proof in Fig. 1 are derived by an application of $C3$. First the sequence (a subtree) R2', R3', R5' derives $e \in S$ from the premises $a \in S$ and $a * a^{-1} = e$. Second the sequence R4', R6', R1' derives $a^{-1} \in S$ using as premises $a \in S$ and $e * a^{-1} = a^{-1}$. Finally, R1' is used to derive $\square$. This is a ground *SSPU*-resolution proof that can be transformed into an assertion level ND-proof by our basic procedure, see next page.

No	Hyp		Formula	Reason
			Initialization	
1.	1	$\vdash$	$\forall x, y, z_{\blacksquare}\, x \in S \wedge y \in S \wedge x * y^{-1} = z \Rightarrow z \in S$	(Asser1)
2.	2	$\vdash$	$a * a^{-1} = e$	(Hyp)
3.	3	$\vdash$	$e * a^{-1} = a^{-1}$	(Hyp)
4.	4	$\vdash$	$a \in S$	(Hyp)
5.	5	$\vdash$	$\neg(a^{-1} \in S)$	(Hyp)
			Translation	
6.	2,1,4	$\vdash$	$e \in S$	(Asser1 4 4 2)
7.	2,3,1,4	$\vdash$	$a^{-1} \in S$	(Asser1 6 4 3)
8.	2,3,1,4,5	$\vdash$	$\perp$	($\neg$E 5 7)
			Contradiction	
9.	2,3,1,4	$\vdash$	$a^{-1} \in S$	(Ind 8)

Note that the conclusion has already been derived in line 7 and that the last two lines are superfluous. This always happens when the conclusion clause is used only in the last step. Actually, we employ a refined version that avoids such indirect proofs. In this example, it skips line 5, 8 and 9.

Theorem 2 (Transformation Theorem)

Let $R_1, R_2, \ldots, R_n$ be the sequence of unit resolvents of a ground SSPU-*resolution proof π. Our basic procedure above produces from π an ND-style proof at the assertion level with $R_1, R_2, \ldots, R_n$ as its intermediate results.*

This theorem follows direct from theorem 1 by induction.

3.2 Decomposing Leaves of *SSPU*-resolution

Viewing proofs as trees, a unit resolution is a sequence of subtrees rooted at a unit resolvent. The basic procedure subsequently transforms such subtrees into an assertion level step in an ND proof, if the premises (the leaves) and the conclusion (the root) are ground literals. Otherwise, additional transformations are needed in order either to instantiate an *SSPU*-resolution, or to decompose the ND proof lines associated with the leaves of a resolution proof into literals. Such translations are performed by applying transformations rules.

Rule I$\forall$ below instantiates universal quantifiers which appear in a premise line. See [HM96] for a complete set of rules, these rules are adapted from [And80, Lin90]. The format of such rules is taken over from Lingenfelder [Lin90]. The rules are to be read as sollows: the lines on the left hand side of the arrow $\leadsto$ will be replaced by those on the right hand side.

I$\forall$ Rule

$$\begin{array}{llll} l_1 & \mathcal{A} & \vdash \forall x_{\blacksquare} F[x] & \text{Rule } \mathcal{R} \\ l_3 & \mathcal{A} & \vdash G & \pi \end{array} \quad \leadsto \quad \begin{cases} l_1 & \mathcal{A} & \vdash \forall x_{\blacksquare} F[x] & \text{Rule } \mathcal{R} \\ l_2 & \mathcal{A} & \vdash F[a] & \forall E \ l_1 \\ l_3 & \mathcal{A} & \vdash G & \pi' \end{cases}$$

Note that the resolution proof π that justifies l_3 must be instantiated to π'. The δ relation must be updated so that the literals connected with l_1 are connected with l_2 afterwards.

M-Choice Rule

$$\begin{array}{llll} l_1 & \mathcal{A} & \vdash \exists x_{\blacksquare} F[x] & \text{Rule } \mathcal{R} \\ l_4 & \mathcal{A} & \vdash G & \pi \end{array} \quad \leadsto \quad \begin{cases} l_1 & \mathcal{A} & \vdash \exists x_{\blacksquare} F[x] & \text{Rule } \mathcal{R} \\ l_2 & F_c & \vdash F_c & \text{Hyp} \\ l_3 & \mathcal{A}, F_c & \vdash G & \pi \\ l_4 & \mathcal{A} & \vdash G & (\text{Choice} l_1 l_3) \end{cases}$$

Since constants are used to instantiate the premises and the conclusion are taken from the ground substitution of the underlying resolution proof, there are restrictions concerning the order in which the transformation rules may be applied. Concretely, if the same constant is chosen, the M-Choice rule must preceed the I$\forall$ rule. For a detailed instantiation algorithm, see [HM96].

Below is one of the rules that split a premise.

I$\wedge$ Rule

$$l_1\ \mathcal{A} \vdash \quad F \wedge G \qquad \text{Rule } \mathcal{R} \qquad \rightsquigarrow \qquad \begin{cases} l_1\ \mathcal{A} & \vdash F \wedge G & \text{Rule } \mathcal{R} \\ l_2\ \mathcal{A} & \vdash F & \wedge E\ \ l_1 \\ l_3\ \mathcal{A} & \vdash G & \wedge E\ \ l_1 \end{cases}$$

3.3 Constructing *SSPU*-Resolutions by Permutation

Unit-refutable resolutions can be restructured into unit resolutions. If they are associated with a *SSP*, we call them *SSPU-refutable*. This property can be easily tested using the following property, which can be proven by induction.

Property (*SSPU*-refutable)

Let C, L, U be the number of non-unit initial clauses, the number of literals in non-unit initial clauses, and the number of unit initial clause in a ground version of a resolution proof π, π is SSPU-refutable *iff* $2(C-1)+U=L$.

Algorithm Permuting *SSPU*-Resolution Proofs

- subsequently do for every resolvent r:
  - If r does not results from a unit resolution step, then postpone r by removing r and connecting other resolvents using r as a parent clause to a proper parent clause of r.
  - If r results from a unit resolution step, keep r.
  - If a postponed resolution can be carried out as a unit resolution step, carry it out.
- If the empty clause is derived, report success, otherwise failure.

Example 1 (Continued)

The resolution proof below is an instantiation of a proof found by the theorem prover MKRP for the theorem stated previously. Note that the unit initial clauses are all ground. The resolution proof used in Fig. 1 can be obtained by the algorithm above.

The set of initial clauses:

$C1 = \{+(a * a^{-1} = e)\}$ $\quad$ $C2 = \{+(e * a^{-1} = a^{-1})\}$

$C3 = \{-(x \in S), -(y \in S), -(x * y^{-1} = z), +(z \in S)\}$

$C4 = \{+(a \in S)\}$ $\quad$ $C5 = \{-(a^{-1} \in S)\}$

The resolution steps:

C3,4 & C3,1: add R1: $\{-(x \in S), -(y \in S), -(x * y^{-1} = z), -(y' \in S), -(z * y'^{-1} = z'), +(z' \in S)\}$

R1,1 & C4,1: add R2: $\{-(y \in S), -(a * y^{-1} = z), -(y' \in S), -(z * y'^{-1} = z', +(z' \in S)\}$

R2,1 & C4,1: add R3: $\{-(a * a^{-1} = z), -(y' \in S), -(z * y'^{-1} = z'), +(z' \in S)\}$

R3,2 & C4,1: add R4: $\{-(a * a^{-1} = z), -(z * a^{-1} = z'), +(z' \in S)\}$

R4,1 & C1,1: add R5: $\{-(e * a^{-1} = z'), +(z' \in S)\}$

R5,1 & C2,1: add R6: $\{+(a^{-1} \in S)\}$

R6,1 & C5,1: add R7: $\square$

Fig. 4. Instantiated MKRP-Resolution Proof

Notice, clauses R2' to R6' in Fig. 1 are transformed from R2 to R6. R1' corresponds to R1, which is postponed. The difference between the two proofs can be intuitively described as follows: since R1 in Fig. 4 can be viewed as an application of C3 on itself,

the two natural sequences in Fig. 1 are mixed up here. The soundness of the algorithm is guaranteed, since at each step the resolution reapplies. For more complicated cases where applications of several assertions are mixed together, see [HM96]. A refinement is also made to even produce *direct SSPU*-resolution proofs, where the conclusion clause is postponed until the last step.

3.4 The Basic Procedure Enriched

Based on the discussion above, we now can present the complete basic procedure that transforms all *SSPU-refutable* resolutions into an ND proof at the assertion level. For the sake of completeness, it handles the degenerated *SSPU*-resolution as well.

Basic Procedure (Enriched):

2. Translation:
   (a) For each unit leaf r in the resolution proof such that $(r, l) \in \delta$, decompose l until it is a ground literal (see section 3.2),.
   (b) Create an *SSPU*-resolution proof from a *SSPU-refutable* proof by permutation(see section 3.3).
   (c) If degenerated *SSPU*-resolution, apply rule I$\bot$ (see below).
   (d) for every unit resolvent r do the following (suppose the subtree in the resolution proof rooted by r has leaves $r_1, \ldots, r_n$, and $(r_i, l_i) \in \delta$. C is the unique non-unit initial clause in this subtree, being the CNF of an ND line L): add a line l to the ND proof, which derives r from lines $l_1, \ldots, l_n$ by applying the assertion L, add $\langle r, l \rangle$ to δ.

I$\bot$ Rule

$$\begin{array}{llll} l_1 & \mathcal{A} & \vdash F & \text{Rule } \mathcal{R} \\ l_2 & \mathcal{A} & \vdash \neg F & \text{Rule } \mathcal{R}' \end{array} \quad \rightsquigarrow \quad \begin{cases} l_1 \ \mathcal{A} \vdash F & \text{Rule } \mathcal{R} \\ l_2 \ \mathcal{A} \vdash \neg F & \text{Rule } \mathcal{R}' \\ l_3 \ \mathcal{A} \vdash \bot & (\neg E l_1, l_2) \end{cases}$$

To summarize, the algorithm above creates an indirect ND-style proof at the assertion level for every *SSPU-refutable* proof. The resulting proof is basically a sequence of applications of assertions involved, interleaved with some instantiations and decompositions. If the conclusion clause is used only in the last step, it even produces a direct proof.

4 Splitting an Arbitrary resolution into SSPU-Refutable Proof Segments

Until now we have only examined operations which decompose or instantiate ND proof lines. Although the latter also instantiate the corresponding resolution proof, none of them change the structure of a given resolution. To become a *SSPU*-resolution, however, a proof needs enough unit clauses. In this section, we introduce transformation rules which split an arbitrary resolution proof into a set of interrelated *SSPU*-resolution proofs. After that, the *SSPU*-resolution subproofs are translated by our basic procedure. The aim of such split is to decompose some non-unit clauses into unit clauses, and thereby producing *SSPU-refutable* proofs. Since non-unit clauses are CNFs of disjunctive premises (including negated conjunctions) or a conjunctive conclusion, we need two dual rules to handle them. The M-Case rule below is one of them. It splits the

original problem into two. The corresponding update on the resolution side effects not only the δ-relation, but also splits resolution proof π into π_1 and π_2 [HM96].

M-Case Rule

$$\begin{array}{llll} l_1 & \mathcal{A} & \vdash F \vee G & \text{Rule } \mathcal{R} \\ l_4 & \mathcal{A} & \vdash H & \pi \end{array} \quad \rightsquigarrow \quad \left\{ \begin{array}{llll} l_1 & \mathcal{A} & \vdash F \vee G & \text{Rule } \mathcal{R} \\ l_2 & F & \vdash F & \text{Hyp} \\ l_3 & \mathcal{A}, F & \vdash H & \pi_1 \\ l_4 & G & \vdash G & \text{Hyp} \\ l_5 & \mathcal{A}, G & \vdash H & \pi_2 \\ l_6 & \mathcal{A} & \vdash H & \text{Case}(l_1, l_3, l_5) \end{array} \right.$$

Summarizing the discussion up to now, our procedure first splits an arbitrary resolution proof into *SSPU-refutable* subproofs, and then proceeds according to the relation between *SSPU*-resolution and the application of assertions. The slitting is basically the same as the traditional procedure, only it is carried out in a controlled way: We distinguish between premises to be decomposed to literals, and definitions and theorems that should act as the assertions in *SSPU*-resolutions. For the split of resolution proofs and for the decomposition of ND lines connected to the leaves of *SSPU-refutable* proofs, we have adapted a complete subset of the transformation rules described in [Lin90]. Below is the integrated algorithm that transforms an arbitrary resolution proof into an ND proof with assertion level justifications. Note that since steps justified by the application of an assertion is defined in terms of a compound ND proof segment, they can be expanded correspondingly if required by a user.

An Integrated Algorithm:

1. If the input resolution is *SSPU*-refutable, then call the basic procedure.
2. Otherwise partition the premises into premises and assertions if not already specified and split one non-unit clause which is the CNF of a premise. Recursively call this algorithm with all subproofs created by splitting.

The completeness of the algorithm is obvious. If a resolution proof has enough unit clause, we can always decompose (if necessary) the corresponding ND lines to make it a *SSPU*-resolution. Otherwise, we can alway split it to increase the number of unit clauses. The basic procedure is very efficient, since the transformation of *SSPU*-resolution and the permutation are both linear. Since the heuristic search used in previous systems is restricted to the strategies concerning the split of resolution proofs, which is seldom used more than one or two times in most tasks, our transformation is usually much cheaper than the original search process. Actually, a high percentage of real-world mathematical problems we have encountered are *SSPU*-resolution, including all examples handled by Lingenfelder [Lin89, Lin90], although this is a question of statistics.

Example 1 (Continued)

Finally, we examine the entire transformation process of example 1 already discussed in section 3.1 and 3.3. The original proof produced by the theorem prover MKRP is given below (edited for layout and renaming):

- Premises: $\forall u\, u * u^{-1} = e \wedge \forall u\, e * u = u$
- Assertion: $\forall x\, \forall y\, x \in S \wedge y \in S \Rightarrow y * x^{-1} \in S$, other group definitions.
- Conclusion: $\forall x\, x \in S \Rightarrow x^{-1} \in S$

The set of initial clauses:

$C1 = \{+(u * u^{-1} = e)\}$ $\quad C2 = \{+(e * w = w)\}$

$C3 = \{-(x \in S), -(y \in S), -(x * y^{-1} = z), +(z \in S)\}$

$C4 = \{+(v \in S)\}$ $\quad C5 = \{-(q^{-1} \in S)\}$

The resolution steps:

C3,4 & C3,1:	add R1:	$\{-(x \in S), -(y \in S), -(x * y^{-1} = z), -(y' \in S), -(z * y'^{-1} = z'), +(z' \in S)\}$
R1,1 &C4,1:	add R2:	$\{-(y \in S), -(v * y^{-1} = z), -(y' \in S), -(z * y'^{-1} = z', +(z' \in S)\}$
R2,1 &C4,1:	add R3:	$\{-(v * v^{-1} = z), -(y' \in S), -(z * y'^{-1} = z', +(z' \in S)\}$
R3,2 &C4,1:	add R4:	$\{-(v * v^{-1} = z), -(z * v^{-1} = z'), +(z' \in S)\}$
R4,1 &C1,1:	add R5:	$\{-(e * v^{-1} = z'), +(z' \in S)\}$
R5,1 &C2,1:	add R6:	$\{+(v^{-1} \in S)\}$
R6,1 &C5,1:	add R7:	$\square$

Fig. 5. Original MKRP-Resolution Proof

This proof can be transformed into an assertion level ND proof by the following steps. The initialization creates line 1, line 2, line 3, and line 10. Decomposition of the conclusion line 10 then adds line 9 and line 4. Simultaneously, all variables in the unit initial clauses, namely u, w, v, q are instantiated to a, a^{-1}, a, a, as given in Fig. 4, producing line 5 and line 6. The instantiated resolution proof is then permuted as described in section 3.3. Finally, the basic procedure translates two *R-application*s as described in section 3.1, and adds line 7 and line 8. Note that the two assertion level steps in line 7 and line 8 can be expanded into calculus level ND proof segments in a schematic way.

NNo	Hyp		Formula	Reason
1.	1	$\vdash$	$\forall x, y, z_{\blacksquare}\ x \in S \wedge y \in S \wedge x * y^{-1} = z \Rightarrow z \in S$	(Asser1)
2.	2	$\vdash$	$\forall u_{\blacksquare}\ u * u^{-1} = e$	(Def-Inverse)
3.	3	$\vdash$	$\forall u_{\blacksquare}\ e * u = u$	(Def-Unit)
4.	4	$\vdash$	$a \in S$	(Hyp)
5.	2	$\vdash$	$a * a^{-1} = e$	(Def-Inverse)
6.	3	$\vdash$	$e * a^{-1} = a^{-1}$	(Def-Unit)
7.	2,1,4	$\vdash$	$e \in S$	(Asser1 4 4 5)
8.	2,3,1,4	$\vdash$	$a^{-1} \in S$	(Asser1 7 4 6)
9.	2,3,1	$\vdash$	$a \in S \Rightarrow a^{-1} \in S$	($\Rightarrow$I 8)
10.	2,3,1	$\vdash$	$\forall x_{\blacksquare}\ x \in S \Rightarrow x^{-1} \in S$	($\forall$I 9)

This work is implemented as part of a system called *PROVERB*, which transforms and verbalizes machine-found proofs [Hua94a]. The final proof above can be verbalized by the system *PROVERB* as following:

> "Let a be in S. According to the definition of inverse element, $a * a^{-1} = e$. According to our hypothesis, e is in S. $e * a^{-1} = a^{-1}$ according to the definition of unit. Again according to our hypothesis, a^{-1} is in S."

5 Discussion

There are two related yet distinct lines of research concerning different logic calculi. On the one hand, there are traditional results showing that certain calculi can simulate each other. More recently, results were reported comparing length of proofs in different calculus (for example, see [Ede92] for more details). This work, however, follows the

other line of research aimed at transforming machine-found proofs into proofs more readable for human users [And80, Mil83, Pfe87, PN90, Lin90, SK95].

Instead of formulating transformation strategies at the level of ND inference rules as in earlier works, we try to understand resolution proofs directly in terms of the vocabularies human mathematicians use to talk about proofs. We have shown, that an *SSPU*-resolution corresponds directly to ND proof segments that mathematicians intuitively understand as the application of an assertion. The significance of this result is mainly twofold. First, contrary to the intuition of many, some resolution proofs, after some restructuring, become quite readable themselves. This leads to a natural correspondence between resolution proofs and ND proofs at a more abstract level. Second, since variations at the calculus level are abstracted away, the main part of our algorithm is deterministic. Shorter and more natural proofs can now be obtained for all examples involving interesting definitions and theorems, as it is usually the case in mathematical problems. For these problems the cost of transformation is linear. Nevertheless, there are limitations to our approach. The performance does not improve much for typical logical exercises concerning primarily manipulations of nested quantifications. A simple example is the theorem $\exists x_{\blacksquare} \forall y_{\blacksquare} [P(x) \Rightarrow P(y)]$.

The system *PROVERB* as it stands is already used in various ways. Assertion level proofs that are transformed from resolution proofs or that are abstracted from ND proofs are used to facilitate user's understanding, to serve as the basis to formulate methods of proof planning, and to produce natural language proofs.

Although we replaced the basic transformation procedure used [Lin90], It is reasonable to investigate how certain global strategies experimented there can be incorporated into our procedure, in particular those concerning the split of resolution proofs and concerning the insertion of a lemma [PN90, Lin90]. We are also working to extend the notion of *SSPU*-resolution to deal with factorizations and paramodulations. Another interesting future development is the adaptation of this technique for other proof formalisms, for instance expansion trees and connection proofs [SK95].

References

[And80] Peter B. Andrews. Transforming matings into natural deduction proofs. In *Proc. of the 5th International Conference on Automated Deduction*, pages 281–292. Springer, 1980.

[Ede92] Elmar Eder. *Relative Complexities of First Order Calculi.* Artificial Intelligence. Vieweg, 1992.

[EO86] Norbert Eisinger and Hans Jürgen Ohlbach. The Markgraf Karl Refutation Procedure. In *Proc. of 8th International Conference on Automated Deduction.* Springer, 1986.

[Gen35] Gerhard Gentzen. Untersuchungen über das logische Schließen I. *Math. Zeitschrift*, 39:176–210, 1935.

[HM96] Xiaorong Huang and Andreas Maier. Natual deduction proofs can be short as resolution proofs. Seki-report, Fachbereich Informatik, Universität des Saarlandes, Saarbrücken, Germany, 1996. forthcoming.

[Hua92] Xiaorong Huang. Applications of assertions as elementary tactics in proof planning. In V. Sgurev and B. du Boulay, editors, *Artificial Intelligence V - Methodology, Systems, Applications*, pages 25–34. Elsevier Science, the Netherlands, 1992.

[Hua94a] Xiaorong Huang. *PROVERB*: A system explaining machine-found proofs. In Ashwin Ram and Kurt Eiselt, editors, *Proc. of 16th Annual Conference of the Cognitive Science Society*, pages 427–432, Atlanta, USA, 1994. Lawrence Erlbaum Associates.

[Hua94b] Xiaorong Huang. Reconstructing proofs at the assertion level. In Alan Bundy, editor, *Proc. of 12th International Conference on Automated Deduction*, LNAI-814, pages 738–752. Springer, 1994.

[Hua96] Xiaorong Huang. *Human Oriented Proof Presentation:A Reconstructive Approach*. Infix, Sankt Augustin, 1996.

[Lin89] Christoph Lingenfelder. Structuring computer generated proofs. In N.S. Sridharan, editor, *Proc. of IJCAI-89*, pages 378–383. Morgan Kaufmann, 1989.

[Lin90] Christoph Lingenfelder. *Transformation and Structuring of Computer Generated Proofs*. PhD thesis, Universität Kaiserslautern, Kaiserslautern, Germany, 1990.

[Mil83] Dale A. Miller. *Proofs in Higher-Order Logic*. PhD thesis, CMU, Pittsburgh, Pennsylvania, USA, 1983.

[Pfe87] Frank Pfenning. *Proof Transformation in Higher-Order Logic*. PhD thesis, CMU, Pittsburgh, Pennsylvania, USA, 1987.

[PN90] Frank Pfenning and Daniel Nesmith. Presenting intuitive deductions via symmetric simplification. In Mark E. Stickel, editor, *Proc. of 10th International Conference on Automated Deduction*, LNAI 449, pages 336–350. Springer, 1990.

[SK95] Stephan Schmitt and Christoph Kreitz. On transforming intuitionistic matrix proofs into standard-sequent proofs. In Peter Baumgartner, Reiner Hähnle, and Joachim Posegga, editors, *Proc. of the 4th Workshop on Theorem Proving with Analytic Tableaux and Related Methods*, LNAI 918, pages 106–121, 1995.

This article was processed using the LaTeX macro package with LLNCS style