



## **Statement**

This is the preliminary version of a technical report i will publish at the department of computer science of the Saarland University, Saarbrücken, Germany. Please note that this current version is neither complete (with respect to the content) nor in the final format (with respect to the representation and language). Since i submitted a paper for publication at CADE-17 which is related to the content of this report, i decided to provide already this preliminary version to support my submission. Hence, this preliminary version is primarily for the referees of my CADE-17 paper. I will publish the complete report in a few weeks.

Andreas Meier

Saarbrücken, February 9, 2000





# Transformation of Machine-Found Proofs into Assertion Level Proofs

Andreas Meier  
Fachbereich Informatik, Universität des Saarlandes, Saarbrücken

February 9, 2000



## Abstract

Most automated theorem provers suffer on the problem that the proofs they produce are difficult to understand even for experienced users. Therefore, many efforts have been made to transform, abstract and restructure machine-found proofs to produce proof formats better understandable for humans. One of the most preferred target formats is the natural deduction proof. Current approaches suffer mainly on two problems. First, the state-of-the-art transformation procedures generate very often natural deduction proofs with many indirect parts since they translate at the literal level steps from the machine-found proofs into the natural deduction proofs. Secondly, the natural deduction calculus itself is not eligible for presenting mathematical proofs. The problem is that inferences in the natural deduction calculus are still on the level of syntactical manipulations of logical connectives and quantifiers. They are not on the level of theorem or definition applications as we have in mathematical textbooks. Thus, an additional abstraction is needed to reach a level of meaningful steps.

In this report we describe an algorithm which transforms refutation graphs – a more abstract representation of resolution proofs – into natural deduction proofs at the assertion level. The assertion level allows for meaningful macro-steps justified by the application of theorems, lemmas, or definitions which are collectively called *assertions*. Furthermore, our algorithm is based on translating structures in refutation graphs that represent assertion applications directly into assertion applications in the natural deduction proofs. Thus, we need not to decompose to the literal level and to abstract then the resulting proofs, but we translate directly at the assertion level steps from the refutation graphs into the natural deduction proofs. Because of the assertion steps the resulting proofs are significantly shorter and much closer to mathematical practice than the pure natural deduction proofs. Moreover, they contain rarely indirect parts. Our transformation algorithm is complete on refutation graphs with equality.

# Contents

<b>1</b>	<b>Introduction</b>	<b>5</b>
1.1	Motivation . . . . .	5
1.2	Classification of this Report in the Context of the TRAMP System . . . . .	6
1.3	Overview of the Report . . . . .	8
<b>2</b>	<b>Foundations</b>	<b>9</b>
2.1	First-Order Logic with Equality . . . . .	9
2.2	Clause Normalization . . . . .	11
2.3	Refutation Graphs . . . . .	14
2.4	ND-Calculus . . . . .	20
2.5	Transformation Problems . . . . .	22
<b>3</b>	<b>Transformation by Decomposition</b>	<b>25</b>
3.1	The Literal Base Cases . . . . .	25
3.2	Decomposition by Transformation Rules . . . . .	27
3.3	Handling of Instantiations . . . . .	31
3.4	The Literal Transformation-Algorithm . . . . .	38
3.5	Heuristic Control and Bad Results . . . . .	41
<b>4</b>	<b>The Assertion Level</b>	<b>45</b>
4.1	Assertion Applications in ND-Proofs . . . . .	46
4.2	Abstraction to Assertion Level . . . . .	50
4.3	Assertion Applications in Terms of Refutation Graphs . . . . .	53
<b>5</b>	<b>UCS-Decomposable Graphs</b>	<b>59</b>
5.1	UCS-Decompositions and UCS-Decomposition Algorithm . . . . .	59
5.2	Binary-Linked Refutation Graphs . . . . .	65
5.3	Further Results on Binary-Linked Refutation Graphs . . . . .	69
5.4	Contra-Links Free Graphs . . . . .	72
5.5	Not UCS-Decomposable Graphs . . . . .	82
5.6	Conclusion . . . . .	87
<b>6</b>	<b>A UCS-Based Transformation Algorithm</b>	<b>89</b>
6.1	The UCS-Decomposable Base Case . . . . .	89
6.2	Satisfying the Graph Conditions . . . . .	98
6.3	Obtaining UCS-decomposable Graphs . . . . .	102
6.4	The UCS-Based Transformation Algorithm . . . . .	111



<b>7</b>	<b>Extensions of the UCS-Based Transformation Algorithm</b>	<b>113</b>
<b>8</b>	<b>Examples</b>	<b>115</b>
<b>A</b>	<b>The Transformation Rules</b>	<b>117</b>
A.1	The Standard Rules . . . . .	118
A.2	The Alternative Rules . . . . .	121
<b>B</b>	<b>Breaking Refutation Graphs</b>	<b>125</b>
B.1	Breaking One Clause . . . . .	125
B.2	Breaking Several Clauses . . . . .	131
<b>C</b>	<b>The Proof of the UCS-Theorem</b>	<b>133</b>

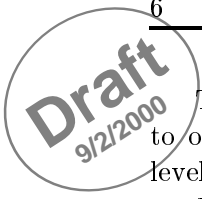
# Chapter 1

## Introduction

### 1.1 Motivation

Automated theorem provers (ATPs) have reached a considerable strength. But unfortunately, most ATPs can produce proofs only in their own particular and machine-oriented formalism such that their proofs are difficult to read even for experienced users. Hence, to explain the proofs to a human user we need a representation of the machine-found proofs that is comprehensible. Similarly, other systems that want to make use of the achievements built into the ATPs need to adapt the output of the ATPs to their own needs. Thereby, different systems have different demands on the representation of the machine-found proofs. Whereas for the communication between fully automatic systems the representation format should mainly enable a fast communication, interactive applications – where user-system communication is crucial – need mainly a representation format that is comprehensible or that they can easily transform to a comprehensible format.

There exist different approaches to transform proofs in machine-oriented formats into human-oriented proofs. Systems as PROVERB [HF96] and ILF [Da97] aim to communicate directly with a human user and can provide as output a translation of machine-found proofs into natural language proofs. Other approaches transform the machine-found proofs into natural deduction (ND) proofs [And80, Mil83, Pfe87, Lin90, SK95]. But the resulting ND-proofs are usually long and contain many levels of indirect proofs. The first problem is that, in general, the transformation procedures decompose the theorem assumptions and conclusion using ND-rules that eliminate quantifiers and connectors until literals are reached. Then at the literal level steps from the machine-found proofs are transformed into ND-steps. Various heuristics have been added to the base algorithms to improve the quality of the resulting ND-proofs. Unfortunately, they rarely go beyond vague general guidelines [Lin90, PN90]. The second problem is the target representation itself. Although each single step in a ND-proof is natural and easy to understand, a reader is often lost in the large number of low-level, pure syntactic manipulations of logical quantifiers and connectives. In contrast thereto, proofs found in standard mathematical textbooks are primarily justified by the applications of definitions or theorems. For instance, the derivation of  $a \in F$  from  $U \subset F$  and  $a \in U$  would be usually justified in a mathematical textbook as one step, an application of the definition of subset encoded as  $\forall S_1, S_2. S_1 \subset S_2 \Leftrightarrow \forall x. x \in S_1 \Rightarrow x \in S_2$ . In contrast thereto, a corresponding ND-proof consists of whole a sequence of steps and is so too tedious and hard to comprehend.



To obtain proofs closer to mathematical practice an abstraction of ND-proofs is needed to obtain steps meaningful for humans. A cognitively motivated approach is the assertion level proposed by Huang [Hua94c]. He proposes a formalization of the intuitive notion of the application of a definition, lemma, or theorem (collectively called assertions) in terms of ND-proof segments. Furthermore, he presents procedures that abstract ND-proofs to the assertion level. This abstraction works well – in the sense that it creates proofs of ‘good’ quality – only on ND-proofs that are mainly direct and short. But ND-proofs generated by previous transformation procedures contain often multiple levels of indirect proofs and a very long. This undesirable characteristics will be inherited by the abstraction. To solve this problem, Huang found structures in resolution proofs representing also assertion steps [Hua96b]. With the same concept formulated in both formalisms, assertion applications in resolution proofs can be translated directly into assertion steps in a ND-proof (without decomposing to literal level).

In this report we apply a similar approach to refutation graphs, a more abstract representation of resolution proofs. Whereas Huang indicates in [Hua96b] only a proof that these structures in resolution proofs represent assertion applications, we describe corresponding structures in refutation graphs and give a formal proof that they represent assertion applications. The practical reason why we prefer refutation graphs is that proofs in many other refutation based formalisms can be transformed easily into refutation graphs (e.g., in [Eis88] a transformation algorithm for resolution proofs is described). Furthermore, the coherences between the input clauses (which literals are contradictory) are directly visible, instead of being spread in separate steps in the resolution proof. This facilitates the usage of heuristics guiding the transformation process. We present an algorithm that is based on translating steps at the assertion level from the refutation graphs into the ND-proofs. The base case of this algorithm are refutation graphs that consist only of a sequence of steps representing assertion applications. Such a refutation graph is transformed by translating successively its steps into corresponding assertion applications in the ND-proof. Huang indicates in [Hua96b] that arbitrary resolution proofs need first to be split into subproofs that are sequences of assertion-level steps. The same holds for refutation graphs. However, whereas the splitting is only sketched out in [Hua96b], we give a structural characterization of such refutation graphs that enables us to formulate a complete algorithm where the splitting is fully spelled out. Furthermore, we show that our algorithm is open to other calculi by integrating paramodulation. Thus, altogether our algorithm is complete on refutation graphs with equality. Moreover, we give many examples of applications of our algorithm that demonstrate that it produces assertion level ND-proofs that are mainly direct and significantly more compact and better structured than basic ND-proofs. Other work indicates that such assertion level proofs are well suited as basic representation level for further human-oriented proof presentation [Hor99].

## 1.2 Classification of this Report in the Context of the TRAMP System

We implemented the transformation algorithm that we present in this report within the TRAMP system. TRAMP can transform the output of several ATPs for first order logic with equality into ND-proofs at the assertion level. Hence, via TRAMP other systems can make use of the ATPs and obtain a uniform representation of the different machine-oriented formalisms. Currently, TRAMP is able to process the output of the ATPs SPASS, BLIKSEM, OTTER,



WALDMEISTER, PROTEIN, and EQP (see [SS97] for references).

TRAMP consists of three parts: (1) At the heart is the transformation process described in this report that transforms a *problem description* (consisting of a set of first order formulas, the theorem assumptions, and one first order formula, the theorem conclusion) and the corresponding output of an ATP into a ND-proof at assertion level. (2) For each ATP interfaced by TRAMP there is a minor transformation process that can transform a problem description into input suitable for this ATP. (3) A communication shell handles the access of the ATPs by TRAMP and the reachability of TRAMP by other systems.

This communication between TRAMP and the ATPs is handled by MATHWEB technology [FK99]. MATHWEB is a system for distributed automated theorem proving. It provides the functionality to turn existing tools into mathematical services that are homogeneously integrated into a networked proof development environment. Thus, both TRAMP and the different ATPs are equipped with MATHWEB communication shells. Then via MATHWEB TRAMP can be reached by other MATHWEB services and can reach the ATP services.

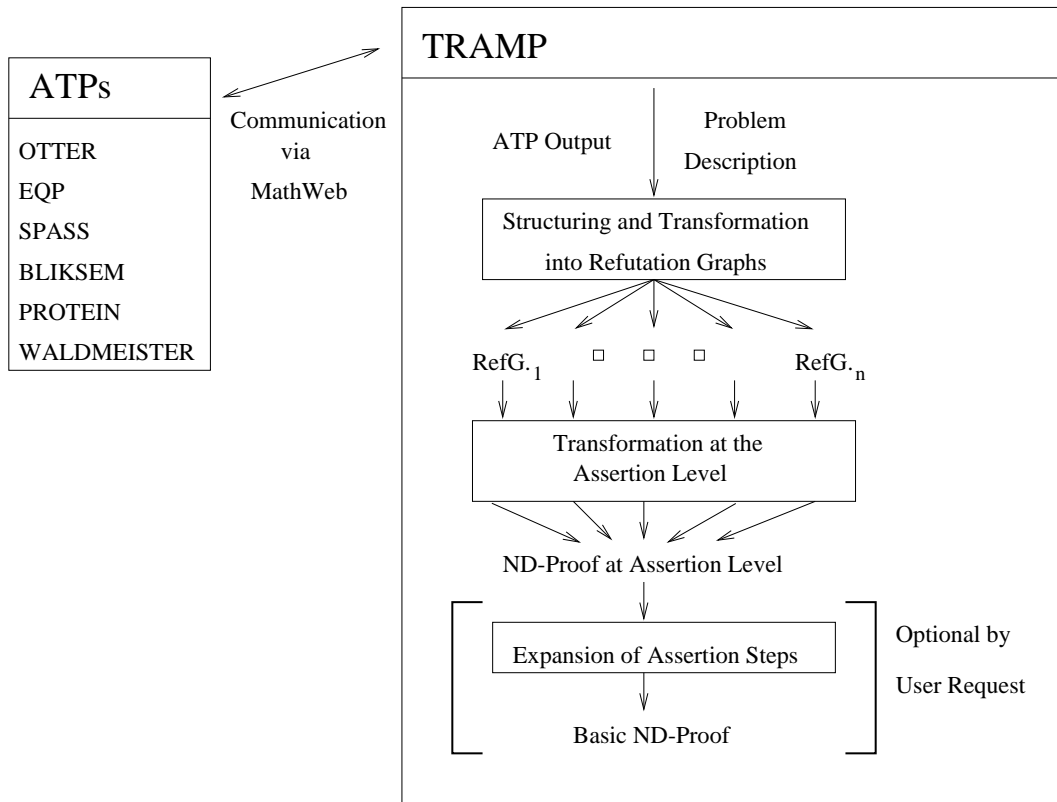


Figure 1.1: The TRAMP System.

The transformation process takes as input a problem description and the corresponding output of an ATP. It produces a ND-proof at assertion level through 3 subprocesses:

**Structuring and Transformation into Refutation Graphs:** The ATP output is structured by cutting of lemmas from the main proof. The resulting proof parts of the (remaining) main problem and the lemmas are each transformed into refutation graphs.

The result of this first subprocess is a sequence of so-called *transformation problems* each consisting of a problem description and a refutation graph as proof.

**Transformation at Assertion Level:** Each transformation problem is transformed into an assertion level ND-proof of the conclusion of its problem description from the assumption of its problem description. These proofs cohere via the lemmas such that we obtain altogether one ND-proof at assertion level. The rest of this report is a detailed description of this subprocess.

**(Optionally) Expansion of Assertion Steps:** If requested by the user each assertion application can be expanded to a sequence of pure ND-steps such that the resulting proof is a basic ND-proof.

### 1.3 Overview of the Report

We describe in this report the algorithm that transforms one transformation problem into a ND-proof at assertion level. Thereby, a transformation problem consists of problem description and a refutation graph such that the refutation graph is a proof of the conclusion of the problem description from the assumptions of the problem description. Likewise, the resulting ND-proof is a proof of this conclusion from these assumptions. This algorithm consists of two phases: (1) In the first phase we decompose the assumptions, the conclusion, and the refutation graph until we obtain refutation graphs consisting only of a sequence of steps representing translatable assertion applications. In this phase we apply ND-rules that eliminate quantifiers and connectors to decompose the assumptions and the conclusion. Since these formulas and the clauses of the refutation graph are related (the clauses of the refutation graph are in the clause normal form of the formulas), we have to decompose thereby also the refutation graphs. (2) In the second phase we transform the refutation graphs that consist only of a sequence of steps representing translatable assertion applications by translating these steps successively into corresponding assertion applications in the ND-proof.

We start in the next section with a short introduction of the logical foundations of this report and introduce all needed concepts. In the third chapter we explain how a transformation algorithm works that decomposes until it reaches the literal level and translates then at the literal level steps from the refutation graph into the ND-proof. In particular, we give a detailed description how we handle the instantiations of quantified variables and skolemization. The fourth chapter introduces the assertion application in both ND-proofs and refutation graphs. In particular, we introduce the *UCS* – a structure in refutation graphs – and prove under which conditions a UCS represents an assertion application. In the fifth chapter we describe how graphs look like that consist only of a sequence of UCSs. Subsequently, we combine in the sixth chapter the results of the chapters three, four, and five to the pronounced transformation algorithm that decomposes only until the assertion level is reached and transforms then at this level assertion applications from the refutation graph into the ND-proof.

## Chapter 2

# Foundations

In this chapter we describe the logical foundations of this report and introduce some needed concepts. The first section introduces the underlying logic, the first-order logic with equality. In the second, third, and fourth section we introduce the clause normalization, refutation graph, and ND-calculus we use throughout this report. Finally, we define in the fifth section the concept of transformation problems, the basic structure we use in the transformation algorithms which we will introduce in this report.

### 2.1 First-Order Logic with Equality

The underlying logic of this report is the first-order logic with equality. We give here no general introduction into first-order logic, since we assume that the reader is familiar with it (otherwise we suggest to read common textbooks as [Fit90, Gal86]).

**Definition 2.1.1 (Signature).**

A *first-order signature*  $\Sigma$  is a pair  $\Sigma = (\mathcal{F}, \mathcal{P})$  where

- $\mathcal{F}$  is a countable set of so-called *function symbols*. Each function symbol has a arity  $n \geq 0$ . If the arity is 0 we call the symbol also a *constant symbol*.
- $\mathcal{P}$  is a countable set of so-called *predicate symbols*. Each predicate symbol has a arity  $n \geq 0$ .

■

The sets  $T(\Sigma)$ ,  $APL(\Sigma)$ , and  $PL(\Sigma)$  of the terms, the atomic formulas, and the formulas (with equality) over  $\Sigma$  and an infinite set  $\mathcal{V}$  of variable symbols are defined as follows:

$$T(\Sigma) = \mathcal{V} \mid f(t_1, \dots, t_n) \text{ when } f \in \mathcal{F} \text{ with arity } n \text{ and } t_1, \dots, t_n \in T(\Sigma)$$

$$APL(\Sigma) = p(t_1, \dots, t_n) \text{ when } p \in \mathcal{P} \text{ with arity } n \text{ and } t_1, \dots, t_n \in T(\Sigma), \\ t_1 = t_2 \text{ when } t_1, t_2 \in T(\Sigma)$$

$$PL(\Sigma) = A \in APL(\Sigma), \\ \perp \mid \top \mid \neg A \mid A \wedge B \mid A \vee B \mid A \Rightarrow B \mid A \Leftrightarrow B \text{ when } A, B \in PL(\Sigma), \\ \forall x.A \mid \exists x.A \text{ when } A \in PL(\Sigma) \text{ and } x \in \mathcal{V}$$

Furthermore, we introduce the following concepts:

**Definition 2.1.2 (Literal, Clause, Contradictory Literals, Ground Terms).**

1. A formula is called *literal* if it is an atomic formula or the negation of an atomic formula.
2. A clause is a set of literal formulas. A clause with the literals  $\{L_1, \dots, L_n\}$  is written as  $[L_1, \dots, L_n]$ . In a clause a positive literal  $A$  is written as  $+A$  and a negative literal  $\neg A$  is written as  $\perp A$ .
3. Two literals are called *contradictory* if the one is the negation of the other.
4. A term is called *ground* if it contains no variable symbols. Similarly, clauses and literals containing no variable symbols are also called *ground*.

■

**Definition 2.1.3 (Substitution).**

A substitution is a mapping  $\sigma : \mathcal{V} \mapsto T(\Sigma)$ . A substitution that maps the variable symbols  $x_1, \dots, x_n$  to the terms  $t_1, \dots, t_n$  is written as  $\{x_1 \mapsto t_1, \dots, x_n \mapsto t_n\}$ .

■

**Definition 2.1.4 (Application of a Substitution).**

The application of a substitution  $\sigma = \{x_1 \mapsto t_1, \dots, x_n \mapsto t_n\}$  on terms and atomic formulas is inductively defined by:

- for  $x \in \mathcal{V}$ :  $x\sigma = \begin{cases} x, & \text{if } x \neq x_1, \dots, x_n \\ t_i, & \text{if } x = x_i \end{cases}$
- for  $f \in \mathcal{F}$  and  $t'_1, \dots, t'_m \in T(\Sigma)$ :  $f(t'_1, \dots, t'_m)\sigma = f(t'_1\sigma, \dots, t'_m\sigma)$ ,
- for  $p \in \mathcal{P}$  and  $t'_1, \dots, t'_m \in T(\Sigma)$ :  $p(t'_1, \dots, t'_m)\sigma = p(t'_1\sigma, \dots, t'_m\sigma)$ .

The application of  $\sigma$  on formulas is inductively defined by:

- $(\neg A)\sigma = \neg(A\sigma)$  where  $A \in PL(\Sigma)$ ,
- $(A \circ B)\sigma = (A\sigma) \circ (B\sigma)$  where  $\circ = \wedge, \vee, \Leftrightarrow, \Rightarrow$  and  $A, B \in PL(\Sigma)$ ,
- $(qx.A)\sigma = qx.(A\sigma')$  where  $A \in PL(\Sigma)$ ,  $q = \forall, \exists$ , and  $\sigma'$  consists of all the pairs of  $\sigma$  except each pair  $x_i \mapsto t_i$  in  $\sigma$  where  $x_i = x$ .

The application of  $\sigma$  on a clause  $[L_1, \dots, L_m]$  is defined by:

$$[L_1, \dots, L_m]\sigma = [L_1\sigma, \dots, L_m\sigma].$$

■

Henceforth, some proofs and algorithms shall be based on case splits by distinguishing formulas by their connectives and quantors. To facilitate these case splits we introduce another classification for formulas, the so-called *uniform notation* introduced by [Smu68, Fit90]. All formulas are divided into:

**$\alpha$ -formulas:** are such formulas that we regard intuitively as conjunctions,  $\alpha = \alpha_1 \wedge \alpha_2$ :

- $\alpha = F \wedge G$  where  $\alpha_1 = F$  and  $\alpha_2 = G$ ,
- $\alpha = \neg(F \vee G)$  where  $\alpha_1 = \neg F$  and  $\alpha_2 = \neg G$ ,
- $\alpha = \neg(F \Rightarrow G)$  where  $\alpha_1 = F$  and  $\alpha_2 = \neg G$ .



**$\beta$ -formulas:** are such formulas that we regard intuitively as disjunctions,  $\beta = \beta_1 \vee \beta_2$ :

- $\beta = F \vee G$  where  $\beta_1 = F$  and  $\beta_2 = G$ ,
- $\beta = F \Rightarrow G$  where  $\beta_1 = \neg F$  and  $\beta_2 = G$ ,
- $\beta = \neg(F \wedge G)$  where  $\beta_1 = \neg F$  and  $\beta_2 = \neg G$ .

**$\gamma$ -formulas:** are such formulas that we regard intuitively as universal quantifications,  $\forall x.\gamma[x]$ :

- $\gamma = \forall x F$  where  $\gamma[x] = F$ ,
- $\gamma = \neg \exists x F$  where  $\gamma[x] = \neg F$ .

**$\delta$ -formulas:** are such formulas that we regard intuitively as existential quantifications,  $\exists x.\delta[x]$ :

- $\delta = \exists x F$  where  $\delta[x] = F$ ,
- $\delta = \neg \forall x F$  where  $\delta[x] = \neg F$ .

## 2.2 Clause Normalization

A first-order formula can be arbitrary complex and nested. Such formulas are not appropriate for automated theorem proving. Therefore, for machine-oriented applications the formulas are first transformed into a normal form that is better suitable for machine-oriented formalisms like the resolution principle [Rob65]. We introduce in this section the so-called *clause normal form* of a formula. This definition is a constructive one. The corresponding algorithm transforms a formula into its corresponding clause normal form which is a set of clauses.

### Definition 2.2.1 (Clause Normal Form (CNF)).

Let  $F$  be a formula, then the *clause normal form (CNF)* of  $F$  is a set  $CNF(F^\emptyset)$  of clauses which is inductively defined by:

**Literal-rule:**  $CNF(L^V) = \{[L]\}$ , where  $L$  is a literal

**$\neg\neg$ -rule:**  $CNF((\neg\neg F)^V) = CNF(F^V)$

**$\alpha$ -rule:**  $CNF(\alpha^V) = CNF(\alpha_1^V) \cup CNF(\alpha_2^V)$ , where  $\cup$  is the union of two sets

**$\beta$ -rule:**  $CNF(\beta^V) = CNF(\beta_1^V) \times CNF(\beta_2^V)$ , where  $\times$  is the crossproduct of two sets

**$\gamma$ -rule:**  $CNF(\gamma^V) = CNF(\gamma[x']^{V \cup \{x'\}})$ , where  $x'$  is a new variable

**$\delta$ -rule:**  $CNF(\delta^V) = CNF(\delta[sk_{term}(V)]^V)$ , where  $sk_{term}(V) = sk_{new}(x_1, \dots, x_n)$  if  $V = \{x_1, \dots, x_n\}$  and  $sk_{new}$  is a new function symbol, a so-called *skolem function symbol*.

■

### Definition 2.2.2 ( $\gamma, \delta$ -Quantified Subformulas and Variables).

Let  $F$  be a formula and  $G$  a quantified subformula of  $F$ . Then  $G$  is called  *$\gamma$ -quantified* with respect to  $F$  if and only if  $G$  is decomposed during the clause normalization of  $F$  by the  $\gamma$ -rule. Similarly  $G$  is called  *$\delta$ -quantified* with respect to  $F$  if and only if  $G$  is decomposed during the clause normalization of  $F$  by the  $\delta$ -rule.

A variable is called  $\gamma$ -quantified ( $\delta$ -quantified) with respect to  $F$  if it is bound in a  $\gamma$ -quantified ( $\delta$ -quantified) subformula of  $F$ . ■

These definitions of  $\gamma$ - and  $\delta$ -quantified subformulas classify quantifications by the logical behavior of the quantifier. Hence, for example, both  $\neg\exists x.P(x)$  and  $\forall x.P(x)$  contain  $\gamma$ -quantified formulas, whereas  $\neg\forall x.Q(x)$  contains a  $\delta$ -quantified formula.

During the recursive computation of the CNF of a formula  $F$ , each subformula is associated with a set  $V$  of variables. This set is the collection of all  $\gamma$ -quantified variables, in whose scopes the subformula is. By application of the  $\delta$ -rule the occurrences of  $\delta$ -quantified variables are replaced by skolem terms depending exactly from these  $\gamma$ -quantified variables in  $V$ .

Next, we introduce the example henceforth used in this report (therefore, we call it the *standard example*).

*Example 2.2.3 (Standard Example).*

To prove is the following theorem, known from group theory:

*Let  $(G, \circ)$  be a group and  $S \subset G$ . Furthermore, for  $S$  holds the subgroup-criterion: for all  $x, y \in S$  is also  $y^{-1} \circ x \in S$ , then for every  $v \in S$  is also  $v^{-1} \in S$ .*

To prove this theorem we need the properties of the neutral element and the inverse elements of a group:

if  $e$  is the neutral element of  $(G, \circ)$ , then  $x \circ e = x$  for each  $x \in G$ ,

for each element  $x \in G$  exists an element  $x^{-1} \in G$  – the inverse of  $x$  – such that  $x^{-1} \circ x = e$ .

If we formalize this problem in first order logic we obtain the following formulas:

$$F_1 = \forall u.(u \circ e = u) \text{ (neutral element)}$$

$$F_2 = \forall w.(w^{-1} \circ w = e) \text{ (inverse element)}$$

$$F_3 = \forall x, y, z.((x \in S \wedge (y \in S \wedge y^{-1} \circ x = z)) \Rightarrow (z \in S)) \text{ (subgroup-criterion)}$$

$$F_4 = \forall v.(v \in S \Rightarrow v^{-1} \in S) \text{ (theorem to prove)}$$

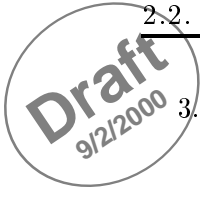
We compute now the CNF of these formulas.

1.  $F_1$  :

$$\begin{aligned} CNF(F_1^\emptyset) &= CNF((\forall u.(u \circ e = u))^\emptyset) \\ &= CNF((u_1 \circ e = u_1)^{\{u_1\}}) \quad (\gamma\text{-rule}) \\ &= \{ [+ (u_1 \circ e = u_1)] \} \quad (\text{Literal-rule}) \end{aligned}$$

2.  $F_2$  :

$$\begin{aligned} CNF(F_2^\emptyset) &= CNF((\forall w.(w^{-1} \circ w = e))^\emptyset) \\ &= CNF((w_1^{-1} \circ w_1 = e)^{\{w_1\}}) \quad (\gamma\text{-rule}) \\ &= \{ [+ (w_1^{-1} \circ w_1 = e)] \} \quad (\text{Literal-rule}) \end{aligned}$$

3.  $F_3$  :

$$\begin{aligned}
CNF(F_3^\emptyset) &= CNF((\forall x, y, z. ((x \in S \wedge (y \in S \wedge y^{-1} \circ x = z)) \\
&\quad \Rightarrow (z \in S)))^\emptyset) \\
&= CNF(((x_1 \in S \wedge (y_1 \in S \wedge y_1^{-1} \circ x_1 = z_1)) \\
&\quad \Rightarrow (z_1 \in S))^{\{x_1, y_1, z_1\}}) \quad (3 * \gamma\text{-rule}) \\
&= CNF((\neg(x_1 \in S \wedge (y_1 \in S \wedge y_1^{-1} \circ x_1 = z_1)))^{\{x_1, y_1, z_1\}} \\
&\quad \times CNF((z_1 \in S))^{\{x_1, y_1, z_1\}}) \quad (\beta\text{-rule}) \\
&= (CNF((\neg(x_1 \in S))^{\{x_1, y_1, z_1\}}) \times (CNF((\neg(y_1 \in S))^{\{x_1, y_1, z_1\}}) \\
&\quad \times CNF((\neg(y_1^{-1} \circ x_1 = z_1))^{\{x_1, y_1, z_1\}}))) \\
&\quad \times CNF((z_1 \in S))^{\{x_1, y_1, z_1\}}) \quad (2 * \beta\text{-rule}) \\
&= (\{\perp (x_1 \in S)\} \times (\{\perp (y_1 \in S)\} \\
&\quad \times \{\perp (y_1^{-1} \circ x_1 = z_1)\})) \\
&\quad \times \{+(z_1 \in S)\} \quad (4 * \text{Literal-rule}) \\
&= \{\perp (x_1 \in S), \perp (y_1 \in S), \perp (y_1^{-1} \circ x_1 = z_1), + (z_1 \in S)\}
\end{aligned}$$

4.  $F_4$  : (the theorem which should be proved is negated for the clause normalization)

$$\begin{aligned}
CNF((\neg F_4)^\emptyset) &= CNF((\neg(\forall v. (v \in S \Rightarrow v^{-1} \in S)))^\emptyset) \\
&= CNF((\neg(sk_1 \in S \Rightarrow sk_1^{-1} \in S))^{\emptyset}) \quad (\delta\text{-rule}) \\
&= CNF((sk_1 \in S)^\emptyset) \cup CNF((\neg(sk_1^{-1} \in S))^{\emptyset}) \quad (\alpha\text{-rule}) \\
&= \{+(sk_1 \in S)\} \cup \{\perp (sk_1^{-1} \in S)\} \quad (2 * \text{Literal-rule}) \\
&= \{+(sk_1 \in S), \perp (sk_1^{-1} \in S)\}
\end{aligned}$$

Altogether, we obtain the following clauses by the clause normalization:

$$C_1 = [+ (u_1 \circ e = u_1)],$$

$$C_2 = [+ (w_1^{-1} \circ w_1 = e)],$$

$$C_3 = [\perp (x_1 \in S), \perp (y_1 \in S), \perp (y_1^{-1} \circ x_1 = z_1), + (z_1 \in S)],$$

$$C_4 = [+ (sk_1 \in S)], \text{ and}$$

$$C_5 = [\perp (sk_1^{-1} \in S)] \quad \blacksquare$$

The derivation of one clause which is produced by clause normalization can be represented by a tree. Thereby, the applications of the clause normalization rules are the edges and the intermediate formulas are the nodes. The root node is the original formula and the leaves are the literals of the derived clause. We call such trees *derivation trees*. Each application of a clause normalization rule is represented by:

 $\neg\neg$ -rule:

$$\begin{array}{c}
(\neg\neg G)^V \\
| \\
G^V
\end{array}$$

 $\alpha$ -rule:

$$\begin{array}{c}
\alpha^V \\
| \\
\alpha_i^V (i = 1 \text{ or } 2)
\end{array}$$

Draft  
9/2/2009

$\beta$ -rule:

$$\begin{array}{c} \beta^V \\ \swarrow \quad \searrow \\ \beta_1^V \quad \beta_2^V \end{array}$$

$\gamma$ -rule:

$$\begin{array}{c} \gamma^V \\ | \\ \gamma[x']^{V \cup \{x'\}} \end{array}$$

$\delta$ -rule:

$$\begin{array}{c} \delta^V \\ | \\ \delta[sk_{term}(V)]^V \end{array}$$

For instance, the following tree is the derivation tree for the clause  $C_3$ :

$$\begin{array}{c} (\forall x, y, z. ((x \in S \wedge (y \in S \wedge y^{-1} \circ x = z)) \Rightarrow (z \in S)))^{\emptyset} \\ | \\ (\forall y, z. ((x_1 \in S \wedge (y \in S \wedge y^{-1} \circ x_1 = z)) \Rightarrow (z \in S)))^{\{x_1\}} \\ | \\ (\forall z. ((x_1 \in S \wedge (y_1 \in S \wedge y_1^{-1} \circ x_1 = z)) \Rightarrow (z \in S)))^{\{x_1, y_1\}} \\ | \\ ((x_1 \in S \wedge (y_1 \in S \wedge y_1^{-1} \circ x_1 = z)) \Rightarrow (z \in S))^{\{x_1, y_1, z_1\}} \\ \swarrow \quad \searrow \\ (\neg(x_1 \in S \wedge (y_1 \in S \wedge y_1^{-1} \circ x_1 = z)))^{\{x_1, y_1, z_1\}} \quad (z_1 \in S)^{\{x_1, y_1, z_1\}} \\ \swarrow \quad \searrow \\ (\neg(x_1 \in S))^{\{x_1, y_1, z_1\}} \quad (\neg(y_1 \in S \wedge y_1^{-1} \circ x_1 = z))^{\{x_1, y_1, z_1\}} \\ \swarrow \quad \searrow \\ (\neg(y_1 \in S))^{\{x_1, y_1, z_1\}} \quad (\neg(y_1^{-1} \circ x_1 = z))^{\{x_1, y_1, z_1\}} \end{array}$$

## 2.3 Refutation Graphs

The clause normal form of the conclusion and the assumptions of a theorem is suitable as input for an ATPs. The ATPs themselves are based on various machine-oriented formalisms. Thus, each ATP produces proofs in its own particular format. To develop for each of these formats an algorithm that transforms the machine-found proofs into ND-proofs would be a large effort. Lesser effort (both in theoretical and implementation) arises if the proofs of the ATPs are first transformed into a intermediate uniform representation. Then we have to perform and to develop the transformation process into a ND-proof only for this uniform





formalism. The practical reason why we prefer refutation graphs as intermediate uniform representation is that proofs in many other refutation based formalisms can be transformed easily into refutation graphs (e.g., in [Eis88] a transformation algorithm for resolution proofs is described). Furthermore, the coherences between the input clauses (which literals are contradictory) are directly visible. This facilitates the handling of the proof objects.

Informally spoken, a refutation graph is a graph, whose nodes are literals (grouped together to clauses), whose edges connect contradictional links, and which has to fulfill some structural graph conditions. In the following we give a formal definition of refutation graphs. Further details and properties of refutation graphs can be found in [Sho76, Eis88].

**Definition 2.3.1 (Clause Graph).**

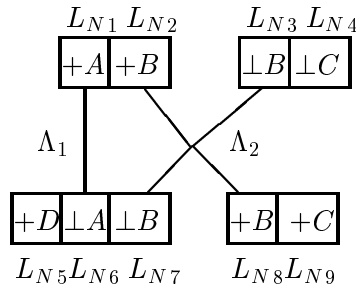
A *clause graph* is a quadruple  $G = (\mathcal{L}, \mathcal{C}, M_{Lit}, \Pi)$ , where

- $\mathcal{L}$  is a finite set. Its members are called the *literal nodes* of  $G$ .
- $\mathcal{C} \subset 2^{\mathcal{L}}$  is a partition of the set of literal nodes. The members of  $\mathcal{C}$  are called the *clause nodes* of  $G$ . Contrary to the standard definition of a partition  $\emptyset \in \mathcal{C}$  is allowed. A clause node consisting of the literal nodes  $L_{N_1}, \dots, L_{N_n}$  is denoted by  $[L_{N_1}, \dots, L_{N_n}]$ . The clause node of a literal node  $L_N$  is denoted by  $|L_N|$ .
- $M_{Lit}$  is a mapping from  $\mathcal{L}$  to a set of literals, which labels the literal nodes with literals. The literal of a literal node  $L_N$  is denoted by  $M_{Lit}(L_N)$ .
- The set of *links*  $\Pi$  is a partition of a subset of  $\mathcal{L}$ , such that for all  $\Lambda \in \Pi$  the following link conditions hold:
  - $\pi_1$ : All the literal nodes in one link are labeled with literals whose atoms are unifiable.
  - $\pi_2$ : There must be at least one positive and one negative literal in a link.

Literal nodes belonging to no link at all are called *pure*. Each link  $\Lambda$  has two opposite *shores*, a *positive shore*  $S^+(\Lambda)$ , and a *negative shore*  $S^-(\Lambda)$ , consisting of the literal nodes with positive and negative literals, respectively. We call the literals that belong respective the mapping  $M_{Lit}$  to the literals nodes of  $G$  the *literals* of  $G$ . ■

Intuitively spoken, links connect contradictory literals.

*Example 2.3.2.*



The set of literal nodes is  $\{L_{N_1}, \dots, L_{N_9}\}$ .

There are four clause nodes

$[L_{N_1}, L_{N_2}]$ ,  $[L_{N_3}, L_{N_4}]$ ,  $[L_{N_5}, L_{N_6}, L_{N_7}]$  and  $[L_{N_8}, L_{N_9}]$

and two links

$\Lambda_1 = \{L_{N1}, L_{N6}\}$  and  $\Lambda_2 = \{L_{N2}, L_{N3}, L_{N7}, L_{N8}\}$

with

$S^+(\Lambda_1) = \{L_{N1}\}$ ,  $S^-(\Lambda_1) = \{L_{N6}\}$  and

$S^+(\Lambda_2) = \{L_{N2}, L_{N8}\}$ ,  $S^-(\Lambda_2) = \{L_{N3}, L_{N7}\}$ .

The literal nodes  $L_{N4}, L_{N5}, L_{N9}$  are pure. ■

Clause graphs can be manipulated with the following operations whose applications produce again clause graphs (taken from [Lin90]).

**Adding a link:** To add a link  $\Lambda$  to a clause graph means to add to  $\Pi$  a set of (until now) pure literal nodes from  $\mathcal{L}$ , that present this new link  $\Lambda$ . Note that the conditions  $\pi_1$  and  $\pi_2$  have also to hold for this new link. The used literal nodes are not longer pure.

**Adding a literal node:** To add a literal node  $L_N$  to a clause graph means to add a new pure literal node to  $\mathcal{L}$ . Simultaneously,  $L_N$  has also to be added to a clause node.

**Adding a clause node:** To add a clause node  $C_N$  to a clause graph means to add a set of new pure literal nodes to  $\mathcal{L}$  and to construct with this new literal nodes the new clause  $C_N$  in  $\mathcal{C}$ .

**Removing a link:** To remove a link  $\Lambda$  from a clause graph means to remove the set of literal nodes that represent  $\Lambda$  from  $\Pi$ . Thus, all these literal nodes become pure.

**Removing a literal node:** To remove a literal node  $L_N$  from a clause graph means to remove  $L_N$  from  $\mathcal{L}$  and from its corresponding clause node  $|L_N|$ . If  $L_N$  is contained in a link  $\Lambda$  it is also removed from  $\Lambda$ . If then the conditions  $\pi_1$  and  $\pi_2$  do not longer hold for  $\Lambda$ ,  $\Lambda$  is also removed.

**Removing a clause node:** To remove a clause node  $C_N$  from a clause graph means to remove  $C_N$  from  $\mathcal{C}$  and to remove all literal nodes of  $C_N$ .

### Definition 2.3.3 (Subgraphs).

$G_{sub}$  is called *subgraph* of a clause graph  $G$ , if  $G_{sub}$  is obtained from  $G$  by removing some clause nodes and links. ■

### Definition 2.3.4 (Walks, Separating Links).

1. A *walk* in a clause graph  $G$  is an alternating sequence  $C_{N0}, \Lambda_1, C_{N1}, \Lambda_2, \dots, C_{Nn-1}, \Lambda_n, C_{Nn}$  ( $n \geq 1$ ) of clause nodes and links such that for each pair of clause nodes  $C_{Nj-1}, C_{Nj}$ , one clause contains a literal node of the positive shore of the connecting link  $\Lambda_j$  and the other contains a literal node of its negative shore.

Draft  
9/2/2000

2. A set  $\psi$  of links is *separating*  $G$  if there exist two clause nodes  $C_{N_1}$  and  $C_{N_2}$  connected by a walk in  $G$  which are no longer connected when  $\psi$  is removed from  $G$ . If  $\psi$  consists of only a single link  $\Lambda$ , we say that  $\Lambda$  *separates*  $G$ .

■

Next we define refutation graphs. Refutation graphs are the class of clause graphs that represent refutation proofs. This means in particular that there are no pure literal nodes.

**Definition 2.3.5 (Deduction Graphs, Refutation Graphs).**

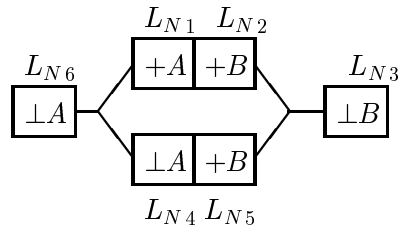
1. A *trail* in a clause graph  $G$  is a walk where all the links used are distinct. A trail *joins* its start and end clause nodes. A *trail to a link*  $\Lambda$  is a trail whose last clause has a literal in  $\Lambda$ .
2. A *cycle* is a trail joining a clause node to itself. If a clause graph  $G$  contains such a cycle it is called *cyclic*, otherwise *acyclic*.
3. A clause graph  $G$  is called *connected* if each pair of clause nodes is joined by a trail.
4. A *deduction graph* is a non-empty, ground (all its literals are ground) and acyclic clause graph.
5. 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 global substitution is required that transforms them into ground graphs without destroying the link conditions  $\pi_1$  and  $\pi_2$  for any of its links.
6. A *minimal deduction (refutation) graph* is one containing no proper subgraph which is itself a deduction (refutation) graph.

■

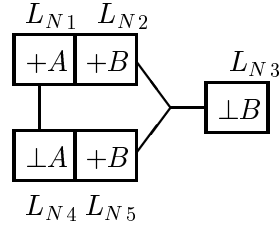
The clause graph of Example 2.3.2 is not a refutation graph, since it contains both pure literal nodes ( $L_{N_4}, L_{N_5}, L_{N_9}$ ) and a cycle (from  $L_{N_2}$  to  $L_{N_7}$  using  $\Lambda_2$ , then from  $L_{N_6}$  to  $L_{N_1}$  using  $\Lambda_1$ ). The following is an example of a refutation graph.

*Example 2.3.6.*

Let  $G$  be the following graph:



$G$  is a refutation graph, but it is not minimal, since we obtain by removing the literal node  $L_{N_6}$  a subgraph of  $G$  which is a refutation graph, too.



This graph is minimal, since removing any other clause nodes or links would produce pure literal nodes. ■

Eisinger describes in [Eis88] an algorithm that transforms a resolution proof into a minimal refutation graph. In general, the original clauses as used as input for an ATP contain variables (e.g. the clauses of the standard example in Example 2.2.3) whereas the clauses in a refutation graph have to be ground. But between the ground clauses of the refutation graph and the original clauses (with variables) exists the following connection: For each ground clause  $C_g$  in the refutation graph exist a corresponding original clause  $C$  and a substitution  $\sigma$  such that  $C_g = C\sigma$ . We call such a substitution a *ground substitutions* of  $C$ . To emphasize the correspondance between the original clauses and the ground clauses in the refutation graphs we represent henceforth the ground clause as result of applying the corresponding ground substitution on the corresponding original clause. Therefore, we use in the refutation graph the original clause and write the ground substitutions next to the clause nodes. Furthermore, we write the clause nodes of such clauses that result from clausenormalizing a conclusion with double boxes. Figure 2.1 contains the refutation graph for our standard example in Example 2.2.3.

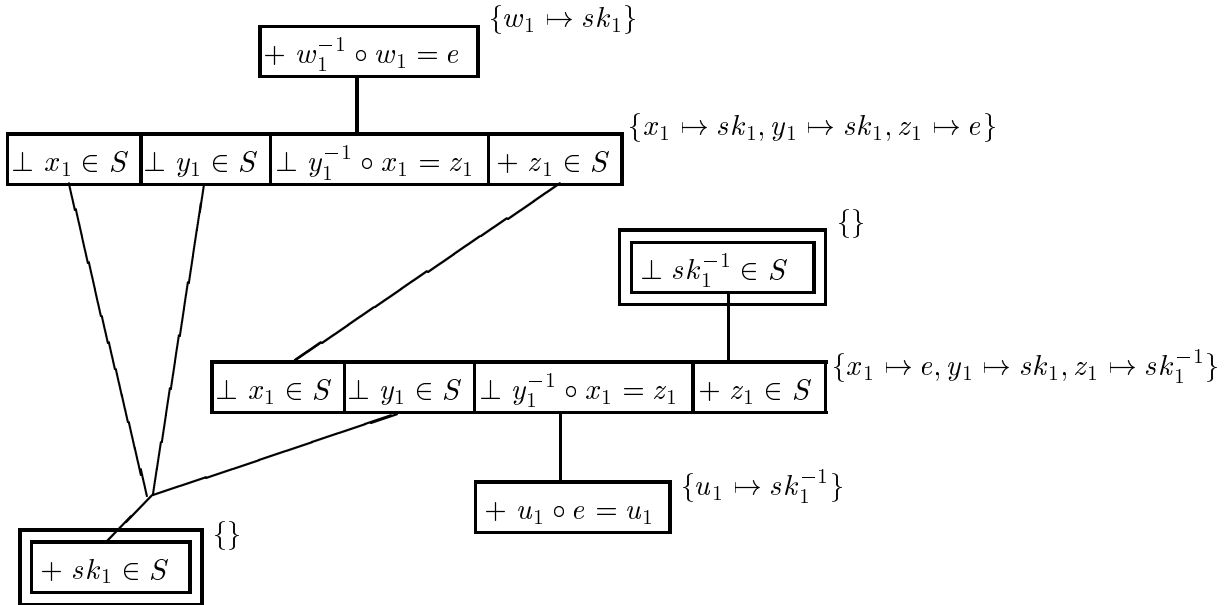
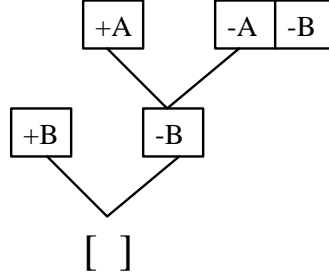


Figure 2.1: The refutation graph of the standard example

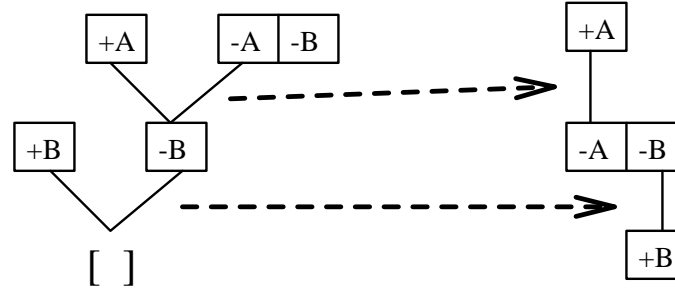
The next example illustrates the correspondance between refutation graphs and resolution proofs.

Example 2.3.7.

Given is the following resolution proof using the clauses  $[+A]$ ,  $[\perp A, \perp B]$  and  $[+B]$ :

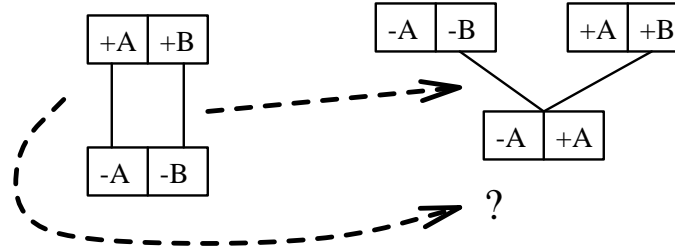


The resolution steps are transformed into corresponding links:



Since the resolution proof ends with the empty clause, in the corresponding refutation graph each literal is connected with a contradictory one (there can be no pure literal nodes).

The next example illustrates why refutation graphs have to be acyclic to represent refutation proofs.



It is not possible to transform this cyclic graph into a resolution proof, since the second link between  $\perp A$  and  $+A$  can not be transformed into a resolution step. ■

Some final remarks:

1. Although we distinguished in the formal definition of a clause graph between literal nodes and literals and clause nodes and clauses, we talk henceforth only of literals and of clauses.
2. Henceforth we use the following notation:  
 We use  $G, G', G'', \dots, G_1, G_2, \dots$  to range over refutation graphs or graphs in general.  
 We use  $C, C', C'', \dots, C_1, C_2, \dots$  to range over clauses. Later we are distinguishing between unit and non unit clauses. Then we use  $UC, UC', UC'', \dots, UC_1, UC_2, \dots$  to

range over unit clauses and  $NC, NC', NC'', \dots, NC_1, NC_2, \dots$  to range over non unit clauses.

We use  $\Lambda, \Lambda', \Lambda'', \dots, \Lambda_1, \Lambda_2, \dots$  to range over links.

We use  $L, L', L'', \dots, L_1, L_2, \dots$  to range over literals.

3. We assume henceforth that the refutation graphs that we obtain by transforming the machine-found proofs are minimal (the transformation algorithm in [Eis88] from resolution proofs into refutation graphs provides minimal graphs).

## 2.4 ND-Calculus

In this paper we use ND-proofs in the linearized version first used in [And80].

### Definition 2.4.1 (ND-Line, ND-Proof).

A *ND-line* consists of

- a finite (possibly empty) set of formulas, called the *hypotheses* of the ND-line
- a single formula called the *conclusion* or simply formula of the ND-line
- a justification

A ND-line with hypotheses  $\Delta$ , formula  $F$  and justification rule  $\mathcal{R}$ , is denoted as:  $L.\Delta \vdash F (\mathcal{R})$ . Thereby,  $L$  is a label for this line. Labels are used to identify the lines later for deductions.

A finite sequence  $S$  of ND-lines is a *ND-proof* of a formula  $F$  from a set of hypotheses  $\mathcal{A}$ , if the following hold:

- $F$  is the conclusion of the last ND-line of  $S$
- $\mathcal{A}$  is the set of hypotheses of the last line
- each line is correctly justified by one of the rules given later.

A ND-line  $\Delta \vdash F (\mathcal{R})$  in a sequence  $S$  is *correctly justified* if  $\Delta \vdash F$  is the lower part of an application of the rule  $\mathcal{R}$  and there are ND-lines in  $S$  before this line, that fulfill the upper part of  $\mathcal{R}$ . ■

The rule set we use is based on Gentzen's NK [Gen35], but is enriched with further, derived rules. We add such additional rules to obtain proofs that are easier better comprehensible. But note that we can obtain ND-proofs in Gentzen's NK by expanding each application of an additional rule to a sequence of applications of rules in NK. The following rule set is taken from [Hua94a].

$F, G, H$  are meta variables for formulas and  $\mathcal{A}$  for a finite set of formulas. The metavariable  $t$  in  $\forall E$  and  $\exists I$  stands for an arbitrary term while the meta variable  $c$  in  $\forall I$  and *Choice* stands for a constant (with the restrictions given explicitly). The formula schemata in the upper

**Draft**  
9/2/2000

part of a rule are the premisses and the formula schemata in the lower part are the conclusion of the rule.

$$\begin{array}{c}
\frac{}{\mathcal{A}, F \vdash F} \text{Hyp} \quad \frac{}{\vdash F \vee \neg F} \text{TND} \quad \frac{\mathcal{A}, F \vdash G}{\mathcal{A} \vdash F \Rightarrow G} \Rightarrow I \\
\\
\frac{\mathcal{A}, G \vdash \perp}{\mathcal{A} \vdash \neg G} \text{IP}_1 \quad \frac{\mathcal{A}, \neg G \vdash \perp}{\mathcal{A} \vdash G} \text{IP}_2 \quad \frac{\mathcal{A} \vdash F \quad \mathcal{A} \vdash \neg F}{\mathcal{A} \vdash \perp} \text{Contradiction} \\
\\
\frac{\mathcal{A} \vdash \perp}{\mathcal{A} \vdash F} \text{Indirect} \quad \frac{\mathcal{A} \vdash \neg(\neg F)}{\mathcal{A} \vdash F} \neg\neg E \\
\\
\frac{\mathcal{A} \vdash F \quad \mathcal{A} \vdash G}{\mathcal{A} \vdash F \wedge G} \wedge I \quad \frac{\mathcal{A} \vdash F \wedge G}{\mathcal{A} \vdash F} \wedge E_L \quad \frac{\mathcal{A} \vdash F \wedge G}{\mathcal{A} \vdash G} \wedge E_R \\
\\
\frac{\mathcal{A} \vdash F \vee G \quad \mathcal{A}, F \vdash H \quad \mathcal{A}, G \vdash H}{\mathcal{A} \vdash H} \text{Cases} \quad \frac{\mathcal{A} \vdash F}{\mathcal{A} \vdash F \vee G} \vee I_L \quad \frac{\mathcal{A} \vdash G}{\mathcal{A} \vdash F \vee G} \vee I_R \\
\\
\frac{\mathcal{A} \vdash F \vee G \quad \mathcal{A} \vdash \neg F}{\mathcal{A} \vdash G} \vee E_L \quad \frac{\mathcal{A} \vdash F \vee G \quad \mathcal{A} \vdash \neg G}{\mathcal{A} \vdash F} \vee E_R \\
\\
\frac{\mathcal{A} \vdash \forall x.F}{\mathcal{A} \vdash F[t]} \forall E \quad \frac{\mathcal{A} \vdash F[c]}{\mathcal{A} \vdash \forall x.F} \forall I \\
\text{thereby in } \forall I \text{ } c \text{ is not allowed to occur in } \forall x.F \text{ or in any of the formulas in } \mathcal{A}. \\
\\
\frac{\mathcal{A} \vdash \exists x.F \quad \mathcal{A}, F[c] \vdash G}{\mathcal{A} \vdash G} \text{Choice} \quad \frac{\mathcal{A} \vdash F[t]}{\mathcal{A} \vdash \exists x.F} \exists I \\
\text{thereby in } \text{Choice } c \text{ is not allowed to occur in } G, \exists x.F, \text{ or in any of the formulas in } \mathcal{A}. \\
\\
\frac{\mathcal{A} \vdash F \quad \mathcal{A} \vdash F \Rightarrow G}{\mathcal{A} \vdash G} \Rightarrow E \quad \frac{\mathcal{A} \vdash \neg G \quad \mathcal{A} \vdash F \Rightarrow G}{\mathcal{A} \vdash \neg F} \text{MT}
\end{array}$$

All these rules change the complexity of the formulas they handle. On the one hand from premisses either more complicated formulas are constructed or subformulas are derived. On the other hand there is a set of rules that does not affect the complexity of their formulas. These rules represent logical equivalences. For example is  $\neg(F \vee G)$  equivalent to  $\neg F \wedge \neg G$  (and vice versa). Thus, we obtain the following two rules:

$$\frac{\mathcal{A} \vdash \neg(F \vee G)}{\mathcal{A} \vdash \neg F \wedge \neg G} \text{Taut} \quad \text{as well as} \quad \frac{\mathcal{A} \vdash \neg F \wedge \neg G}{\mathcal{A} \vdash \neg(F \vee G)} \text{Taut}$$

We call these rules *tautology rules*. All needed tautology rules are derived from the following set of pairs of equivalent formulas.

$\neg(F \vee G)$	$\neg F \wedge \neg G$
$\neg(F \wedge G)$	$\neg F \vee \neg G$
$\neg\neg F$	$F$
$\neg(F \Rightarrow G)$	$F \wedge \neg G$
$\neg\forall x.F$	$\exists x.\neg F$
$\neg\exists x.F$	$\forall x.\neg F$
$\neg(F \Leftrightarrow G)$	$\neg(F \Rightarrow G) \vee \neg(G \Rightarrow F)$
$F \Leftrightarrow G$	$(F \Rightarrow G) \wedge (G \Rightarrow F)$
$F \Rightarrow G$	$\neg F \vee G$
$\neg F \Rightarrow G$	$F \vee G$

The rule  $\neg\neg E$  can now also be interpreted as a tautology rule. The following is a short example of a proof in the ND-calculus.

*Example 2.4.2.*

$L_1.$	$L_1$	$\vdash \forall x.(P(x) \Rightarrow Q(x))$	$(Hyp)$
$L_2.$	$L_2$	$\vdash \forall y.P(y)$	$(Hyp)$
$L_3.$	$L_1$	$\vdash P(c) \Rightarrow Q(c)$	$(\forall E L_1)$
$L_4.$	$L_2$	$\vdash P(c)$	$(\forall E L_2)$
$L_5.$	$L_1, L_2$	$\vdash Q(c)$	$(\Rightarrow E L_3 L_4)$
$L_6.$	$L_1, L_2$	$\vdash \forall z.Q(z)$	$(\forall I L_5)$

This is a ND-proof of  $\forall z.Q(z)$  under the hypothesis  $\forall y.P(y)$  and  $\forall x.(P(x) \Rightarrow Q(x))$ . ■

Henceforth we use  $\mathcal{N}, \mathcal{N}_1, \mathcal{N}_2, \dots$  to range over ND-proofs. Furthermore, we identify henceforth ND-lines with their labels.

## 2.5 Transformation Problems

We extend the concept of ND-proofs to the concept of generalized ND-proofs by allowing proofs found by ATPs as justifications. This definition is taken from [Lin90].

### Definition 2.5.1 (Generalized ND-Proof).

A finite sequence  $S$  of proof lines is called a *generalized natural deduction proof* (*generalized ND-proof*) of a formula  $F$  from a set of hypothesis  $\mathcal{A}$ , if

- $F$  is the conclusion of the last line of  $S$
- $\mathcal{A}$  is the set of hypothesis of this last line
- every line is justified either by a rule of the ND-calculus (as given in the last section) or by a proof of its conclusion from its hypotheses in another proof formalism (we use refutation graphs).

Proof lines justified by ND-rules are called *closed lines*, lines justified by proofs in other formalisms are called *open lines* (open means, that they are still open, still to show in ND-calculus). A ND-proof without open lines is called *closed*. ■

Assume the following situation: We want to prove a theorem  $Th$  under a set of assumptions  $S_{Ass}$ . To prove this theorem we compute the clause normal form of the the formulas and call



an ATP on this clause set. The ATP provides us with a proof that we transform into a refutation graph  $G$ . Then we can give a generalized ND-proof for  $Th$ , consisting of the following lines: first the assumptions of  $S_{Ass}$  justified by  $Hyp$ , and then  $Th$  justified by  $G$ . We call this generalized ND-proof the *initial ND-proof*. The following is an initial ND-proof for the standard example.

*Example 2.5.2 (Generalized ND-proof for the Standard Example).*

$L_1.$	$L_1$	$\vdash \forall u.(u \circ e = u)$	$(Hyp)$
$L_2.$	$L_2$	$\vdash \forall w.(w^{-1} \circ w = e)$	$(Hyp)$
$L_3.$	$L_3$	$\vdash \forall x, y, z.((x \in S \wedge (y \in S \wedge y^{-1} \circ x = z)) \Rightarrow (z \in S))$	$(Hyp)$
$L_4.$	$L_1, L_2, L_3$	$\vdash \forall v.(v \in S \Rightarrow v^{-1} \in S)$	$(G)$

$G$  is the refutation graph in Figure 2.1. ■

If we have an initial ND-proof  $\mathcal{N}$  with a refutation graph  $G$  justifying the open line, then each clause of the refutation graph occurs in the clause normal form of one of the formulas of  $\mathcal{N}$ . Moreover, we can determine for each literal of each clause to which atomic subformula in the lines of our ND-proof it corresponds by clause normalization. Such a relationship is called a  $\Delta$ -relation (see also [Lin90]).

*Example 2.5.3 ( $\Delta$ -relation for the Standard Example).*

For the initial ND-proof of our standard example the  $\Delta$ -relation connects the unique literal of the clause  $[+ sk_1^{-1} \circ e = sk_1^{-1}]$  (this is the ground clause as used in the refutation graph  $G$  in Figure 2.1) with the atomic subformula  $u \circ e = u$  of line  $L_1$ . Furthermore, the second literal of clause  $[\perp sk_1 \in S, \perp sk_1 \in S, \perp sk_1^{-1} \circ sk_1 = e, + e \in S]$  is connected with the atomic subformula  $y \in S$  of line  $L_3, \dots$  ■

We use  $\Delta, \Delta', \Delta'', \dots, \Delta_1, \Delta_2, \dots$  to range over  $\Delta$ -relations. A delta-relation connects literals in clauses and atomic subformulas in ND-lines. This expresses also which clauses are in the CNF of which formula. Thus, we say henceforth say also that a  $\Delta$ -relation connects clauses and ND-lines.

**Definition 2.5.4 (Transformation Problem, Transformation Invariants).**

A *transformation problem* is a triple  $(\mathcal{N}, \mathcal{G}, \Delta)$  where

- $\mathcal{N}$  is a generalized ND-proof,
- $\mathcal{G}$  is a set  $\{G_1, \dots, G_m\}$  of refutation graphs, one for each open line in  $\mathcal{N}$ ,
- $\Delta$  is a  $\Delta$ -relation connecting all literals of the clauses of the refutation graphs in  $\mathcal{G}$  with atomic subformulas of a line in  $\mathcal{N}$ .

Furthermore, we demand the following conditions from a transformation problem:

1. Each refutation graph in  $\mathcal{G}$  is minimal.
2. No ND-line in  $\mathcal{N}$  contains free variables.
3.  $\Delta$  connects literals and atomic subformulas such that the clauses of the literals are in the CNF of the ND-lines of the atomic subformulas.

These conditions are called the *transformation invariants*. Henceforth, we refer to the transformation-invariants as *invariant 1*, *invariant 2*, and *invariant 3*.

An *initial transformation problem* is a triple  $(\mathcal{N}_i, \{G\}, \Delta_i)$  where  $\mathcal{N}_i$  is an initial ND-proof,  $G$  is the justification of the unique open line of  $\mathcal{N}_i$ , and  $\Delta_i$  is the  $\Delta$ -relation connecting the clauses and literals of  $G$  with the proof lines of  $\mathcal{N}_i$ .

An *empty transformation problem* is a triple  $(\mathcal{N}_c, \emptyset, \emptyset)$  with a closed ND-proof. ■

To guarantee that the initial transformation problem satisfies invariant 2 we have to demand that the theorem we want to prove and the assumptions contain no free variables.

With this definition we can describe the procedure of transforming a refutation graph into a closed ND-proof as follows: Starting with an initial transformation problem we have to reach an empty transformation problem with a closed ND-proof. Indeed, the transformation algorithms we describe in this report consist of the construction of sequences of transformation problems which start with an initial transformation problem and finish with an empty transformation problem. Thus, completeness proofs of these transformation algorithms are done by proving that an empty transformation proof is reached. The proofs of correctness of these transformation algorithms (or parts of them) are done by proving that (always) a transformation problem is produced from a transformation problem and that thereby correctly justified ND-lines are created.

## Chapter 3

# Transformation by Decomposition

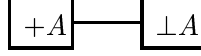
We present in this chapter a first transformation algorithm that starts with an initial transformation problem and computes via a sequence of intermediate transformation problems an empty transformation problem. In a transformation problem a refutation justifying an open line closes the gap between the hypotheses and the conclusion of this line. The formulas in an ND-proof – the conclusions of the ND-lines – can consist of various logical connectives and quantifiers. Accordingly, the ND-calculus consists of various ND-steps manipulating the different logical quantifiers and connectives. In contrast thereto, the formulas of a refutation graph – the clauses – are normalized and the steps in a refutation graph are all of the same kind: connections of contradictory literals. The transformation algorithm consists of two phases: (1) If the gap that is closed by the refutation graph is 'small' enough and if we have in the ND-lines also literals as in the refutation graph, then we can close the gap within the ND-calculus by transforming at the literal level a step (two contradictory literals) in the refutation graph into corresponding steps in the ND-proof. We call such base cases *literal base cases*. (2) If the gap is too big to be closed directly we use the refutation graph as a plan how to apply ND-rules that eliminate logical connectives and quantifiers and produce by applying so-called *transformation rules* [Lin90] step-by-step ND-lines with conclusions whose formulas are lesser complex. In such a manner, a complex case is reduced to a sequence of literal base cases.

In the first section we describe the transformation of the literal base cases. The decomposition by applications of transformation rules is described in the second section. The third section contains a detailed description how the instantiation of quantifications is handled; especially the role of skolem terms is explained. In the fourth section we combine the results from the first three sections to a complete transformation algorithm. Finally, we discuss in the last section some problems of this transformation algorithm.

Note that the content of the first and second section are mainly taken from [Lin90]. The content of the third section is new work presented first in this report.

### 3.1 The Literal Base Cases

Assume we have a transformation problem  $(\mathcal{N}, \{G_1, \dots, G_{i-1}, G_i, G_{i+1}, \dots, G_n\}, \Delta)$  where  $G_i$  is:



Via  $\Delta$  we can compute the ND-lines  $L_1$  and  $L_2$  that are connected with the clauses. If the formulas of these ND-lines are literal, then there are exactly two possible cases in our ND-proof since a transformation problem satisfies invariant 3:

1.

$$\begin{array}{llll} L_1. & \mathcal{A} & \vdash A & (\mathcal{R}) \\ L_2. & \mathcal{A} & \vdash A & (G) \end{array}$$

Clause  $[+A]$  is in the CNF of formula  $A$  of line  $L_1$  and clause  $[\perp A]$  is in the CNF of formula  $A$  of line  $L_2$  (open lines are negated for clause normalization). We can close the gap between  $L_1$  and  $L_2$  in the ND-calculus by justifying  $L_2$  with an application of the rule *Weaken* on  $L_1$ <sup>1</sup>.

$$\begin{array}{llll} L_1. & \mathcal{A} & \vdash A & (\mathcal{R}) \\ L_2. & \mathcal{A} & \vdash A & (\textit{Weaken } L_1) \end{array}$$

Similarly, if  $L_1$  and  $L_2$  contain both  $\neg A$  as formula we justify  $L_2$  also by rule *Weaken* on  $L_1$ .

2.

$$\begin{array}{llll} L_1. & \mathcal{A} & \vdash A & (\mathcal{R}) \\ L_2. & \mathcal{A} & \vdash \neg A & (\mathcal{R}) \\ L_3. & \mathcal{A} & \vdash F & (G) \end{array}$$

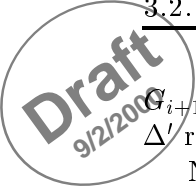
Here clause  $[+A]$  belongs to ND-line  $L_1$  and clause  $[\perp A]$  belongs to ND-line  $L_2$ . In this case we close the gap between  $L_1$ ,  $L_2$  and  $L_3$  by justifying  $L_3$  by applying rule *Indirect* on  $L_2$  and  $L_3$ .

$$\begin{array}{llll} L_1. & \mathcal{A} & \vdash A & (\mathcal{R}) \\ L_2. & \mathcal{A} & \vdash \neg A & (\mathcal{R}) \\ L_4. & \mathcal{A} & \vdash \perp & (\textit{Contradiction } L_1 L_2) \\ L_3. & \mathcal{A} & \vdash F & (\textit{Indirect } L_4) \end{array}$$

We call these two cases the *literal base cases* since the transformation of these cases is done at literal level: the link between the contradictory literals in the refutation graph is transformed into the application of a ND-rule connecting the corresponding ND-lines that contain literals as formulas.

Formally, we obtain by the *application* or *transformation of a literal base case* – with a refutation graph  $G_i$  as justification of an open line  $L$  – from a transformation problem  $(\mathcal{N}, \{G_1, \dots, G_{i-1}, G_i, G_{i+1}, \dots, G_n\}, \Delta)$  a new transformation problem  $(\mathcal{N}', \{G_1, \dots, G_{i-1},$

<sup>1</sup>The rule *Weaken* is not really a ND-rule. We allow to use it during the transformation, but when the transformation is finished the occurrences of this rule should be removed.



$G_{i+1}, \dots, G_n\}, \Delta')$ . Thereby,  $\mathcal{N}'$  results from  $\mathcal{N}$  by justifying  $L$  within the ND-calculus and  $\Delta'$  results from  $\Delta$  by removing all connections with clauses of  $G_i$ .

Note that we have a literal base case only if *both* the refutation graph and the corresponding ND-lines satisfy the necessary conditions. The refutation graph has to consists only of two linked unit clauses whose literals are connected with ND-lines that have also literals as formulas.

### 3.2 Decomposition by Transformation Rules

With the results of the last section we can transform literal base cases. The idea for a complete transformation algorithm is to reduce a more complex case into a set of literal base cases. We do this by decomposing refutation graphs and formulas in such ND-lines that are connected with clauses in the refutation graphs. Hence, the refutation graph serves as plan which ND-lines have to be decomposed. After each decomposition we want to reach again a transformation problem. To do so, we have to satisfy the transformation invariants. This demands that refutation graphs, ND-proof, and  $\Delta$ -relation are always changed simultaneously. These simultaneous changes are done by applying so-called *transformation rules* [Lin90].

Our goal by applying transformation rules is to obtain stepwise smaller refutation graphs whose literals are connected stepwise with ND-lines with lesser complex formulas. In the following we describe some transformation rules.

**TI $\wedge$ :** Assume we have a transformation problem  $(\mathcal{N}, \{G_1, \dots, G_n\}, \Delta)$  with a closed ND-line  $L$  in  $\mathcal{N}$ , such that  $L$  has the formula  $F \wedge H$  (thereby  $F$  and  $H$  have not to be atomic but can be arbitrary complex formulas). Furthermore,  $L$  is connected by  $\Delta$  with some clauses of a refutation graph  $G_i$  ( $1 \leq i \leq n$ ) which justifies an open ND-line of  $\mathcal{N}$ . To decompose  $F \wedge H$  we can use the ND-rules  $\wedge E_L$  and  $\wedge E_R$ .

$$L. \quad \mathcal{A} \quad \vdash F \wedge H \quad (\mathcal{R}) \quad \rightarrow \quad \left\{ \begin{array}{lll} L. & \mathcal{A} & \vdash F \wedge H \quad (\mathcal{R}) \\ L'. & \mathcal{A} & \vdash F \quad (\wedge E_L L) \\ L''. & \mathcal{A} & \vdash H \quad (\wedge E_R L) \end{array} \right.$$

But then we have to change  $\Delta$  too. The clauses previously connected with  $L$  become connected with the lines  $L'$  or  $L''$ , respectively. Thus, the clauses become connected with lesser complex formulas. Since  $F \wedge H$  is a  $\alpha$ -formula the CNF of  $L$  is the union of the CNFs of  $L'$  and  $L''$ . Therefore, the clauses themselves and the refutation graph  $G_i$  keep unchanged.  $L$  is not longer connected with any clause in  $G_i$ .

**TE $\wedge$ :** Assume we have a transformation problem  $(\mathcal{N}, \{G_1, \dots, G_n\}, \Delta)$  with an open ND-line  $L$  that has  $F \wedge H$  as formula and the refutation graph  $G_i$  ( $1 \leq i \leq n$ ) as justification. To decompose  $F \wedge H$  in this line we can use the ND-rule  $\wedge I$  backwardly.

$$L. \quad \mathcal{A} \quad \vdash F \wedge H \quad (G_i) \quad \rightarrow \quad \left\{ \begin{array}{lll} L'. & \mathcal{A} & \vdash F \quad (G'_i) \\ L''. & \mathcal{A} & \vdash H \quad (G''_i) \\ L. & \mathcal{A} & \vdash F \wedge H \quad (\wedge I L' L'') \end{array} \right.$$

Then we have not only to change  $\Delta$  by replacing occurrences of  $L$  by occurrences of  $L'$  or  $L''$ , respectively, but we have also to construct from the one refutation graph  $G_i$

Draft  
9/2/2000

two new graphs  $G'_i$  and  $G''_i$  justifying  $L'$  and  $L''$ , respectively. Since the formula of  $L$  is negated for clause normalization ( $L$  is an open line) we have with  $\neg(F \wedge H)$  a  $\beta$ -formula for clause normalization. Thus, the CNF of  $L$  is the cross product of the CNFs of  $L'$  and  $L''$  ( $CNF(\neg(F \wedge H)) = CNF(\neg F) \times CNF(\neg H)$ ). If we decompose  $F \wedge H$  into  $F$  and  $H$ , we have to break each clause in  $G_i$  connected with  $L$  into two clauses: one clause in  $CNF(\neg F)$  and one clause in  $CNF(\neg H)$ . This breaking of the clauses causes a breaking of the whole refutation graph  $G_i$  into two new refutation graphs  $G'_i$  and  $G''_i$  such that  $G'_i$  contains no clauses in  $CNF(\neg H)$  and  $G''_i$  contains no clauses in  $CNF(\neg F)$ .  $\Delta$  is changed such that the literals of the clauses produced by this breaking of  $F \wedge H$  are connected with  $L'$  and  $L''$ , respectively.  $L$  is not longer connected with any clause in  $G'_i$  or  $G''_i$ .

*TMInf*: Assume we have a transformation problem  $(\mathcal{N}, \{G_1, \dots, G_n\}, \Delta)$  with an open ND-line  $L_2$  justified by  $G_i$  ( $1 \leq i \leq n$ ) and a closed ND-line  $L_1$  with formula  $F \Rightarrow H$ . To decompose  $F \Rightarrow H$  we can use the ND-rule  $\Rightarrow E$  backwardly.

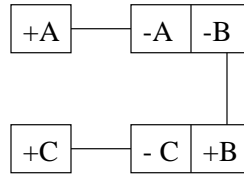
$$\begin{array}{lcl} L_1. & \mathcal{A} & \vdash F \Rightarrow H \quad (\mathcal{R}) \\ L_2. & \mathcal{A} & \vdash I \quad (G_i) \end{array} \rightarrow \left\{ \begin{array}{lcl} L_3. & \mathcal{A} & \vdash F \quad (G'_i) \\ L_1. & \mathcal{A} & \vdash F \Rightarrow H \quad (\mathcal{R}) \\ L_4. & \mathcal{A} & \vdash H \quad (\Rightarrow E \ L_3 \ L_2) \\ L_2. & \mathcal{A} & \vdash I \quad (G''_i) \end{array} \right.$$

Since the decomposed formula is a  $\beta$ -formula we have here a similar situation as in transformation rule  $TE\wedge$ .  $G_i$  is divided into  $G'_i$  and  $G''_i$  by breaking the clauses that are connected with  $L_1$  into two clauses, respectively, one clause in  $CNF(\neg F)$  ( $L_3$  is an open ND-line and therefore negated for clause normalization) and one clause in  $CNF(H)$ .  $\Delta$  is changed such that the clauses produced by this breaking of  $F \Rightarrow H$  become connected with  $L_3$  and  $L_4$ , respectively.  $L_1$  is not longer connected with any clause in  $G'_i$  or  $G''_i$ .

With these three rules we are able to decompose  $\wedge$ -formulae both in closed lines and open lines, and to decompose  $\Rightarrow$ -formulae in closed lines. We use them now to transform a simple example.

*Example 3.2.1.*

We want to prove that  $A \wedge B$  follows from  $A \wedge C$ , and  $C \Rightarrow B$ . By clause normalization we obtain the clauses  $[+A]$ ,  $[+C]$ ,  $[\neg A, \neg B]$ , and  $[\neg A, \neg B]$ . A refutation graph using these clauses is the following graph  $G$ :



We have the initial ND-proof  $\mathcal{N}$ :

$$\begin{array}{lll} L_1. & L_1 & \vdash A \wedge C \quad (Hyp) \\ L_2. & L_2 & \vdash C \Rightarrow B \quad (Hyp) \\ L_3. & L_1, L_2 & \vdash A \wedge B \quad (G) \end{array}$$



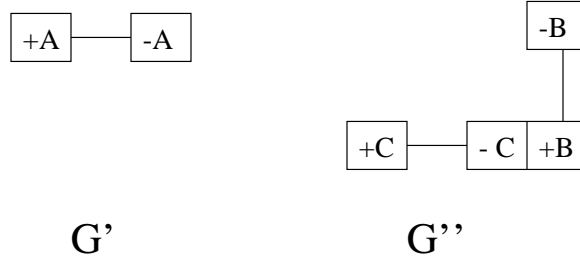
The delta-relation  $\Delta$  connects the literals of  $G$  and the formulae of  $\mathcal{N}$  in the following way:

- Literal  $+A$  of clause  $[+A]$  is connected with  $A$  in line  $L_1$ .
- Literal  $+C$  of clause  $[+C]$  is connected with  $C$  in line  $L_1$ .
- Literal  $\perp C$  of clause  $[\perp C, +B]$  is connected with the subformula  $C$  in line  $L_2$ .
- Literal  $+B$  of clause  $[\perp C, +B]$  is connected with the subformula  $B$  in line  $L_2$ .
- Literal  $\perp B$  of clause  $[\perp A, \perp B]$  is connected with the subformula  $B$  in line  $L_3$ .
- Literal  $\perp A$  of clause  $[\perp A, \perp B]$  is connected with the subformula  $A$  in line  $L_3$ .

By applying transformation rule  $TE\wedge$  on line  $L_3$  we obtain:

$L_1.$	$L_1$	$\vdash A \wedge C$	$(Hyp)$
$L_2.$	$L_2$	$\vdash C \Rightarrow B$	$(Hyp)$
$L_4.$	$L_1, L_2$	$\vdash A$	$(G')$
$L_5.$	$L_1, L_2$	$\vdash B$	$(G'')$
$L_3.$	$L_1, L_2$	$\vdash A \wedge B$	$(\wedge I L_4 L_5)$

where  $G'$  and  $G''$  arise from  $G$  by splitting the clause  $[\perp A, \perp B]$  connected with  $L_3$ .



$\Delta$  is changed by replacing the connections between  $L_3$  and  $[\perp A, \perp B]$  by connections between  $L_4, L_5$  and  $[\perp A], [\perp B]$ :

- Literal  $\perp A$  of clause  $[\perp A]$  is connected with  $A$  in line  $L_4$ .
- Literal  $\perp B$  of clause  $[\perp B]$  is connected with  $B$  in line  $L_5$ .

Note that  $G'$  is already the refutation graph of a literal base case, but the ND-lines connected with its clauses are not yet literal (since  $[+A]$  is connected with the subformula  $A$  in  $A \wedge C$ ). Therefore, we apply as next transformation rule  $TE\wedge$  on line  $L_1$ . We obtain:

$L_1.$	$L_1$	$\vdash A \wedge C$	$(Hyp)$
$L_6.$	$L_1$	$\vdash A$	$(\wedge E_L L_1)$
$L_7.$	$L_1$	$\vdash C$	$(\wedge E_R L_1)$
$L_2.$	$L_2$	$\vdash C \Rightarrow B$	$(Hyp)$
$L_4.$	$L_1, L_2$	$\vdash A$	$(G')$
$L_5.$	$L_1, L_2$	$\vdash B$	$(G'')$
$L_3.$	$L_1, L_2$	$\vdash A \wedge B$	$(\wedge I L_4 L_5)$

**Draft**  
9/2/2000

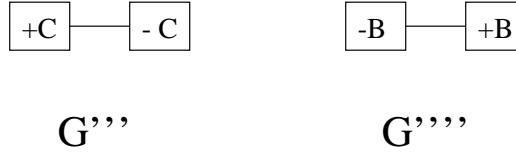
Neither  $G'$  nor  $G''$  are changed.  $\Delta$  is changed by replacing connections between  $L_1$  and  $[+A], [+C]$  by connections between  $L_6, L_7$  and  $[+A], [+C]$ :

- Literal  $+A$  of clause  $[+A]$  is connected with  $A$  in line  $L_6$ .
- Literal  $+C$  of clause  $[+C]$  is connected with  $B$  in line  $L_7$ .

Then we have a literal base case with  $G'$  and the ND-lines  $L_6$  and  $L_4$  that are connected with the clauses of  $G'$ . We delay its transformation until we have reached completely literal base cases. The only complex, non-literal line connected with clauses in the current refutation graphs is  $L_2$ . We can decompose it with transformation rule  $TMInf$ . The resulting ND-proof is:

$L_1.$	$L_1$	$\vdash A \wedge C$	$(Hyp)$
$L_6.$	$L_1$	$\vdash A$	$(\wedge E_L L_1)$
$L_7.$	$L_1$	$\vdash C$	$(\wedge E_R L_1)$
$L_8.$	$L_1, L_2$	$\vdash C$	$(G''')$
$L_2.$	$L_2$	$\vdash C \Rightarrow B$	$(Hyp)$
$L_9.$	$L_1, L_2$	$\vdash B$	$(\Rightarrow E L_8 L_2)$
$L_4.$	$L_1, L_2$	$\vdash A$	$(G')$
$L_5.$	$L_1, L_2$	$\vdash B$	$(G''')$
$L_3.$	$L_1, L_2$	$\vdash A \wedge B$	$(\wedge I L_4 L_5)$

$G'''$  and  $G''''$  result from  $G''$  by splitting the clause  $[\perp C, +B]$ .



In  $\Delta$  the connections between  $L_2$  and  $[\perp C, +B]$  are replaced by connections between  $L_8, L \perp 9$  and  $[\perp C], [+B]$ :

- Literal  $\perp C$  of clause  $[\perp C]$  is connected with  $C$  in line  $L_8$ .
- Literal  $+B$  of clause  $[+B]$  is connected with  $B$  in line  $L_9$ .

We have now a set of literal base cases and can close the remaining open lines how described in the last section:

$L_1.$	$L_1$	$\vdash A \wedge C$	$(Hyp)$
$L_6.$	$L_1$	$\vdash A$	$(\wedge E_L L_1)$
$L_7.$	$L_1$	$\vdash C$	$(\wedge E_R L_1)$
$L_8.$	$L_1, L_2$	$\vdash C$	$(Weaken L_7)$
$L_2.$	$L_2$	$\vdash C \Rightarrow B$	$(Hyp)$
$L_9.$	$L_1, L_2$	$\vdash B$	$(\Rightarrow E L_8 L_2)$
$L_4.$	$L_1, L_2$	$\vdash A$	$(Weaken L_6)$
$L_5.$	$L_1, L_2$	$\vdash B$	$(Weaken L_9)$
$L_3.$	$L_1, L_2$	$\vdash A \wedge B$	$(\wedge I L_4 L_5)$



If we remove the lines justified by *Weaken* we get the following closed ND-proof as result of the transformation process:

$L_1.$	$L_1$	$\vdash A \wedge C$	$(Hyp)$
$L_6.$	$L_1$	$\vdash A$	$(\wedge E_L L_1)$
$L_7.$	$L_1$	$\vdash C$	$(\wedge E_R L_1)$
$L_2.$	$L_2$	$\vdash C \Rightarrow B$	$(Hyp)$
$L_9.$	$L_1, L_2$	$\vdash B$	$(\Rightarrow E L_7 L_2)$
$L_3.$	$L_1, L_2$	$\vdash A \wedge B$	$(\wedge I L_6 L_9)$

■

We describe in the next section transformation rules that handle instantiations of quantified formulas. The introduction of the complete set of transformation problem we postpone to Appendix A. There we describe rules decomposing each logical connective and quantifier; at least one rule to decompose closed ND-lines and one rule to decompose open ND-lines are introduced.

In Example 3.2.1 it was quite easy and obvious how to break the refutation graphs. In general this breaking is more complex, since may several clauses have to be broken at once. We give a complete algorithm for breaking refutation graphs in the Appendix B. This algorithm produces from a minimal refutation graph refutation graphs that are minimal, too.

### 3.3 Handling of Instantiations

In this section we describe how ND-lines with quantified formulas are decomposed by appropriate transformation rules. Our approach explicitly supports the usage of skolem terms. Only with this extension the transformation concept presented in this report becomes applicable on proofs found by ATPs.

We do not want to search for a proof in the ND-calculus but we want to use the refutation graph as a plan how to construct a ND-proof. Hence, the refutation graph should also guide the applications of transformation rules decomposing quantified formulas. The refutation graphs is already a finished proof that contains instantiations for the quantified variables such that linked literals are contradictory. On the one hand, we obtain from  $\gamma$ -quantified variables by clause normalization free variables in the clauses. Since we express a ground clause in a refutation graph as a pair consisting of a clause with variables and a ground-substitution for these variables (see Section 2.3) we obtain the instantiations of the variables of a clause from the corresponding ground-substitutions. On the other hand,  $\delta$ -quantified variables become skolem functions by clause normalization. In the following we describe how these instantiations in the refutation graphs are transfered into instantiations in the ND-proof. We start with describing transformation rules to decompose closed quantified formulas.

$$TMChoice \quad \begin{array}{l} L_1. \quad \mathcal{A} \quad \vdash \exists x.F \quad (\mathcal{R}) \\ L_2. \quad \mathcal{A} \quad \vdash H \quad (G) \end{array} \rightarrow \left\{ \begin{array}{l} L_1. \quad \mathcal{A} \quad \vdash \exists x.F \quad (\mathcal{R}) \\ L_3. \quad L_2 \quad \vdash F[c] \quad (Hyp) \\ L_4. \quad \mathcal{A}, L_2 \quad \vdash H \quad (G') \\ L_2. \quad \mathcal{A} \quad \vdash H \quad (Choice L_1 L_4) \end{array} \right.$$

where  $c$  has to be a new constant.



$$TIV \quad \begin{array}{l} L_1. \quad \mathcal{A} \quad \vdash \forall x.F \quad (\mathcal{R}) \\ L_2. \quad \mathcal{A} \quad \vdash H \quad (G) \end{array} \rightarrow \left\{ \begin{array}{l} L_1. \quad \mathcal{A} \quad \vdash \forall x.F \quad (\mathcal{R}) \\ L_3. \quad \mathcal{A} \quad \vdash F[t] \quad (\forall E) \\ L_2. \quad \mathcal{A} \quad \vdash H \quad (G') \end{array} \right.$$

where  $t$  is a ground term.

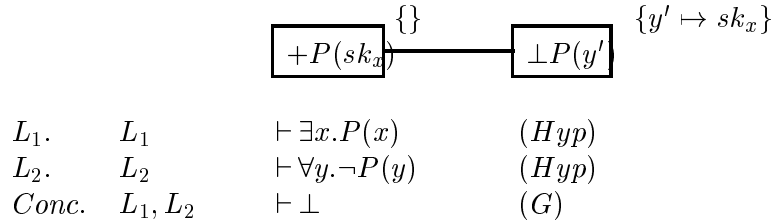
We demand that the  $t$  in  $TIV$  is a ground term to guarantee invariant 2. The condition in  $TMChoice$ , that  $c$  is a new constant, is necessary since we apply the ND-rule *Choice*

$$\frac{\mathcal{A} \vdash \exists x.F \quad \mathcal{A}, F[c] \vdash G}{\mathcal{A} \vdash G} \textit{Choice}$$

which demands that  $c$  occurs neither in  $G$  nor in  $\exists x.F$  nor in one of the formulas in  $\mathcal{A}$  (this condition is also called the *Eigenvariable condition*). The introduction of a new constant guarantees that the application of  $TMChoice$  is correct. There are also corresponding transformation rules to decompose open quantified ND-lines. We omit to introduce them here, since with  $TMChoice$  and  $TIV$  we have already rules decomposing both  $\gamma$ - and  $\delta$ -quantified formulas. The handling of the transformation rules that decomposing open quantified ND-lines is similar to the handling of these two rules, respectively. Furthermore, we omit at the moment to describe in the transformation rules the effects on the refutation graphs (indicated by the change from  $G$  to  $G'$ ) and on the  $\Delta$ -relations caused by the application of  $TIV$  and  $TMChoice$ . We continue with a small example that illustrates the problems that arise when we want to apply these two transformation rules.

#### Example 3.3.1.

Assume we have the following refutation graph  $G$  and initial ND-proof  $\mathcal{N}$ :



Furthermore, a delta-relation  $\Delta$  connects the literal  $+P(sk_x)$  of clause  $[+P(sk_x)]$  with the literal subformula  $P(x)$  in  $L_1$  and the literal  $\perp P(y')$  of clause  $[\perp P(y')]$  with the literal subformula  $P(y)$  in  $L_2$ . Intuitively spoken, we need to instantiate  $x$  and  $y$  such that  $P(x)$  and  $P(y)$  in the ND-proof become equal. But then the following two questions arise:

1. Should we first apply  $TIV$  on line  $L_2$  or first  $TMChoice$  on  $L_1$  or is the order of the applications equal?
2. In the application of  $TMChoice$  (when ever) the choice of  $c$  is straight forward: an arbitrary new constant. But which ground term  $t$  should we use to instantiate  $y$  in the application of  $TIV$ ?

Assume we would apply as first  $TIV$  with an arbitrary ground term  $t$  on line  $L_2$ . Then we would obtain:

$L_1.$	$L_1$	$\vdash \exists x.P(x)$	$(Hyp)$
$L_2.$	$L_2$	$\vdash \forall y.\neg P(y)$	$(Hyp)$
$L_3.$	$L_2$	$\vdash \neg P(t)$	$(\forall E L_2)$
$Conc.$	$L_1, L_2$	$\vdash \perp$	$(W')$

But if we apply then *TMChoice* on  $L_1$  we can not obtain  $P(t)$  since even if  $t$  would be a constant it would not be a new constant (since already used in  $L_3$ ). In this simple example the right sequence of *TMChoice* and *TI* $\forall$  and the right choice for  $t$  in *TI* $\forall$  are obviously to apply first *TMChoice* on  $L_1$  using an arbitrary new constant  $c$  and then to apply *TI* $\forall$  on  $L_2$  using  $c$  as ground term.

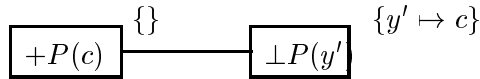
$L_1.$	$L_1$	$\vdash \exists x.P(x)$	$(Hyp)$
$L_2.$	$L_2$	$\vdash \forall y.\neg P(y)$	$(Hyp)$
$L_3.$	$L_3$	$\vdash P(c)$	$(Hyp)$
$L_5.$	$L_2$	$\vdash \neg P(c)$	$(\forall E L_2)$
$L_4.$	$L_1, L_2, L_3$	$\vdash \perp$	$(G')$
$Conc.$	$L_1, L_2$	$\vdash \perp$	$(Choice L_1 L_4)$

Since we reached then literal base cases we can close line  $L_4$  by transforming the literal base case. ■

In this small example it was quite easy to determine the order in that we have to apply the transformation rules and to determine the term  $t$  that we have to use in the application of the rule *TI* $\forall$ . Nevertheless we use this small example now to developed a general concept how to order the applications of the transformation rules and how to choose the terms for the instantiation.

If we compare in Example 3.3.1 the initial refutation graph with the lines of the ND-proof we see that the existentially quantified ( $\delta$ -quantified) variable  $x$  in the closed  $L_1$  corresponds to a skolem function  $sk_x$  in the refutation graph. When we instantiate  $\exists xP(x)$  by an application of the transformation rule *TMChoice*, we have to introduce a new constant  $c$  in the ND-proof and obtain the new ND-line  $L_3$  with formula  $P(c)$ . We have to change the  $\Delta$ -relation such that afterwards the clause  $[+P(sk_x)]$  in the refutation graph is connected with the new line  $L_3$  instead of  $L_1$  (since by the application of transformation rules we want to obtain less complex ND-lines connected with the clauses). But then we have to change the refutation graph, too, to satisfy invariant 3 and to obtain again a transformation problem after the application of *TMChoice* ( $[+P(sk_x)]$  is not in the CNF of  $P(c)$ ). Therefore, we replace during the application of *TMChoice* each occurrence of  $sk_x$  in the refutation graph by an occurrence of  $c$  (in both the clauses *and* the ground substitutions). The resulting clause  $[+P(c)]$  is then in the CNF of  $P(c)$ .

We obtain as new refutation graph:

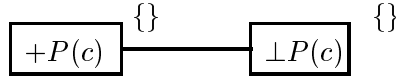


If a closed ND-line with an universally quantified ( $\gamma$ -quantified) formula is clause normalized the quantified variable  $x$  becomes in the clauses a variable  $x'$ . Since a refutation graph consists of ground clauses each occurrence of such a clause used in a refutation graph together with a ground substitution  $\sigma$  that maps all variables of the clause to ground terms, for

instance  $x' \mapsto t'$ . When we apply  $TIV$  to decompose an universally quantified ND-line with variable  $x$  we want to use directly this term  $t'$ . But it can happen that  $t'$  contains skolem terms. To use skolem terms is a concept typical for refutation based automated theorem proving. Therefore, we do not want to introduce skolem terms into our ND-proofs and forbid to apply  $TIV$  with  $t'$  if  $t'$  contains skolem terms.

If we compare these results with Example 3.3.1 we see that we tried to apply the transformation rules first in the wrong order. We tried to apply first  $TIV$  to instantiate the  $\gamma$ -quantified variable  $y$ . But the term that corresponds to  $y$  in the ground substitution is  $sk_x$ . During the application of  $TMChoice$   $sk_x$  is replaced in the ground substitution by  $c$ . This  $c$  can be used in an application of  $TIV$  to instantiate  $y$ .

During the application of  $TIV$  on  $L_2$  we have similar effects concerning invariant 3 like during the application of  $TMChoice$ . When we apply  $TIV$  on  $L_2$  we obtain the new ND-line  $L_5$  with formula  $\neg P(c)$ . The clause  $[\perp P(y)]^{\{y \mapsto c\}}$  should become connected with  $L_5$  afterwards, but the clause  $[\perp P(y')]$  is not in CNF of the formula  $\neg P(c)$ . We have to instantiate the variable  $y'$  in the clauses connected with  $L_2$  by applying the ground instantiation  $\{y' \mapsto c\}$  on these clauses. Thereby we obtain the clause  $[\perp P(c)]$  which is in the CNF of formula  $\neg P(c)$ . As refutation graph we obtain.



We can now give complete formalizations of  $TIV$  and  $TMChoice$  (including the effects on refutation graphs and  $\Delta$ -relations):

*TMChoice*

$$\begin{array}{lcl} L_1. \mathcal{A} & \vdash \exists x.F & (\mathcal{R}) \\ L_2. \mathcal{A} & \vdash H & (G) \end{array} \rightarrow \left\{ \begin{array}{lcl} L_1. \mathcal{A} & \vdash \exists x.F & (\mathcal{R}) \\ L_3. L_2 & \vdash F[c] & (Hyp) \\ L_4. \mathcal{A}, L_2 & \vdash H & (G') \\ L_2. \mathcal{A} & \vdash H & (Choice\ L_1\ L_4) \end{array} \right.$$

where  $c$  has to be a new constant.

$G'$  results from  $G$  by replacing in both the clauses and the ground substitutions the skolem term that corresponds to  $x$  by  $c$ .

We change the  $\Delta$ -relation such that all clauses previously connected with  $L_1$  are afterwards connected with  $L_3$ .

*TIV*

$$\begin{array}{lcl} L_1. \mathcal{A} & \vdash \forall x.F & (\mathcal{R}) \\ L_2. \mathcal{A} & \vdash H & (G) \end{array} \rightarrow \left\{ \begin{array}{lcl} L_1. \mathcal{A} & \vdash \forall x.F & (\mathcal{R}) \\ L_3. \mathcal{A} & \vdash F[t] & (\forall E) \\ L_2. \mathcal{A} & \vdash H & (G') \end{array} \right.$$

where  $t$  has to be a ground term. Furthermore,  $t$  has to occur in the ground substitution of some clauses of  $G$  that are connected with  $L_1$  such that the ground substitution contains a pair  $x' \mapsto t$  ( where  $x'$  is the renaming of  $x$  in the clause).

$G'$  results from  $G$  by applying at each clause which is connected to  $L_1$  and whose ground substitution contains the pair  $x' \mapsto t$  the substitution  $\{x' \mapsto t\}$  and by removing the



pair  $x' \mapsto t$  from the ground instantiation.

We change the  $\Delta$ -relation such that the changed clauses (which were previously connected with  $L_1$  and contain a pair  $x' \mapsto t$  in their ground substitution) are afterwards connected with  $L_3$ .

Both rules *TI* and *TMChoice* change significantly the refutation graphs. However, for both rules holds that we reach after their application again a transformation problem satisfying the transformation invariants. We omit to give a technical proof thereof but we explain the key points in the following two remarks:

1. During the application of *TI* on an universally quantified formula the arity of the skolem functions that correspond to  $\delta$ -quantified subformulas of the universally quantified formula is decreased. Assume we have a formula  $\forall x. \exists y. P(x, y)$  in a closed ND-line. Then we obtain by clause normalization the clause  $[+P(x', sk_y(x'))]$ . This clause can be used in a refutation graph with a ground substitution  $\{x' \mapsto t'\}$  where we assume that  $t$  contains no further skolem terms. By applying *TI* with term  $t$  on  $\forall x. \exists y. P(x, y)$  we obtain a new line with formula  $\exists y. P(t, y)$  connected with a clause  $[+P(t, sk_y(t))]$ . Although the function  $sk_y$  itself has arity 1 it is used now with a fix argument  $t$  such that we can regard  $sk_y(t)$  altogether as a skolem function with arity 0. Hence, the arity of the skolem functions is decreased such that  $[+P(t, sk_y(t))]$  is in the CNF of formula  $\exists y. P(t, y)$  and we satisfy invariant 3.
2. Applications of both *TI* and *TMChoice* produce again refutation graphs. Since the structure of the graph itself is not changed the critical point is whether after the application of *TI* and *TMChoice* still contradictory literals are linked. The partial application of the ground substitution of some clauses during the application of *TI* does not endanger this property of a refutation graph since the ground clauses in the graph keep the same; we only move parts from the ground substitution into the clauses themselves. That the replacements done during an application of *TMChoice* do not destroy the refutation graph we guarantee by our clause normalization algorithm (see Section 2.2). This clause normalization algorithm guarantees that skolem functions depend exactly on the  $\gamma$ -quantified variables in whose scope their corresponding  $\delta$ -quantified subformulas are. Thus, if we reach (maybe after some applications of transformation rules) in the ND-proof a  $\delta$ -quantified formula, then its corresponding skolem function has arity 0 (since all  $\gamma$ -quantified variables in whose scope the  $\delta$ -quantified formula maybe was, have already been instantiated, compare with the first remark). Therefore, the replacement in the refutation graph as described in rule *TMChoice* replaces only such skolem terms that are ground and contain no further skolem terms as subterms (\*). This guarantees that after the decomposition of the  $\delta$ -quantified variable still contradictory literals are linked.

We could not guarantee this if we would replace skolem terms not satisfying (\*). An example is the following situation: The two literals  $\perp P(x)$  (with ground substitution  $\{x \mapsto sk_1(sk_2)\}$ ) and  $+P(sk_1(y))$  (with ground substitution  $\{y \mapsto sk_2\}$ ) are linked. If we would replace (how ever)  $sk_1(sk_2)$  by a new constant  $c$ , we would obtain the linked literals  $\perp P(x)$  (with ground substitution  $\{x \mapsto c\}$ ) and  $+P(sk_1(y))$  (with ground substitution  $\{y \mapsto sk_2\}$ ). But these (ground) literals are not longer contradictory. If we would replace (how ever)  $sk_1(y)$  by  $c$  we would obtain the linked literals  $\perp P(x)$  (with

Draft  
9/2/2000

ground substitution  $\{x \mapsto sk_1(sk_2)\}$  and  $+P(c)$  (with ground substitution  $\{y \mapsto sk_2\}$ ). Again these resulting (ground) literals are not contradictory.

Note that we have to apply the rule  $TI\forall$  maybe several times on one ND-line. Assume we have an universally quantified formula  $\forall x.P(x)$ . Then by clause normalization we obtain a corresponding clause  $[+P(x')]$  with a free variable  $x'$ . This clause can be used in a refutation graph with several ground substitutions. For example if we need  $P(a)$  and  $P(b)$  we have two occurrences of  $[+P(x')]$  one with ground substitution  $\{x' \mapsto a\}$  and one with ground substitution  $\{x' \mapsto b\}$ . Then we have to apply the rule  $TI\forall$  twice, one time with ground term  $a$  and one time with ground term  $b$ .  $TI\forall$  is different to other rules since after its application with one ground term  $t$  not all clauses previously connected with the quantified formula are afterwards connected with the decomposed formula.

The order of the application of the transformation rules  $TI\forall$  and  $TMChoice$  is partially determined by the restriction that  $\gamma$ -quantified ND-lines can not be decomposed if the corresponding term  $t$  in the ground substitution contains skolem terms. This restriction is called the  $\gamma$ -restriction. If the decomposition of a  $\gamma$ -quantified ND-line is blocked by the  $\gamma$ -restriction we have first to decompose some other ND-lines that contain the existentially quantified formulas from whose clause normalization the skolem terms arise. Applications of  $TMChoice$  replace the skolem terms by constants. If all skolem terms in  $t$  are replaced we can apply the rule  $TI\forall$  with a skolem-free  $t'$ . Since the  $\gamma$ -restriction forbids some applications of transformation rules the question arises whether we can reach a situation such that we can apply no transformation rule since all lines are blocked by the  $\gamma$ -restriction. We prove in the following lemma that this case can not happen.

**Lemma 3.3.2.**

*During the transformation process it can not happen that the decomposition of all lines that are connected with clauses of the refutation graphs are blocked by the  $\gamma$ -restriction.*

**Proof:** Assume that the decomposition of all ND-lines  $L_1, \dots, L_n$  connected with some refutation graphs are blocked by the  $\gamma$ -restriction.

Then all ND-lines  $L_1, \dots, L_n$  are  $\gamma$ -quantified formulas  $q_1x_1.R[x_1], \dots, q_nx_n.R[x_n]$  (\*).

Let  $T_{SK}$  be the set of all terms  $t$  in the ground substitutions of the clauses connected with these formulas, such that  $t$  occurs in a pair  $x'_i \mapsto t$  ( $1 \leq i \leq n$ ).

Because of our proof assumption all these terms contain skolem terms.

Let  $T_{SK}^*$  be the union of  $T_{SK}$  and the set of all subterms of terms in  $T_{SK}$ .

Since a refutation graph is a finite object both  $T_{SK}$  and  $T_{SK}^*$  are finite and since the relation 'is subterm of and begins with a skolem function' is a partial order on terms, exists a minimal term  $t \in T_{SK}^*$  (with respect to this order).

$t$  has two properties: (1) it begins with a skolem function  $f_{sk}$  and (2) it does not contain any other subterms that contain other skolem functions (\*\*) (otherwise  $t$  would not be minimal). The skolem function  $f_{sk}$  arises during clause normalization of a  $\delta$ -quantified subformula  $qyP[y]$  of the formula of a line  $L_i$  ( $1 \leq i \leq n$ ).

Since  $L_i$  has as formula the  $\gamma$ -quantified formula  $q_nx_i.R[x_i]$  (see (\*))  $\exists yP[y]$  is at least in the scope of one  $\gamma$ -quantified variable, namely  $x_i$ .

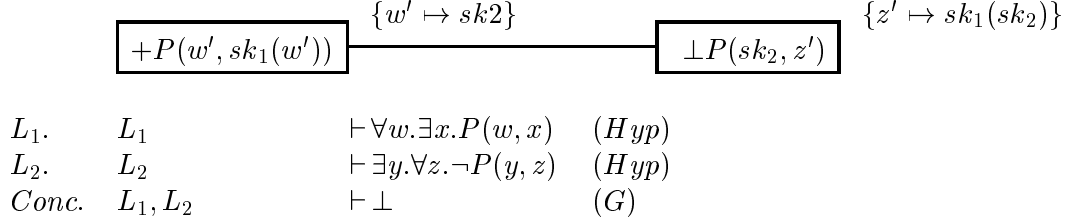
Thus  $f_{sk}$  has at least arity 1 and  $t$  has the following form:  $t = f_{sk}(t', \dots)$ . Thereby,  $t'$  is a subterm of  $t$  and can not contain a skolem term (see (\*\*)) (\*\*\*) .

But then the usage of  $t'$  is not blocked by the  $\gamma$ -restriction as a possible instantiation for  $L_i$ . This is a contradiction to our assumption.  $\square$

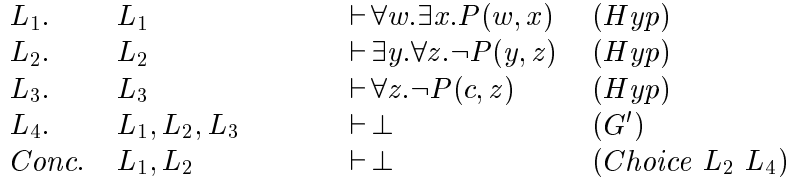
We finish this section with another example that demonstrates how the  $\gamma$ -restriction orders the instantiation steps.

*Example 3.3.3.*

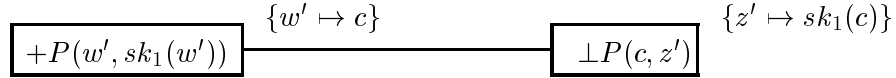
Assume we have the following refutation graph  $G$  with corresponding initial ND-proof  $\mathcal{N}$ :



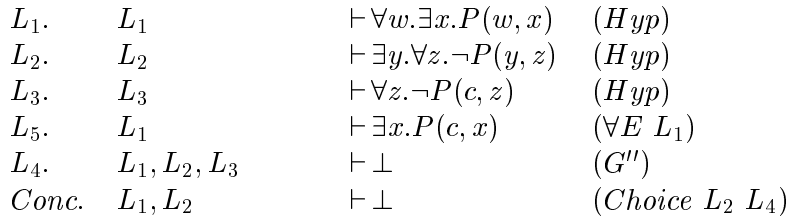
The left clause is connected by the  $\Delta$ -relation with  $L_1$  and the right clause with  $L_2$ . The  $\gamma$ -restriction forbids to apply  $TIV$  on  $L_1$ , since we would have to use as ground term  $t$  the skolem term  $sk_2$ . Therefore, we apply first  $TMChoice$  on  $L_2$  and obtain:



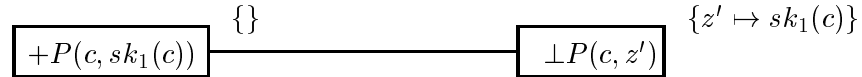
where  $G'$  is



Now the  $\gamma$ -restriction forbids the application of  $TIV$  on line  $L_3$ . But we can apply  $TIV$  on  $L_1$  with ground term  $c$  and obtain:



where  $G''$  is



The  $\gamma$ -restriction still forbids to apply  $TIV$  on  $L_3$  but we can apply  $TMChoice$  on  $L_5$ . Afterwards also  $TIV$  on  $L_3$  is applicable and we obtain a literal base case. As final ND-proof we obtain after the transformation of this literal base case:

$L_1.$	$L_1$	$\vdash \forall w. \exists x. P(w, x)$	$(Hyp)$
$L_2.$	$L_2$	$\vdash \exists y. \forall z. \neg P(y, z)$	$(Hyp)$
$L_3.$	$L_3$	$\vdash \forall z. \neg P(c, z)$	$(Hyp)$
$L_5.$	$L_1$	$\vdash \exists x. P(c, x)$	$(\forall E \ L_1)$
$L_6.$	$L_6$	$\vdash P(c, c')$	$(Hyp)$
$L_8.$	$L_3$	$\vdash \neg P(c, c')$	$(\forall E \ L_3)$
$L_7.$	$L_1, L_2, L_3, L_6$	$\vdash \perp$	$(Contradiction \ L_8 \ L_6)$
$L_4.$	$L_1, L_2, L_3$	$\vdash \perp$	$(Choice \ L_5 \ L_7)$
<i>Conc.</i>	$L_1, L_2$	$\vdash \perp$	$(Choice \ L_2 \ L_4)$

■

### 3.4 The Literal Transformation-Algorithm

In this section we combine the results of the last three sections to a complete transformation algorithm. This algorithm consists of two phases. First we apply transformation rules to decompose the refutation graphs and the ND-lines connected with the clauses of the refutation graphs until we reach literal base cases. Then we transform these literal base cases.

#### Algorithm 3.4.1 (Literal Transformation Algorithm).

Let  $G$  be a minimal refutation graph,  $\mathcal{N}$  the according initial ND-proof, and  $\Delta$  a delta-relation between them such that  $(\mathcal{N}, \{G\}, \Delta)$  is an initial transformation problem.

**decomposition:** Let  $(\mathcal{N}', \{G_1, \dots, G_n\}, \Delta')$  be the current transformation problem. Compute the set  $S$  of all ND-lines in  $\mathcal{N}'$  that are connected by  $\Delta'$  with at least one clause of any refutation graph  $G_1, \dots, G_n$  and whose formula is not literal. If  $S$  is not empty choose a ND-line  $L \in S$  such that  $L$  is not blocked by the  $\gamma$ -restriction and apply an appropriate transformation rule to decompose  $L$ . Continue this process until  $S$  is empty.

**literal base cases:** Let  $(\mathcal{N}'', \{G_1, \dots, G_n\}, \Delta'')$  the current transformation problem. If the set  $S$  that we described in the decomposition step is empty we have reached completely literal base cases in the refutation graphs and the ND-lines connected with the clauses of the refutation graphs. We transform these literal base cases to obtain a closed ND-proof.

■

#### Theorem 3.4.2 (Correctness and Completeness of the Literal Trans. Alg.).

Let  $G$  be a minimal refutation graph,  $\mathcal{N}$  the according initial ND-proof, and  $\Delta$  a delta-relation between them such that  $(\mathcal{N}, \{G\}, \Delta)$  is an initial transformation problem. Then the literal transformation algorithm 3.4.1 computes for this initial transformation problem a sequence of transformation problem that terminates with an empty transformation problem with a closed ND-proof.

**Proof:**

**Correctness:** We have to prove that we obtain – if the algorithm terminates with an empty transformation problem – a correct ND-proof. The correctness of the algorithm follows



from the fact that each single application of a transformation rule and each single application of literal base case produces proof lines and justifications that are correct in the sense of the ND-calculus as introduced in Section 2.4. Formally, the proof consist of an induction over the number of applied transformation rules and literal base cases.

**Termination:** By the construction of the transformation rules we obtain by each application of a transformation rules lesser complex refutation graphs whose clauses are connected with lesser complex formulas (even  $TI\forall$  has to be applied on one ND-line only finitely often). Since the literal base cases are the simplest possible refutation graph and the simplest corresponding ND-lines we have to reach after a finite number of transformation rule applications literal base cases. Successively, we can then transform these literal bases.

**Completeness:** For the completeness we have to prove that we reach indeed an empty transformation problem. That we cannot reach a blocked situation follows from the following facts: (1) For each chosen line  $L \in S$  in the decomposition step there exists an appropriate transformation rule in the set of standard transformation rules that is defined in Appendix A.<sup>2</sup> (2) The only restriction on the choice of  $L$  is that  $L$  is not blocked by the  $\gamma$ -restriction. But in Lemma 3.3.2 we proved that the  $\gamma$ -restriction cannot block each line in  $S$ . (3) We can transform each possible literal base case (see Section 3.1).

□

Note that the correct applicability of the transformation rules and the literal base cases is guaranteed by the transformation invariants. The transformation rules are constructed such that we obtain by applying them a sequence of transformation problems, each satisfying the transformation invariants. These invariants guarantee that the next transformation rule or literal base case can be applied:

**Invariant 1:** guarantees that we have always minimal refutation graphs. Informally spoken, this guarantees that our graphs contain no unnecessary parts. Formally, this property is needed to reach by decomposition indeed refutation graphs that consisting only of (exactly) two linked unit clauses whose literals are contradictory. Without invariant 1 we maybe would obtain not-minimal refutation graphs consisting of more clauses. Then the application of the literal base cases would be more complex.

**Invariant 2:** guarantees that no formula of a ND-line contains free variables. This property is needed to guarantee together with invariant 3 that we can transfer the instantiations of the variables of the clauses in the refutation graphs (stated in the ground substitutions of the clauses) into corresponding instantiations for the quantified ND-lines. Without invariant 2 it would be possible that ND-lines contain free variables. These variables would be also variables in the clauses corresponding to these ND-lines by clause normalization. Thus, the variables in these clauses would be mapped to ground terms by the ground substitutions in the refutation graphs. But we could not transfer these instantiations from the refutation graphs into the ND-proof since the corresponding variables

---

<sup>2</sup>In Appendix A we divide the transformation rules into two sets, the *standard rules* and the *alternative rules*. Note that already the standard rules contain enough rules that the literal transformation algorithm is complete. How and when we should use alternative rules instead of standard rules is described in the next section and in Appendix A.



in the ND-proof are free and not  $\gamma$ -quantified so that we could not apply transformation rules handling the instantiations in the ND-proof.

**Invariant 3:** guarantees that always such clauses and ND-lines are linked by a  $\Delta$ -relation such that the clauses are in the CNF of the ND-lines. This property guarantees that the literal base cases and the transformation rules are correctly applicable.

A literal base case is applicable since invariant 3 guarantees that the literals in the refutation graphs are connected with corresponding ND-lines with literals as formulas in the ND-proof. This guarantees that we can close the gap in the ND-proof between (exactly) these ND-lines as described in Section 3.1. Without invariant 3 it would be not possible to compute these ND-lines which that we need to transfer the link in the refutation graph into some steps in the ND-proof.

A transformation rule decomposes simultaneously a refutation graph and a ND-line. Invariant 3 guarantees that these changes can be coordinated correctly. The transformation rules can decompose corresponding clauses and ND-lines. These guarantees that after the application of a transformation rule the refutation graph is indeed a proof of the ND-line that is justified by it.

We apply now the literal transformation algorithm to compute a ND-proof of our standard example.

*Example 3.4.3 (Standard Example (Continuation)).*

We presented the initial refutation graph  $G$  in Figure 2.1, the initial ND-proof  $\mathcal{N}$  in Example 2.5.2, and the initial  $\Delta$ -relation  $\Delta$  in Example 2.5.3 (to construct the rest of the delta-relation we left to the reader). Altogether,  $G, \mathcal{N}$ , and  $\Delta$  form the initial transformation problem  $(\mathcal{N}, \{G\}, \Delta)$ .

As first  $S$  consists of the ND-lines  $L_1, L_2, L_3$  and  $L_4$ . The  $\gamma$ -restriction blocks all applications of the transformation rule  $TI\forall$  on  $L_1, L_2$ , and  $L_3$  (except one application of  $TI\forall$  on  $L_3$  with term  $e$  for  $x$  that we ignore at the moment). Therefore, we choose as first  $L_4$  and apply the transformation rule  $TE\forall$  on it; as result we get:

$L_1.$	$L_1$	$\vdash \forall u.(u \circ e = u)$	(Hyp)
$L_2.$	$L_2$	$\vdash \forall w.(w^{-1} \circ w = e)$	(Hyp)
$L_3.$	$L_3$	$\vdash \forall x, y, z.((x \in S \wedge (y \in S \wedge y^{-1} \circ x = z))$ $\Rightarrow (z \in S))$	(Hyp)
$L_5.$	$L_1, L_2, L_3$	$\vdash a \in S \Rightarrow a^{-1} \in S$	( $G'$ )
$L_4.$	$L_1, L_2, L_3$	$\vdash \forall v.(v \in S \Rightarrow v^{-1} \in S)$	( $\forall I$ $L_5$ )

The application of  $TE\forall$  replaces in the refutation graph  $G$  the only occurring skolem function  $sk_1$  by the new constant  $a$ . Afterwards, we have no further skolem terms in the ground substitutions such that henceforth the *gamma*-restriction cannot further block any applications of transformation rules. Therefore, we can now apply on the lines  $L_1, L_2$  and  $L_3$  all needed applications of rule  $TI\forall$ . Note that  $L_3$  has to be decomposed twice, since there are two occurrences of its corresponding clause  $[\perp(x_1 \in S), \perp(y_1 \in S), \perp(y_1^{-1} \circ x_1 = z_1), +(z_1 \in S)]$  in the refutation graph, one with the ground substitution  $\{x_1 \mapsto sk_1, y_1 \mapsto sk_1, z_1 \mapsto e\}$  and one with the ground substitution  $\{x_1 \mapsto e, y_1 \mapsto sk_1, z_1 \mapsto sk_1^{-1}\}$ .

$L_1.$	$L_1$	$\vdash \forall u.(u \circ e = u)$	$(Hyp)$
$L_6.$	$L_1$	$\vdash a^{-1} \circ e = a^{-1}$	$(\forall E L_1)$
$L_2.$	$L_2$	$\vdash \forall w.(w^{-1} \circ w = e)$	$(Hyp)$
$L_7.$	$L_2$	$\vdash a^{-1} \circ a = e$	$(\forall E L_2)$
$L_3.$	$L_3$	$\vdash \forall x, y, z.((x \in S \wedge (y \in S \wedge y^{-1} \circ x = z)) \Rightarrow (z \in S))$	$(Hyp)$
$L_8.$	$L_3$	$\vdash (a \in S \wedge (a \in S \wedge a^{-1} \circ a = e)) \Rightarrow (e \in S)$	$(3 * \forall E L_3)$
$L_9.$	$L_3$	$\vdash (e \in S \wedge (a \in S \wedge a^{-1} \circ e = a^{-1})) \Rightarrow (a^{-1} \in S)$	$(3 * \forall E L_3)$
$L_5.$	$L_1, L_2, L_3$	$\vdash a \in S \Rightarrow a^{-1} \in S$	$(G''')$
$L_4.$	$L_1, L_2, L_3$	$\vdash \forall v.(v \in S \Rightarrow v^{-1} \in S)$	$(\forall I L_5)$

Now  $S$  contains the ND-lines  $L_5, L_8$ , and  $L_9$ . We choose  $L_5$  and decompose it by an application of the transformation rule  $TE \Rightarrow$ . Afterwards, we apply on  $L_8$  and  $L_9$   $TMinf$ , respectively. Whereas by the application of  $TE \Rightarrow$  on  $L_5$  ND-lines with literal formulas arise ( $L_{10}$  and  $L_{16}$  in the following ND-proof), by the applications of  $TMinf$  ND-lines arise with conjunctions as formulas ( $L_{12}$  and  $L_{15}$  in the following ND-proof). On these lines we have to apply  $TE \wedge$ . Finally, we reach literal base cases that we transform to close all remaining open ND-lines. After removing the ND-lines justified with *Weaken* we obtain the following closed ND-proof:

$L_{10}.$	$L_{10}$	$\vdash a \in S$	$(Hyp)$
$L_1.$	$L_1$	$\vdash \forall u.(u \circ e = u)$	$(Hyp)$
$L_6.$	$L_1$	$\vdash a^{-1} \circ e = a^{-1}$	$(\forall E L_1)$
$L_2.$	$L_2$	$\vdash \forall w.(w^{-1} \circ w = e)$	$(Hyp)$
$L_7.$	$L_2$	$\vdash a^{-1} \circ a = e$	$(\forall E L_2)$
$L_3.$	$L_3$	$\vdash \forall x, y, z.((x \in S \wedge (y \in S \wedge y^{-1} \circ x = z)) \Rightarrow (z \in S))$	$(Hyp)$
$L_8.$	$L_3$	$\vdash (a \in S \wedge (a \in S \wedge a^{-1} \circ a = e)) \Rightarrow (e \in S)$	$(3 * \forall E L_3)$
$L_{11}.$	$L_{10}, L_2$	$\vdash a \in S \wedge a^{-1} \circ a = e$	$(\wedge I L_{10} L_7)$
$L_{12}.$	$L_{10}, L_2$	$\vdash a \in S \wedge (a \in S \wedge a^{-1} \circ a = e)$	$(\wedge I L_{10} L_{11})$
$L_{13}.$	$L_{10}, L_2, L_3$	$\vdash e \in S$	$(\Rightarrow E L_{12} L_8)$
$L_9.$	$L_3$	$\vdash (e \in S \wedge (a \in S \wedge a^{-1} \circ e = a^{-1})) \Rightarrow (a^{-1} \in S)$	$(3 * \forall E L_3)$
$L_{14}.$	$L_{10}, L_1$	$\vdash a \in S \wedge a^{-1} \circ e = a^{-1}$	$(\wedge I L_{10} L_6)$
$L_{15}.$	$L_{10}, L_1, L_2, L_3$	$\vdash e \in S \wedge (a \in S \wedge a^{-1} \circ e = a^{-1})$	$(\wedge I L_{14} L_{13})$
$L_{16}.$	$L_{10}, L_1, L_2, L_3$	$\vdash a^{-1} \in S$	$(\Rightarrow E L_{15} L_9)$
$L_5.$	$L_1, L_2, L_3$	$\vdash a \in S \Rightarrow a^{-1} \in S$	$(\Rightarrow I L_{10} L_{16})$
$L_4.$	$L_1, L_2, L_3$	$\vdash \forall v.(v \in S \Rightarrow v^{-1} \in S)$	$(\forall I L_5)$

■

### 3.5 Heuristic Control and Bad Results

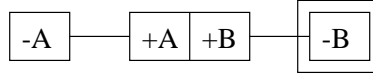
We finish this chapter with a discussion of some problems of the literal transformation algorithm that motivated us to develop another transformation algorithm. Our aim in trans-

forming refutation graphs is to obtain a uniform representation of the machine-found proofs in various machine-oriented formats that is better comprehensible for humans. Now that we have a transformation algorithm the question is whether the ND-proofs that we obtain from this algorithm satisfy our aim and are better comprehensible for humans. Unfortunately this is not the case since the ND-proofs resulting from the literal transformation algorithm are often very long and consist of many levels of indirect proofs. Although each single step in such a ND-proof is intuitive, the proof as a whole is not more comprehensible as the machine-found proofs. One reason for the bad quality of the ND-proofs is that we obtain from our transformation algorithm many indirect proof parts. In the following we explain why this happens in a transformation algorithm that decomposes until the literal level in both the refutation graph and the ND-lines is reached.

In some cases there are several transformation rules applicable to decompose one ND-line. For instance, to decompose a closed line with formula  $A \vee B$  we can use the rules  $TM \vee left$ ,  $TM \vee right$ , and  $TM Cases$  (see Appendix A). Assume we are in the following situation:

$L_1.$	$L_1$	$\vdash \neg A$	$(Hyp)$
$L_2.$	$L_2$	$\vdash A \vee B$	$(Hyp)$
$L_3.$	$L_1, L_2$	$\vdash B$	$(G)$

where  $G$  is:



What happens if we apply these three transformation rules on  $L_2$ , respectively?

If we apply  $TM \vee left$  on  $L_2$  we obtain:

$L_1.$	$L_1$	$\vdash \neg A$	$(Hyp)$
$L_4.$	$L_1$	$\vdash \neg A$	$(G')$
$L_2.$	$L_2$	$\vdash A \vee B$	$(Hyp)$
$L_5.$	$L_1, L_2$	$\vdash B$	$(\vee E_L L_2 L_4)$
$L_3.$	$L_1, L_2$	$\vdash B$	$(G'')$

Then we can close  $L_4$  and  $L_3$  by the corresponding literal base cases and obtain:

$L_1.$	$L_1$	$\vdash \neg A$	$(Hyp)$
$L_2.$	$L_2$	$\vdash A \vee B$	$(Hyp)$
$L_3.$	$L_1, L_2$	$\vdash B$	$(\vee E_L L_2 L_1)$

If we apply  $TM Cases$  and apply then the literal base cases we obtain:

$L_1.$	$L_1$	$\vdash \neg A$	$(Hyp)$
$L_2.$	$L_2$	$\vdash A \vee B$	$(Hyp)$
$L_4.$	$L_4$	$\vdash A$	$(Hyp)$
$L_5.$	$L_4, L_1$	$\vdash \perp$	$(Contradiction L_4 L_1)$
$L_6.$	$L_4, L_1$	$\vdash B$	$(Indirect L_5)$
$L_7.$	$L_7$	$\vdash B$	$(Hyp)$
$L_3.$	$L_1, L_2$	$\vdash B$	$(Cases L_2 L_6 L_7)$

Draft  
9/2/2009

If we would apply  $TM \vee right$  we would obtain the following situation:

$L_1.$	$L_1$	$\vdash \neg A$	$(Hyp)$
$L_4.$	$L_1, L_2$	$\vdash \neg B$	$(G')$
$L_2.$	$L_2$	$\vdash A \vee B$	$(Hyp)$
$L_5.$	$L_1, L_2$	$\vdash A$	$(\vee E_R L_2 L_4)$
$L_3.$	$L_1, L_2$	$\vdash B$	$(G'')$

This situation is not a transformation problem since  $L_4$  and  $L_3$  would not be justified correctly ( $L_4$  can not be proved by  $L_1$  and  $L_3$  can not be proved by  $L_5$ ). In similar example the application of  $TM \vee left$  would lead to a inconsistent situation. In Appendix A the transformation rules are divided into two sets, the *standard rules* and the *alternative rules*. The standard rules are complete and correct: They are complete in the sense that if we use only these rules the literal transformation algorithm is already complete (there is for each connector and quantifier an according transformation rule, see also the completeness results in [Lin90]). They are correct in the sense that the application of a standard rule produces never an inconsistent situation. In our example  $TM Cases$  is a standard rule and  $TM \vee left$  and  $TM \vee right$  are in alternative rules. Therefore, the applications of alternative rules are controlled by heuristics (see Appendix A). In our example the heuristic controlling  $TM \vee right$  forbids here the application of  $TM \vee right$ .

In our example we make the following observations:

- The application of  $TM \vee left$  produces a short, comprehensible, and direct proof.
- The application of  $TM Cases$  produces a much longer proof which contains indirect parts and is uncomprehensible.
- The application of  $TM \vee right$  would lead to an inconsistent situation which no transformation problem.

Although, their usage is complete and correct some standard rules suffer on the problem that they often produce bad and uncomprehensible proofs (like here  $TM Cases$  in comparison to  $TM \vee left$ ). Therefore, the alternative rules should be preferred if applicable. During the transformation of the standard example in the last section we used  $TM Inf$  which is an alternative rule to decompose line  $L_8$ . Finally, we obtained the following proof part inferring  $e \in S$  by using  $L_8$ :

$L_8.$	$L_3$	$\vdash (a \in S \wedge (a \in S \wedge a^{-1} \circ a = e)) \Rightarrow (e \in S)$	$(3 * \forall E L_3)$
$L_{11}.$	$L_{10}, L_2$	$\vdash a \in S \wedge a^{-1} \circ a = e$	$(\wedge I L_{10} L_7)$
$L_{12}.$	$L_{10}, L_2$	$\vdash a \in S \wedge (a \in S \wedge a^{-1} \circ a = e)$	$(\wedge I L_{10} L_{11})$
$L_{13}.$	$L_{10}, L_2, L_3$	$\vdash e \in S$	$(\Rightarrow E L_{12} L_8)$

If we apply instead of  $TM Inf$  the corresponding standard transformation rule  $TI \Rightarrow$  and continue with applying only standard rules we obtain the following proof part inferring  $e \in S$ :

$L_{10}.$	$L_{10}$	$\vdash a \in S$	( <i>Hyp</i> )
$L_7.$	$L_2$	$\vdash a^{-1} \circ a = e$	( $\forall E$ $L_2$ )
$L_8.$	$L_3$	$\vdash (a \in S \wedge (a \in S \wedge a^{-1} \circ a = e)) \Rightarrow (e \in S)$	( $3 * \forall E$ $L_3$ )
$L_9.$	$L_3$	$\vdash \neg(a \in S \wedge (a \in S \wedge a^{-1} \circ a = e)) \vee (e \in S)$	( <i>Taut</i> $L_8$ )
$L_{11}.$	$L_{11}$	$\vdash \neg(a \in S \wedge (a \in S \wedge a^{-1} \circ a = e))$	( <i>Hyp</i> )
$L_{13}.$	$L_{11}$	$\vdash \neg a \in S \vee \neg(a \in S \wedge a^{-1} \circ a = e)$	( <i>Taut</i> $L_{10}$ )
$L_{14}.$	$L_{14}$	$\vdash \neg a \in S$	( <i>Hyp</i> )
$L_{15}.$	$L_{10}, L_{14}$	$\vdash \perp$	( <i>Contradiction</i> $L_{10}$ $L_{14}$ )
$L_{16}.$	$L_{10}, L_{14}$	$\vdash a^{-1} \in S$	( <i>Indirect</i> $L_{15}$ )
$L_{17}.$	$L_{17}$	$\vdash \neg(a \in S \wedge a^{-1} \circ a = e)$	( <i>Hyp</i> )
$L_{18}.$	$L_{17}$	$\vdash \neg a \in S \vee \neg a^{-1} \circ a = e$	( <i>Taut</i> $L_{17}$ )
$L_{19}.$	$L_{19}$	$\vdash \neg a \in S$	( <i>Hyp</i> )
$L_{20}.$	$L_{10}, L_{19}$	$\vdash \perp$	( <i>Contradiction</i> $L_{10}$ $L_{19}$ )
$L_{21}.$	$L_{10}, L_{19}$	$\vdash a^{-1} \in S$	( <i>Indirect</i> $L_{20}$ )
$L_{22}.$	$L_{22}$	$\vdash \neg a^{-1} \circ a = e$	( <i>Hyp</i> )
$L_{23}.$	$L_7, L_{22}$	$\vdash \perp$	( <i>Contradiction</i> $L_7$ $L_{22}$ )
$L_{24}.$	$L_7, L_{22}$	$\vdash a^{-1} \in S$	( <i>Indirect</i> $L_{23}$ )
$L_{25}.$	$L_7, L_{10}, L_{17}$	$\vdash a^{-1} \in S$	( <i>Cases</i> $L_{17}$ $L_{24}$ $L_{21}$ )
$L_{26}.$	$L_7, L_{10}, L_{11}$	$\vdash a^{-1} \in S$	( <i>Cases</i> $L_{13}$ $L_{25}$ $L_{16}$ )
$L_{12}.$	$L_{12}$	$\vdash e \in S$	( <i>Hyp</i> )

In comparison with the ND-proof presented in the last section we produced by an application of *TMinf*, this proof part is very long and contains many indirect proof parts such that such a ND-proof is uncomprehensible for humans.

A transformation algorithm that decomposes until literal level is reached has to apply a lot of transformation rules since it decomposes until the smallest possible units are reached. Many applications of transformation rules provide many possibilities to make bad transformation rule choices. Thus, the quality of the resulting ND-proof depends then significantly on the quality of the heuristics controlling the choice of the transformation rules. If there are 'good' heuristics allowing in many cases the application of the alternative rules that avoid indirect parts then the resulting ND-proof will contain lesser indirect parts. But if the heuristics are of 'bad' quality such that we have to apply mainly the standard rules we will obtain 'ugly' ND-proofs with a lot of indirect parts. Most heuristics in previous works that aim to avoid indirect proof parts go rarely beyond vague guidelines [Lin90, PN90]. Although the heuristics described in Appendix A that control the application of the alternative rules are enriched versions of the heuristics in [Lin90] the problem remains on principle: when we decompose to literal level we have to apply many decomposition steps whose applications are hard to control such that ND-proofs of 'good' quality arise.

## Chapter 4

# The Assertion Level

We presented in the last chapter the literal transformation algorithm that transforms refutation graphs into ND-proofs. Furthermore, we described that the quality of the ND-proofs that we obtain by applications of this algorithm is not sufficient since the algorithm creates very long ND-proofs that contain many levels of indirect parts. This is partially caused by the algorithm itself. The literal transformation algorithm first decomposes the formulas in the ND-proof and the refutation graph until it reaches literal level and transfers then at the literal level steps from refutation graphs (linked contradictory literals) into steps in the ND-proof. Because of the decomposition to literal level the algorithm has to perform many decomposition steps that produce many ND-lines and many indirect parts (see Section 3.5). But another problem is the ND-calculus as target format itself. Our goal is to obtain by the transformation of the machine-found proofs a proof that is better comprehensible for humans. Hence, let us regard the standard example and how a mathematician would prove it:

*Let  $(G, \circ)$  be a group and  $S \subset G$ . If for  $S$  holds the so-called subgroup criterion: for all  $x, y \in S$  is also  $y^{-1} \circ x \in S$ , then holds also, that for every  $v \in S$  also  $v^{-1} \in S$ .*

**Proof:**

Let be  $a \in S$ .

Since  $a \in S$  we have applying the subgroup criterion that  $(a^{-1} \circ a) \in S$  and thus with  $a^{-1} \circ a = e$  we have  $e \in S$ .

Since  $e, a \in S$  we have applying the subgroup criterion that  $(a^{-1} \circ e) \in S$  and thus with  $a^{-1} \circ e = a^{-1}$  we have  $a^{-1} \in S$ .

Since  $a$  was chosen arbitrary we have that this conclusion holds in common, so we have  $\forall v(v \in S \Rightarrow v^{-1} \in S)$ . □

Although this would be already a very detailed proof for a presentation in a mathematical textbook, we find that, in comparison to this proof, our ND-proofs generated in the last chapter are very long and too tedious. Even the 'good' proof in Section 3.4 contains too many steps such that it is hardly comprehensible. A 'bad' proof (as partially given in Section 3.5) is completely uncomprehensible since it is even longer and contains a lot of indirect parts. The key steps in the 'human' proof, namely, the two applications of the subgroup criterion proving  $e \in S$  and  $a^{-1} \in S$  are hid somewhere in the mass of ND-steps. The problem is that the steps in the ND-proofs are at the level of pure syntactical manipulations of primitive logical connectives and quantifiers. Each single step is better comprehensible for humans as the typically cryptic steps in machine-oriented formats but the steps are not really meaningful for humans as, for instance, the operation 'application of subgroup criterion'.

If we examine mathematical proofs we find that most of the steps are at the level of theorem and definition applications (e.g., 'application of subgroup criterion'). Thus, in this chapter we describe an approach to reach this level also in ND-proofs. First, we examine in the next section how a theorem or definition application is expressed in the ND-calculus. We allow then steps in our ND-proofs which are justified by such theorem or definition applications. Theorems and definitions are called together *assertions*. In the second section we give algorithms which abstract ND-proofs to assertion level. The work of this two sections is taken from [Hua92, Hua94c]. Theoretically, a combination of our literal transformation algorithm with such an abstraction algorithm would provide us with higher-level ND-proofs at assertion level. Unfortunately, this combination suffers on some problems also discussed in the second section such that it is practically not possible. Thus, in the third section we introduce the basic idea for another transformation algorithm. We describe structures in refutation graphs that also represent assertion applications. The discovery of such structures provides us with a new possibility for the transformation: we can directly transform structures representing assertion steps in refutation graphs into corresponding assertion applications in the ND-proof. Note that the content of this section is based on similar results for resolution proof presented in [Hua96b].

## 4.1 Assertion Applications in ND-Proofs

Regard the following problem (we henceforth call it the *subset example*):

*Given is the subset definition:  $\forall S_1, S_2. (S_1 \subset S_2 \Leftrightarrow (\forall x. x \in S_1 \Rightarrow x \in S_2))$   
Let be  $U, F$  sets where  $U \subset F$  and  $a \in U$ . Show that  $a \in F$ .*

For a mathematician the proof of this sentence consists of a single step the application of the definition of subset:

*From  $U \subset F$  and  $a \in U$  follows that  $a \in F$  by the subset definition. (\*)*

A corresponding (direct) ND-proof is given in Figure 4.1<sup>1</sup>.

$$\begin{array}{c}
 \frac{\forall S_1, S_2 (S_1 \subset S_2 \Leftrightarrow (\forall x. x \in S_1 \Rightarrow x \in S_2))}{\forall S_2 (U \subset S_2 \Leftrightarrow (\forall x. x \in U \Rightarrow x \in S_2))} \forall E \\
 \frac{\frac{U \subset F \Leftrightarrow (\forall x. x \in U \Rightarrow x \in F)}{U \subset F \Rightarrow (\forall x. x \in U \Rightarrow x \in F)} \Rightarrow E}{\frac{\forall x. x \in U \Rightarrow x \in F}{a \in U \Rightarrow a \in F} \forall E} \Rightarrow E \\
 \frac{a \in U \Rightarrow a \in F}{a \in F} \Rightarrow E
 \end{array}$$

Figure 4.1: ND-proof of subset-example

Whereas the intuitive human proof consists of exactly one step at the level of definition application, this ND-proof consists of a sequence of 6 steps, each at the level of a pure syntactical manipulation of a logical connective or quantifier. One single step at this level is may comprehensible but a whole proof may consisting of hundreds of steps is not comprehensible.

<sup>1</sup>Whereas we prefer in general to present ND-proofs as sequence of lines, we choose in this section a representation as trees, since this presentation facilitates the understanding of this section.



If we want to obtain steps at the level of theorem or definition applications, the first question is how structures in ND-proofs look like whose conclusions a human would infer from their premises by the application of a theorem or a definition. A cognitive motivated approach is the *assertion level* suggested by Xiaorong Huang [Hua94c, Hua92]. Thereby theorems, definitions, lemmas, and so on are together called *assertions*.

An assertion application in ND-proofs is described by the so-called *compositions and decompositions constraints*. First, we need the definitions of a decomposition and a composition rule.

**Definition 4.1.1 (Decomposition Rule).**

An inference rule of the form  $\frac{\mathcal{A} \vdash A, \mathcal{A} \vdash P_1, \dots, \mathcal{A} \vdash P_n}{\mathcal{A} \vdash Q}$  is a *decomposition rule* with respect to formula schema  $A$ , if all applications of it, written as  $\frac{\mathcal{A} \vdash A', \mathcal{A} \vdash P'_1, \dots, \mathcal{A} \vdash P'_n}{\mathcal{A} \vdash Q'}$  satisfy the following condition:  $P'_1, \dots, P'_n$  and  $Q'$  are either

1. a proper subformula of  $A'$ , or
2. a specialization of  $A'$  or of one of its proper subformulas, or,
3. a negation of one of the first two cases.

■

**Definition 4.1.2 (Composition Rule).**

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

1. a proper subformula of  $Q'$ , or
2. a specialization of  $Q'$  or of one of its proper subformulas, or
3. a negation of one of the first two cases.

■

Decomposition rules are  $\frac{\mathcal{A} \vdash A \wedge B}{\mathcal{A} \vdash A} \wedge E_L$  and the analogous rule  $\wedge E_R$ ,  $\frac{\mathcal{A} \vdash A \Rightarrow B \quad \mathcal{A} \vdash A}{\mathcal{A} \vdash B} \Rightarrow E$ ,  $\frac{\mathcal{A} \vdash A \Rightarrow B \quad \mathcal{A} \vdash \neg B}{\mathcal{A} \vdash \neg A} MT$ ,  $\frac{\mathcal{A} \vdash \forall x. P[x]}{\mathcal{A} \vdash P[a]} \forall E$  and  $\frac{\mathcal{A} \vdash A \vee B \quad \mathcal{A} \vdash \neg B}{\mathcal{A} \vdash A} \vee E_L$  and the analogous rule  $\vee E_R$ . Composition rules are  $\frac{\mathcal{A} \vdash A \quad \mathcal{A} \vdash B}{\mathcal{A} \vdash A \wedge B} \wedge I$ ,  $\frac{\mathcal{A} \vdash A}{\mathcal{A} \vdash A \vee B} \vee I_L$  and the analogous rule  $\vee I_R$  and  $\frac{\mathcal{A} \vdash P[a]}{\mathcal{A} \vdash \exists x. P[x]} \exists I$ .

Table 4.1 gives a formal description of the compositions and decompositions constraints, an intuitive explanation we can give on two examples: The steps along the branch from the assertion  $\mathcal{A}$  to the root (of the assertion application tree) are all justified by decomposition rules. This branch is called the *main branch* of the assertion application. All other preconditions for steps along this main branch are created only with applications of composition rules. This branches ending at the main branch are called *side branches* of the assertion application. In the subset example in Figure 4.1 the main branch starts with the applied assertion, the subset definition, and leads via a sequence of decompositions steps to the root of the assertion application,  $U \subset F$ . As premises for these decomposition steps we need only  $U \subset F$  and  $a \in U$  so that we have not to apply some composition rules. Another example for an assertion application is in Figure 4.2.1.

### Compositions and Decompositions Constraints

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

**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 4.2, with the main branch is the branch from  $\mathcal{A}$  to  $A_n$ . The formula  $\mathcal{A}, A_1, \dots, A_n$  denote the main intermediate conclusions, and  $P_{i,1}, \dots, P_{i,m_i}$  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  $\mathcal{A}$ .

**decomposition property:** Main intermediate conclusion are justified by decomposition rules  $R_1, \dots, R_n$  with respect to the previous main intermediate conclusion. The inference along the main branch is therefore a linear process of decomposition of the assertion  $\mathcal{A}$ .

**composition property:** Other intermediate conclusions are justified by composition rules with respect to the corresponding intermediate conclusion.

Table 4.1: compositions and decompositions constraints


This figure contains also an assertion application. The main branch starts with the subgroup criterion as assertion and ends with the result  $e \in S$ . But here we need for one decomposition step at the main branch the premise  $a \in S \wedge (a \in S \wedge a^{-1} \circ a = e)$ . This premise is created on a side branch by some applications of a composition rule. The decompositions and compositions constraints guarantee that each formula on such a proof tree is a subpart of the applied assertion  $\mathcal{A}$ . Therefore, such a structure can be recognized by humans as one step, the application of the corresponding assertion  $\mathcal{A}$ .

Henceforth we allow the justification of ND-lines by assertion applications. We distinguish between two kinds of steps: first steps at assertion level, so-called *assertion level steps* and secondly steps in our basic ND-calculus as described in Section 2.4, so-called *basic steps*. We call the rules of this basic ND-calculus also the *basic ND-rules*. Furthermore, we call ND-proofs containing assertion level steps *assertion level ND-proofs* and ND-proofs without assertion level steps *basic ND-proofs*. For instance, the following is an assertion level proof of the subset example:

$$\frac{U \subset F \quad a \in U}{a \in F} \text{ subset definition}$$

Note that the compositions and decompositions constraints divide our basic ND-rules into three disjunct sets:

1. the decomposition rules, usable on the main branch of assertion applications



$$\begin{array}{c}
\frac{P_1}{\vdots} \\
\frac{\mathcal{A}, \frac{P_{0,1}}{\vdots}, \dots, P_{0,m_0}}{A_1} R_1 \\
\vdots \\
\frac{P_{i,1}, \dots, P_{i,m_i}}{A_i} R_i \\
\vdots \\
\frac{A_{n-1} R_{n-1} \quad P_{n-1,1}, \dots, \frac{P_m}{\vdots}}{A_n} R_n
\end{array}$$

Figure 4.2: quasi-linearity

$$\begin{array}{c}
\frac{\forall x, y, z. (x \in S \wedge (y \in S \wedge y^{-1} \circ x = z) \Rightarrow z \in S)}{\forall y, z. (a \in S \wedge (y \in S \wedge y^{-1} \circ a = z) \Rightarrow z \in S)} \forall E \\
\frac{\forall y, z. (a \in S \wedge (y \in S \wedge y^{-1} \circ a = z) \Rightarrow z \in S)}{\forall z. (a \in S \wedge (a \in S \wedge a^{-1} \circ a = z) \Rightarrow z \in S)} \forall E \\
\frac{\forall z. (a \in S \wedge (a \in S \wedge a^{-1} \circ a = z) \Rightarrow z \in S)}{a \in S \wedge (a \in S \wedge a^{-1} \circ a = e) \Rightarrow e \in S} \forall E \\
\frac{a \in S \quad \frac{a^{-1} \circ a = e}{a \in S \wedge a^{-1} \circ a = e} \wedge I}{a \in S \wedge (a \in S \wedge a^{-1} \circ a = e) \Rightarrow e \in S} \wedge I \\
\frac{a \in S \wedge (a \in S \wedge a^{-1} \circ a = e) \Rightarrow e \in S}{e \in S} \Rightarrow E
\end{array}$$

Figure 4.3: assertion application from the subset-example

2. the composition rules, usable on the side branches of assertion applications
3. rules, not usable at all in assertion applications (e.g., *Cases*, *Indirect*, *Contradiction*, *Choice*)

Tautology rules are neither decomposition nor composition rules but we need them in assertion applications (see [Hua94c]). To have unique decompositions and compositions sequences we cannot allow to use of these rules anywhere, since then we could produce sequences of the following kind:

$$\begin{array}{c}
\frac{F}{\neg \neg F} Taut \\
\frac{\neg \neg F}{F} Taut \\
\vdots
\end{array}$$

Therefore, we direct the tautology rules such that one direction is allowed only in decomposition sequences (on main branches) and the other direction is allowed only in composition sequences (on side branches). From [Hua94c] we take the following cognitive motivated approach: we divide the formulas which are connected by tautology rules into two categories: more natural forms and less natural forms. On the main branch applications of tautology rules are only allowed from the less natural form to the more natural, whereas on side branches only applications of tautology rules from the more natural to the less natural form are allowed.



	less natural	more natural
(1)	$\neg(F \vee G)$	$\neg F \wedge \neg G$
(2)	$\neg(F \wedge G)$	$\neg F \vee \neg G$
(3)	$\neg\neg F$	$F$
(4)	$\neg(F \Rightarrow G)$	$F \wedge \neg G$
(5)	$\neg\forall x.F$	$\exists x.\neg F$
(6)	$\neg\exists x.F$	$\forall x.\neg F$
(7)	$\neg(F \Leftrightarrow G)$	$\neg(F \Rightarrow G) \vee \neg(G \Rightarrow F)$
(8)	$F \Leftrightarrow G$	$(F \Rightarrow G) \wedge (G \Rightarrow F)$
(9)	$F \Rightarrow G$	$\neg F \vee G$
(10)	$\neg F \Rightarrow G$	$F \vee G$

In the next section we describe how a basic ND-proof as the proof for the subset example in Figure 4.1 can be abstracted to such an assertion level proof. But not that we can also compute always for an assertion level step a corresponding sequence of basic ND-steps proving the same conclusion with the same premises. This operation is called *expansion of assertion level steps*.

## 4.2 Abstraction to Assertion Level

We discuss in this section two algorithms suggested by Xiaorong Huang in [Hua94c] to abstract basic ND-proofs to assertion level ND-proofs. As examples we use the subset example from the last section and our standard example. Writing the proof of the subset example from Figure 4.1 in line notation we obtain the proof  $N_1$  in Figure 4.4.

$L_1.$	$L_1$	$\vdash U \subset F$	$(Hyp)$
$L_2.$	$L_2$	$\vdash a \in U$	$(Hyp)$
$L_3.$	$L_3$	$\vdash \forall S_1, S_2. S_1 \subset S_2 \Leftrightarrow (\forall x. x \in S_1 \Rightarrow x \in S_2)$	$(Hyp)$
$L_4.$	$L_3$	$\vdash U \subset F \Leftrightarrow (\forall x. x \in U \Rightarrow x \in F)$	$(2 * \forall E L_3)$
$L_5.$	$L_3$	$\vdash U \subset F \Rightarrow (\forall x. x \in U \Rightarrow x \in F)$	$(\Leftrightarrow E L_4)$
$L_6.$	$L_3, L_1$	$\vdash \forall x. x \in U \Rightarrow x \in F$	$(\Rightarrow E L_5)$
$L_7.$	$L_3, L_1$	$\vdash a \in U \Rightarrow a \in F$	$(\forall E L_6)$
$L_8.$	$L_3, L_1, L_2$	$\vdash a \in F$	$(\Rightarrow E L_7)$

Figure 4.4:  $N_1$

The first abstraction algorithm, the so-called *forward directed algorithm*, searches in the given basic ND-proofs for parts that satisfy the compositions and decompositions constraints. Such parts are then replaced by a single line, whose justification is the corresponding assertion application. If we apply this kind of abstraction on the subset example we obtain the proof in Figure 4.5 where the justification  $(L_3 L_1 L_2)$  means: Application of assertion  $L_3$  (the subset definition) on preconditions  $L_1$  and  $L_2$ .

Next, we apply this forward directed algorithm on the standard example.

*Example 4.2.1 (Standard Example (Continuation)).*

In the ND-proof from Example 3.4.3 we have two subparts fulfilling the compositions and decompositions constraints. The one tree is given in Figure 4.3 and produces  $e \in S$  as result



$L_1.$	$L_1$	$\vdash U \subset F$	$(Hyp)$
$L_2.$	$L_2$	$\vdash a \in U$	$(Hyp)$
$L_3.$	$L_3$	$\vdash \forall S_1, S_2. S_1 \subset S_2 \Leftrightarrow (\forall x. x \in S_1 \Rightarrow x \in S_2)$	$(Hyp)$
$L_8.$	$L_3, L_1, L_2$	$\vdash a \in F$	$(L_3 \ L_1 \ L_2)$

Figure 4.5: Abstraction of  $N_1$ 

of an application of the subgroup criterion; a similar tree infers the result  $a^{-1} \in S$  also by an application of the subgroup criterion. When we abstract these trees to assertion level we obtain the following abstracted ND-proof:

$L_{10}.$	$L_{10}$	$\vdash a \in S$	$(Hyp)$
$L_1.$	$L_1$	$\vdash \forall u. (u \circ e = u)$	$(Hyp)$
$L_6.$	$L_1$	$\vdash a^{-1} \circ e = a^{-1}$	$(\forall E \ L_1)$
$L_2.$	$L_2$	$\vdash \forall w. (w^{-1} \circ w = e)$	$(Hyp)$
$L_7.$	$L_2$	$\vdash a^{-1} \circ a = e$	$(\forall E \ L_2)$
$L_3.$	$L_3$	$\vdash \forall x, y, z. ((x \in S \wedge (y \in S \wedge y^{-1} \circ x = z)) \Rightarrow (z \in S))$	$(Hyp)$
$L_{13}.$	$L_{10}, L_2, L_3$	$\vdash e \in S$	$(L_3 \ L_{10} \ L_{10} \ L_7)$
$L_{16}.$	$L_{10}, L_1, L_2, L_3$	$\vdash a^{-1} \in S$	$(L_3 \ L_{13} \ L_{10} \ L_6)$
$L_5.$	$L_1, L_2, L_3$	$\vdash a \in S \Rightarrow a^{-1} \in S$	$(\Rightarrow I \ L_{10} \ L_{16})$
$L_4.$	$L_1, L_2, L_3$	$\vdash \forall v. (v \in S \Rightarrow v^{-1} \in S)$	$(\forall I \ L_5)$

■

How discussed in Section 3.5 it can happen that the ND-proof resulting from the transformation of a refutation graph is long and contains many indirect parts. A ND-proof for the subset example in this style is  $N_2$  in Figure 4.6.

In this case the forward directed algorithm is not able to abstract parts, since neither parts containing indirect steps (using *Contradiction* or *Indirect* as rules) nor parts containing case splittings (using *Cases*) satisfy the compositions and decompositions constraints. But the second abstraction algorithm, the so-called *backward directed algorithm* can be applied. This algorithm checks for every line  $L$  in the ND-proof whether  $L$  could also be inferred by the application of an assertion. Theoretically applicable assertions are thereby all ND-lines which depend from lesser (in the sense of subset) hypotheses as  $L$  itself. The algorithm starts with the conclusion of the ND-proof and works backwardly until the hypotheses of the proof are reached. In  $N_2$  we can infer  $L_{18}$  also by an assertion application from  $L_3$  on  $L_1$  and  $L_2$ . Thus, we can remove all the indirect parts and the case splits and abstract  $N_2$  to the same assertion level proof as  $N_1$ .

Theoretically, we could structure and improve the comprehensibility of the basic ND-proofs themselves by using this algorithm. A 'bad' ND-proof could be abstracted to assertion level and then the assertion level steps could be expanded again to corresponding sequences of basic ND-steps. But the proof parts resulting from the expansion of the assertion steps would satisfy the compositions and decomposition constraints. Thus, the result would be – equal of what quality the original basic ND-proof is – a basic ND-proof which is well structured and with less indirect parts. Unfortunately, the backward directed abstraction algorithm is more the search for new proofs than the abstraction of the given. Hence, its complexity is to high

$L_1.$	$L_1$	$\vdash U \subset F$	$(Hyp)$
$L_2.$	$L_2$	$\vdash a \in U$	$(Hyp)$
$L_3.$	$L_3$	$\vdash \forall S_1, S_2. S_1 \subset S_2 \Leftrightarrow (\forall x. x \in S_1 \Rightarrow x \in S_2)$	$(Hyp)$
$L_4.$	$L_3$	$\vdash U \subset F \Leftrightarrow (\forall x. x \in U \Rightarrow x \in F)$	$(2 * \forall E L_3)$
$L_5.$	$L_3$	$\vdash U \subset F \Rightarrow (\forall x. x \in U \Rightarrow x \in F)$	$(\Leftrightarrow E L_4)$
$L_6.$	$L_3$	$\vdash \neg(U \subset F) \vee (\forall x. x \in U \Rightarrow x \in F)$	$(Taut L_5)$
$L_7.$	$L_7$	$\vdash \neg(U \subset F)$	$(Hyp)$
$L_8.$	$L_7, L_1$	$\vdash \perp$	$(Contradiction L_7 L_1)$
$L_9.$	$L_7, L_1$	$\vdash a \in F$	$(Indirect L_8)$
$L_{10}.$	$L_{10}$	$\vdash \forall x. x \in U \Rightarrow x \in F$	$(Hyp)$
$L_{11}.$	$L_{10}$	$\vdash a \in U \Rightarrow a \in F$	$(\forall E L_{10})$
$L_{12}.$	$L_{10}$	$\vdash \neg(a \in U) \vee a \in F$	$(Taut L_{11})$
$L_{13}.$	$L_{13}$	$\vdash \neg(a \in U)$	$(Hyp)$
$L_{14}.$	$L_{13}, L_2$	$\vdash \perp$	$(Contradiction L_{13} L_2)$
$L_{15}.$	$L_{13}, L_2$	$\vdash a \in F$	$(Indirect L_{14})$
$L_{16}.$	$L_{16}$	$\vdash a \in F$	$(Hyp)$
$L_{17}.$	$L_2, L_{10}$	$\vdash a \in F$	$(Cases L_{12} L_{15} L_{16})$
$L_{18}.$	$L_1, L_2, L_3$	$\vdash a \in F$	$(Cases L_6 L_9 L_{17})$

Figure 4.6:  $N_2$ 

and it is not usable in practice [Hua94b]. As result, we have no practically usable algorithm to abstract 'bad' ND-proofs as  $N_2$ .

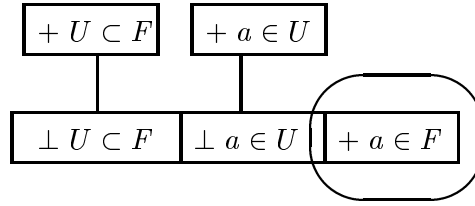
We call the combination of the two steps of first transforming a refutation graph into a basic ND-proof using the literal transformation algorithm from last chapter and then abstracting the ND-proof with the forward directed transformation algorithm the *two steps concept*. The problem of this two steps concept is that the final result mainly depends on the first step. If we are able to produce a 'good' ND-proof with less indirect parts and case splits we are able to find many abstractions and obtain as final result a compact proof with many assertion level steps. But if we create 'bad' ND-proofs with many indirect parts we are not able to further abstract them.

We discussed already in Section 3.5 why the literal transformation algorithm produces often proofs with many indirect parts such that we hardly can abstract these proofs. Another observation concerning the two steps concept is that we do double work: First, we decompose in the transformation step all structures in ND-lines and the refutation graphs until literal level is reached; then in the abstraction step we create again more abstract units. Hence, a better approach is obviously to identify directly in the refutation graph structures that represent assertion applications (in terms of refutation graphs). Such structures could be transformed into corresponding assertion level applications in the ND-proofs without further decomposition. This concept has two advantages: first we have not to make so many decomposition steps and secondly we directly obtain ND-proofs at assertion level without further abstraction. This idea is taken from Xiaorong Huang who examined in [Hua96b] such structures in resolution proofs. In the next section we transfer this idea to refutation graphs.

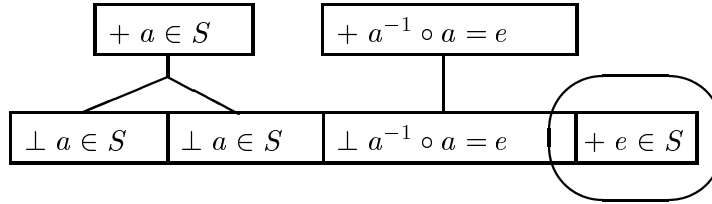
### 4.3 Assertion Applications in Terms of Refutation Graphs

We describe in this section structures in refutation graphs that represent assertion applications in term of refutation graphs. The identification of such structures allows another type of transformation algorithm: we can translate steps in refutation graphs that represent assertion applications directly into assertion steps in the ND-proof without decomposing the refutation graphs and the ND-lines to literal level as done by the literal transformation algorithm. We start with regarding some assertion applications already known from the last section and how they can be expressed in terms of refutation graphs.

First, regard the following graph of the subset example introduced in the first section of this chapter:



At the heart in this graph is a non unit clause which is in the CNF of the assertion subset definition. This non unit clause is linked with a set of unit clauses. This structure is similar to the structure of the corresponding ND-proof in Figure 4.1. There the subset definition as starting point of the main branch is at the heart, too. All other branches end at this main branch. The result of the assertion application is  $a \in F$  which also the unlinked literal of our graph. Next, regard the following graph corresponding to another assertion application taken from the standard example (see Example 4.2.1 in the last section):



Here we find again the characteristic structure of a non unit clause which is in the CNF of the assertion (here the subgroup criterion) and a set of unit clauses linked with the non unit clause. The conclusion of the assertion application is the unlinked literal (here  $+(e \in S)$ ). We call this characteristic structure *unit clause step (UCS)*.

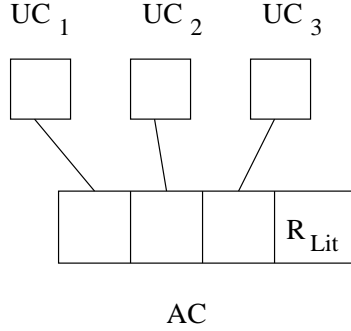
#### Definition 4.3.1 (Unit Clause Step (UCS)).

Let  $G$  be a refutation graph. A triple  $(AC, \{UC_1, \dots, UC_n\}, R_{lit})$  is called *unit clause step (UCS)* in  $G$  if:

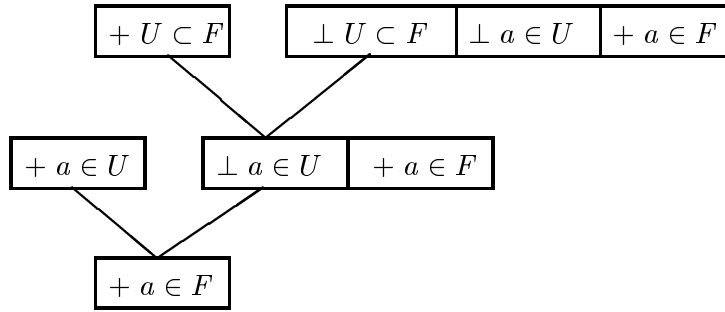
- $AC, UC_1, \dots, UC_n$  are clauses of  $G$  and  $R_{lit}$  is a literal of  $AC$ ,
- $UC_1, \dots, UC_n$  are unit clauses and  $AC$  is not an unit clause,
- each literal of  $AC$  except  $R_{lit}$  is linked with the literal of one of the unit clauses of  $UC_1, \dots, UC_n$  and vice versa.

We call the clauses  $UC_1, \dots, UC_n$  the *unit clauses of the UCS*, the clause  $AC$  the *assertion clause of the UCS*, and the literal  $R_{lit}$  the *result literal* or simply *the result of the UCS*. ■

Note again that an UCS has the following characteristic form:

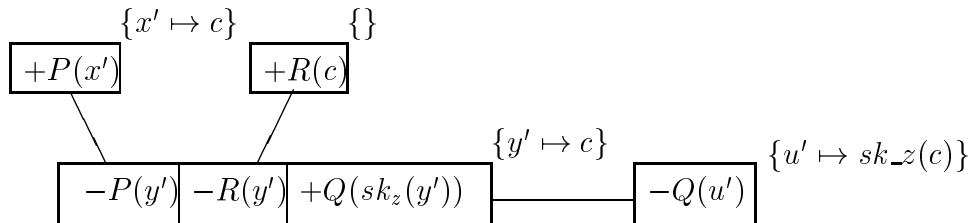


Xiaorong Huang examined in [Hua96b] similar structures in resolution proofs. Thereby, an UCS corresponds to a sequence of unit resolution steps decreasing little-by-little a non unit clause (the assertion clause), for instance, for the subset example we have the unit resolution steps:



Even if there are similarities between an UCS and a ND-tree representing an assertion application (as described above in this section), the UCS itself is only a graph structure in a refutation graph. The question is when represents an UCS which assertion application. To develop an answer for this question we give now an example which contains no UCS that represents an assertion application whereas examples for UCSs representing assertion applications we already gave at the beginning of this section.

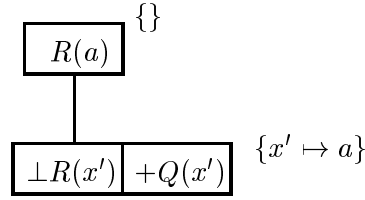
*Example 4.3.2 (Non-Assertion UCS).*



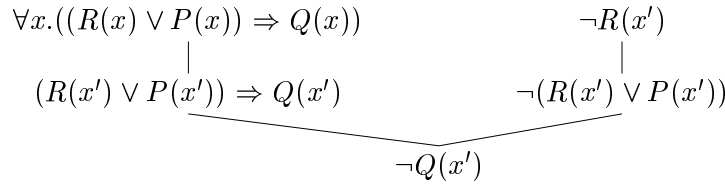
The clauses of this graph could be connected by a delta-relation with the lines of the following ND-proof:







The result of this UCS is the literal  $Q(x')$ . If we follow the branch from the root of the derivation tree to the result literal, we see that this branch satisfies the decompositions constraints similar to the main branch of an assertion application. Therefore, we call this branch the *main branch of the derivation tree*. The other branches of the tree are called *side branches of the derivation tree*. If we 'invert' the side branch of our derivation tree we obtain the following tree:



By 'inversion' the decompositions on the side branch become compositions such that this 'inverted' side branch satisfies the compositions constraints. Indeed such derivation trees with 'inverted' side branches fulfill both the compositions and decompositions constraints.

The complete proof of the UCS-theorem in Appendix C consist mainly of an algorithm that uses this idea and constructs from the derivation tree of the assertion clause of an UCS a ND-tree which fulfills the compositions and decompositions constraints, whose main branch starts with the assertion formula and ends with the result literal of the UCS, and whose side branches start with the literals of the unit premises clauses of the UCS. This algorithm stepwise translates steps from the derivation tree into corresponding steps in the ND-proof. Some of these correspondations are obvious, for instance:

$$\begin{array}{c} G \wedge H \\ \downarrow \\ G \end{array} \quad (\alpha\text{-rule}) \quad \text{and} \quad \frac{G \wedge H}{G} \wedge E_L$$

In other cases the relationships are not such obvious since the ND-rules are not based on the  $\alpha, \beta, \gamma, \delta$  concept but directly on the logical connectives and quantifiers. If we use the algorithm on the UCS and the assertion formula of our example we obtain the following ND-tree:

$$\frac{\frac{\forall x.((R(x) \vee P(x)) \Rightarrow Q(x))}{(R(a) \vee P(a)) \Rightarrow Q(a)} \forall E \quad \frac{R(a)}{R(a) \vee P(a)} \vee I_L}{Q(a)} \Rightarrow E$$

Not that if the unit premise clauses  $UC_1, \dots, UC_n$  would not be ground we would have to instantiate them (how ever) on the side branches. But since this is not possible (there are no ND-rules instantiating free variables in formulas) we need condition  $C_1$  demanding that  $UC_1, \dots, UC_n$  are ground. We need condition  $C_2$  since the application of the  $\delta$ -rule in a clause normalization tree would correspond to an application of the *Choice* rule during

the transformation of the derivation tree into a ND-tree. But the *Choice* rule is neither a composition nor a decomposition rule and therefore it is not possible to use it in a tree that should fulfill the compositions and decompositions constraints. The graph in Example 4.3.2 contains no UCSs corresponding to assertion application since both the condition  $C_1$  and condition  $C_2$  are violated.

In Section 3.1 we established a transformation based on a correspondence between refutation graphs and ND-proofs at literal level (the literal base cases). In this section we presented a relationship on a higher level, namely, the assertion level. In both the refutation graphs and ND-proofs we described structures representing assertion applications. In Chapter 6 we shall present a transformation algorithm using this relationship by directly transforming UCSs corresponding to assertion level steps into assertion level steps in the ND-proof. This spares to decompose first until literal level is reached and to abstract then to assertion level. But before we can give this new transformation algorithm we have first to examine in the next chapter the question how we can split graphs into a sequence of UCSs and how refutation graphs look like that can be split into a sequence of UCSs.



## Chapter 5

# UCS-Decomposable Graphs

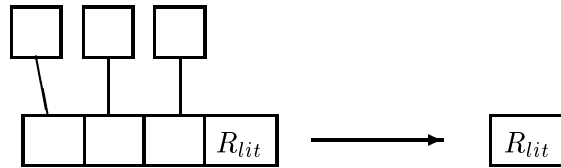
In the last chapter we introduced the assertion level which allows to use in ND-steps macro-steps applying theorems, lemmas, or definitions. Furthermore, we showed in the UCS-theorem that there are also structures in refutation graphs representing assertion applications. The idea of a new transformation algorithm is straight forward: we want to translate a UCS which represents an assertion application into a corresponding assertion application in the ND-proof. A precondition to translate UCSs is to extract it from a refutation graphs. Thereby, the best case is a refutation graph consisting only of a sequence of UCSs. Such graphs are called *UCS-decomposable*.

A UCS itself is only a structure in a graph; it is another question whether a UCS represents a particular assertion application. Similarly, the property of UCS-decomposability is a pure structural property of a refutation graphs, stating not an extracted UCS represents an assertion application.

We start this chapter with a formal definition of *UCS-decomposability* of refutation graphs. Then we give a complete algorithm which splits a UCS-decomposable refutation graph into a sequence of UCSs. The rest of the chapter describes which classes of refutation graphs are UCS-decomposable. The content of this chapter is of very technical nature since it contains many technical proofs about properties of classes of particular graphs. For the reader not interested in the details of these proofs we recommend to read only the first section, introducing the concept of UCS-decomposability of refutation graphs, and the last section, summarizing the results of this chapter.

### 5.1 UCS-Decompositions and UCS-Decomposition Algorithm

How already described in the introduction of this chapter we want to use UCSs or at least UCSs representing assertion applications to translate them directly into an assertion step in the ND-proof. Informally spoken, from such a translated UCS remains only its result literal in the graph which forms a new unit clause:



Then this new unit clause can be used to find new UCSs and so on. We now define this process formally.

**Definition 5.1.1 (UCS-Replacement).**

Let  $G$  be a refutation graph and  $(AC, \{UC_1, \dots, UC_n\}, R_{lit})$  a UCS in  $G$  and  $UC_{new}$  a new unit clause consisting of the literal  $R_{lit}$ . Under the *UCS-replacement* of this UCS in  $G$  we understand the procedure of creating the modified refutation graph  $G'$  from  $G$  by the following steps:

1. Remove  $AC$  from  $G$ .
2. Add  $UC_{new}$  to  $G$ .
3. Let  $\Lambda$  be the link connecting  $R_{lit}$  of  $AC$ . If  $\Lambda$  was removed by step 1, add a new link  $\Lambda'$  connecting the literal  $R_{lit}$  of  $UC_{new}$  and the literals previously connected by  $\Lambda$  (except the one in  $AC$  that is removed by step 1). If  $\Lambda$  was not removed, add  $R_{lit}$  of  $UC_{new}$  to  $\Lambda$ .
4. Each clause  $UC_1, \dots, UC_n$  whose unique literal has become pure by step 1 is removed, too.

■

Note that the removing of the clauses  $AC$  and  $UC_1, \dots, UC_n$  can also cause the removing of some links (see Section 2.3). We show now that the resulting graph  $G'$  is again a minimal refutation graph if  $G$  was a minimal refutation graph. To do so, we need the following theorem taken from [Eis88] (theorem 5.14)

**Theorem 5.1.2 (Deduction Graph Theorem).**

*A deduction graph is minimal if and only if it has one more clauses as links.*

**Lemma 5.1.3 (Correctness of the UCS-Replacement).**

*Let  $G$  be a minimal refutation graph. An UCS-replacement in  $G$  generates again a minimal refutation graph  $G'$ .*

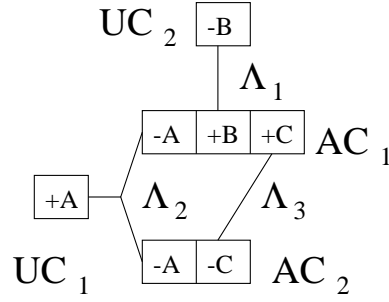
**Proof:**

For each trail in  $G'$  is also a similar trail in  $G$ . Therefore,  $G'$  has to be acyclic since  $G$  is acyclic (if  $G'$  would be cyclic,  $G$  would also be cyclic; this would be a contradiction to the fact that  $G$  is a refutation graph). That  $G'$  has no pure literals follows from the fact that all literals which could maybe pure are explicitly connected (by step 3) or removed from the graph (by step 4). Thus, we have that  $G'$  is a refutation graph.

There is still a link connecting  $R_{lit}$ . Depending on step 3 this is still the original link or it is a new 'copy' of it. Other links which may be removed are the links between the other literals of  $AC$  (except  $R_{lit}$ ) and the unit clauses  $UC_1, \dots, UC_n$ . But then step 4 guarantees that exactly as many links are removed as clauses.

$G$  is minimal and so (by the deduction graph Theorem 5.1.2)  $G$  contains one more clause as links. Since the removed clause  $AC$  is replaced by a new clause  $UC_{new}$  and else as many clauses as links are removed we have after the UCS-replacement still exactly one more clause as links. Therefore,  $G'$  is also a minimal refutation graph by Theorem 5.1.2. □

Example 5.1.4.

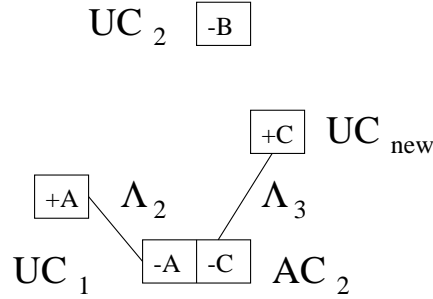


This refutation graph contains two UCSs:

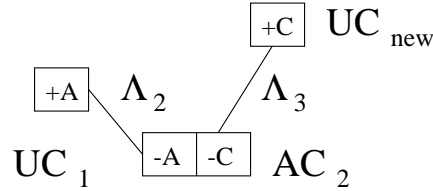
$\mathcal{U}_1 = (AC_1, \{UC_1, UC_2\}, +C)$  and  $\mathcal{U}_2 = (AC_2, \{UC_1\}, \perp C)$

We make now a UCS-replacement with  $\mathcal{U}_1$ .

First  $AC_1$  is removed, then  $UC_{new} = [+C]$  is added and a new link is established using  $UC_{new}$  (steps 1, 2, and 3 of the USC-replacement).



$\Lambda_1$  is also removed since its positive shore  $S^+(\Lambda_1)$  became empty. Thus, the literal of  $UC_2$  becomes pure too. So by step 4 we have to remove  $UC_2$  too. We obtain as result of the UCS-replacement the following minimal graph:



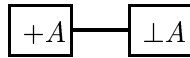
■

We define now inductively UCS-decomposable graphs and the according UCS-decompositions.

**Definition 5.1.5 (UCS-Decomposable Refutation Graphs).**

Let  $G$  be a refutation graph.  $G$  is called *UCS-decomposable* with a *UCS-decomposition*  $D(G) = \{\mathcal{U}_1, \dots, \mathcal{U}_n, E\}$ , with UCSs  $\mathcal{U}_1, \dots, \mathcal{U}_n$  ( $n \geq 0$ ) and end step  $E$ , if

1.  $G$  has the form





In this case a UCS-decomposition of  $G$  consists only of the end step  $([+A], [\perp A])$  such that  $D(G) = \{([+A], [\perp A])\}$ . Thereby, graphs like  $G$  consisting only of two contradictory and linked unit clauses are called an *end step*.

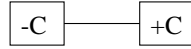
2. There is a UCS  $\mathcal{U}_1$  in  $G$  and the graph  $G'$  that is the result of the UCS-replacement of  $\mathcal{U}_1$  in  $G$  is UCS-decomposable with an UCS-decomposition  $D(G') = \{\mathcal{U}_2, \dots, \mathcal{U}_n, E\}$ . Then a UCS-decomposition of  $G$  is  $D(G) = \{\mathcal{U}_1, \mathcal{U}_2, \dots, \mathcal{U}_n, E\}$ .

■

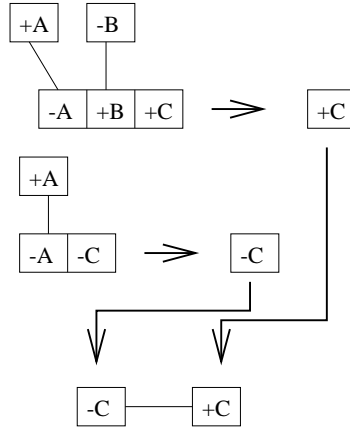
We continue Example 5.1.4 by giving a complete UCS-decomposition of the refutation graph.

*Example 5.1.6.*

In the final refutation graph of Example 5.1.4 are again two UCSs:  $\mathcal{U}_3 = (AC_2, \{UC_1\}, \perp C)$  and  $\mathcal{U}_4 = (AC_2, \{UC_{new}\}, \perp A)$ . If we continue the UCS-decomposition with  $\mathcal{U}_3$  we obtain as resulting refutation graph:

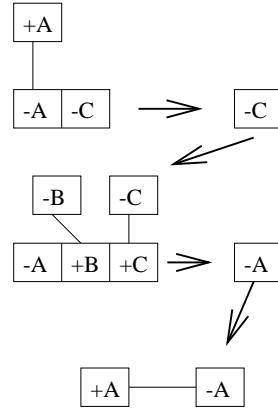


This graph is an end step and is therefore UCS-decomposable by definition. Thus, we obtain for the original graph we started with in Example 5.1.4 that it is also UCS-decomposable with a UCS-decomposition  $\{\mathcal{U}_1, \mathcal{U}_3, ([\perp C], [+C])\}$ .



Thereby the arrows mark the introduction of the new unit clauses and how they are used again. Note that this decomposition is not the only possible one. There are also other possible UCS-decompositions of the original graph, for instance, the following one:



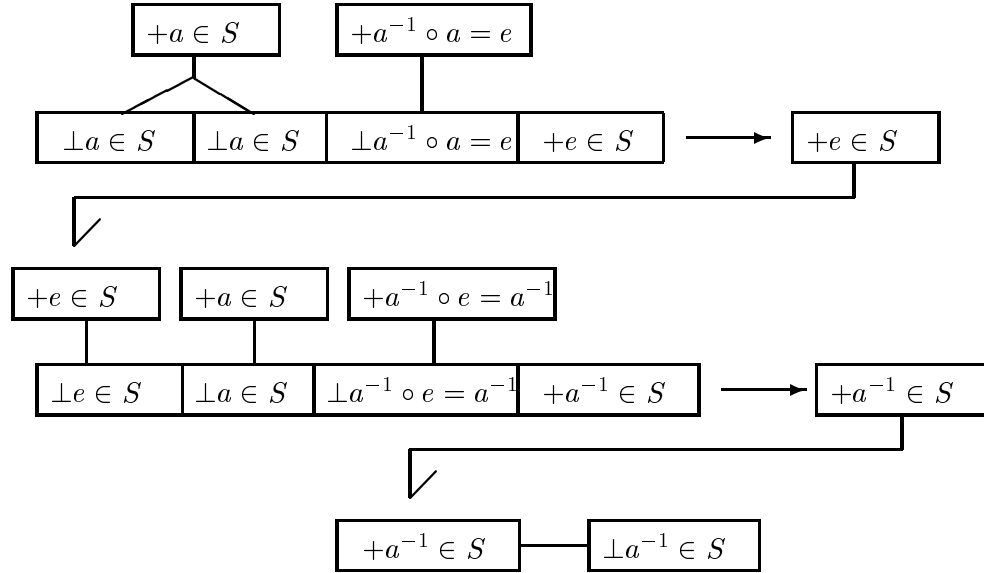


■

As next we present a UCS-decomposition of our standard example:

*Example 5.1.7 (Standard Example (Continuation)).*

(compare with the refutation graph in Figure 2.1)

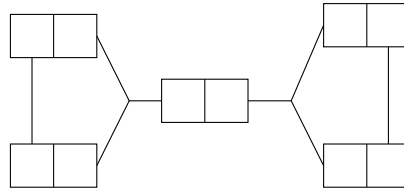


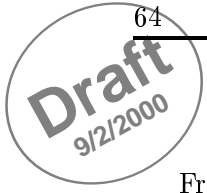
Note that there are other possible UCS-decompositions, too.

■

In contrast to the last examples that are UCS-decomposable, the following is an example of a refutation graph that is not UCS-decomposable:

*Example 5.1.8.*





From the inductive definition of UCS-decomposability we directly obtain the following recursive algorithm to compute a UCS-decomposition. ■

**Algorithm 5.1.9 (UCS-Decomposition Algorithm).**

Let  $G$  be a refutation graph.

**Initialization-step:** Assign  $D(G) = \{\}$  and  $G_{curr} = G$ .

**UCS-replacement-step:** As long as  $G_{curr}$  has not the form of an end step:

- Seek an UCS  $\mathcal{U}$  in  $G_{curr}$ .
- Replace  $\mathcal{U}$  in  $G_{curr}$  and assign  $G_{curr}$  to the resulting refutation graph.
- Assign  $D(G) = D(G) \cup \{\mathcal{U}\}$

If there is no UCS  $\mathcal{U}$  in  $G_{curr}$  stop with the message: Cannot terminate properly!

**Final-step:** If  $G_{curr}$  has the form of an end step  $S$ , assign  $D(G) = D(G) \cup S$  and finish. ■

The correctness of this algorithm is obvious:

**Theorem 5.1.10 (Correctness of the UCS-Decomposition Algorithm).**

*Let  $G$  be a non empty, minimal refutation graph and let  $D(G) = \{\mathcal{U}_1, \dots, \mathcal{U}_n, S\}$  be the result of the UCS-decomposition algorithm on  $G$ . Then  $G$  is UCS-decomposable and  $D(G)$  is a UCS-decomposition of  $G$ .*

**Proof:**

Direct consequence of the fact that the algorithm is the recursive correspondence to the inductive definition of UCS-decomposability and a UCS-decomposition. Formally, a proof would use induction over the number  $n$  of non-unit clauses in  $G$ . If  $n = 0$   $G$  (as non empty minimal refutation graph) has to be an end step, if  $n \geq 0$  there has to be a UCS in  $G$  that we can replace. □

Whereas the correctness of this algorithm is obvious, the completeness of this algorithm is not such clear. Does this algorithm compute for every UCS-decomposable graph a corresponding UCS-decomposition? The problem is that we demand in the inductive definition of UCS-decomposable graphs that such a graph  $G$  contains (only) *one*  $\mathcal{U}$  such that the graph  $G'$  resulting from  $G$  by the UCS-replacement is again UCS-decomposable. We demand in this definition not that for *each* UCS in  $G$  the graph  $G'$  resulting by a UCS-replacement is again UCS-decomposable. In contrast thereto, we take in the recursive UCS-decomposition algorithm an arbitrary UCS in  $G$  to continue the decomposition. Can it happen that  $G$  is UCS-decomposable but since we take the 'wrong' UCS we obtain after the UCS-replacement a graph  $G'$  which is not UCS-decomposable?

In the next sections we shall answer this question. We shall see that it is equal which UCS is chosen as next in the UCS-decomposition algorithm, since each UCS-replacement in a UCS-decomposable graph produces again a UCS-decomposable graph. We show this by describing two classes of refutation graphs which are UCS-decomposable, namely the *binary-linked* and

the *contra-links free* graphs. Since we can show that the class of *contra-links free* graphs is the maximal class of UCS-decomposable graphs at all, we shall prove the completeness of the UCS-decomposition algorithm by proving that it is complete on this class of graphs.

## 5.2 Binary-Linked Refutation Graphs

Obviously, a graphs that should be UCS-decomposable has to contain some unit clauses to start a UCS-decomposition. If we look at Example 5.1.8 containing a not UCS-decomposable graph, we see that this graph has no unit clauses. In general, we can state nothing about the number of unit clauses in refutation graphs, but in a special class of refutation graphs we can do so.

### Definition 5.2.1 (Binary-Links, Multi-Links, Binary-Linked Refutation Graphs).

Let  $G$  be a refutation graph. A link  $\Lambda$  in  $G$  is called:

**binary-link** if  $\Lambda$  connects exactly two contradictory literals ( $|S^+(\Lambda)| = |S^-(\Lambda)| = 1$ ).

**simple multi-link** if either  $|S^+(\Lambda)| > 1$  or  $|S^-(\Lambda)| > 1$ .

**double multi-link** if both  $|S^+(\Lambda)| > 1$  and  $|S^-(\Lambda)| > 1$ .

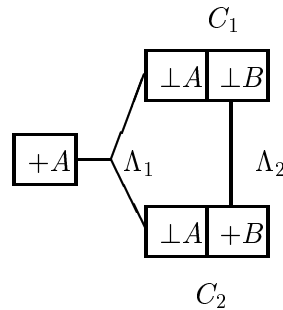
A refutation or deduction graph containing only binary-links is called a *binary-linked graph*. ■

We show now that in a minimal binary-linked graph holds a certain relationship between the number of unit clauses and the number of non unit clauses. But we need first the following results from [Eis88]:

### Definition 5.2.2 (Order on Links).

Let  $\Lambda_1$  and  $\Lambda_2$  be two links in a refutation graph  $G$ .  $\Lambda_1$  is called *lesser* than  $\Lambda_2$ ,  $\Lambda_1 <_L \Lambda_2$ , if there are clauses  $C_1$  and  $C_2$  in  $G$  containing literals which are in the same shore of  $\Lambda_1$  and which are connected by a trail using  $\Lambda_2$ .  $<_L^*$  is the reflexive and transitive closure of  $<_L$ . ■

*Example 5.2.3.*



$\Lambda_1 <_L \Lambda_2$  since  $C_1$  and  $C_2$  have both a literal in the same shore of  $\Lambda_1$  (here  $S^-(\Lambda_1)$ ) and  $C_1$  and  $C_2$  are connected by the trail  $(C_1, \Lambda_2, C_2)$  containing  $\Lambda_2$ . ■

From [Eis88] we take also the following lemmas (the lemmas 5.6, 5.7, 5.11 in [Eis88]).

**Lemma 5.2.4.**

1. In a deduction graph  $<_L^*$  is a partial order on the links.
2. If the link  $\Lambda$  in a deduction graph  $G$  is minimal with respect to  $<_L^*$ , then  $\Lambda$  is separating in  $G$ .
3. Let  $\Lambda$  be minimal with respect to  $<_L^*$  in a minimal deduction graph  $G$ . Furthermore, let  $G \perp \Lambda$  be the graph resulting by removing  $\Lambda$  from  $G$ . Then holds:  
 $G \perp \Lambda$  consists of two separated (see 2) minimal deduction graphs, such that each deduction graph contains one shore of  $\Lambda$ .

**Corollary 5.2.5.**

In a binary-linked refutation graph each link is separating.

**Proof:** In a binary-linked refutation graph we have no multi-links. Hence, there are no two different links  $\Lambda_1$  and  $\Lambda_2$  with  $\Lambda_1 <_L^* \Lambda_2$ . Therefore, all links are minimal and by Lemma 5.2.4 separating.  $\square$

With these results we can now prove the following lemma:

**Lemma 5.2.6.**

Let  $G$  be a non empty, binary-linked, and minimal deduction graph then holds

$$2 * (N_C \perp 1) + U_C + L_P = \sum NU_{Lit}.$$

where

$N_C$  is the number of non unit clauses in  $G$ ,

$U_C$  is the number of unit clauses in  $G$ ,

$L_P$  is the number of pure literals in  $G$ , and

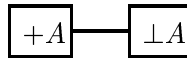
$\sum NU_{Lit}$  is the sum of the number of all literals of all non unit clauses in  $G$ .

**Proof:** Induction over the number  $n$  of non unit clauses in  $G$ .

**Induction Base Case:  $n = 0$** 

There are two possibilities:

1.  $G$  is an end step:



Then holds:  $N_C = 0, U_C = 2, L_P = 0, \sum NU_{Lit} = 0$  which satisfies  $2 * (N_C \perp 1) + U_C + L_P = \sum NU_{Lit}$ .

2.  $G$  consists of one single unit clause.

Then holds:  $N_C = 0, U_C = 1, L_P = 1, \sum NU_{Lit} = 0$  which satisfies  $2 * (N_C \perp 1) + U_C + L_P = \sum NU_{Lit}$ .

Further cases are not possible, since (by the proof assumptions)  $G$  is non empty and minimal.

**Induction Step:**  $n \mapsto n + 1$

$G$  contains  $n + 1 \geq 1$  non unit clauses. Let  $C$  be an arbitrary non unit clause and  $v \geq 0$  the number of links on  $C$ . Since  $G$  is binary-linked, by Corrolary 5.2.5 each link in  $G$  is separating. If we remove step-by-step each of these links on  $C$  we obtain by Lemma 5.2.4  $v + 1$  minimal deduction graphs  $G_1, \dots, G_{v+1}$  (for each removed link we obtain one more separated graph). One of these graphs, assume it is  $G_1$ , consists only of one clause, namely  $C$ , and no links (since we explicitly removed all links on  $C$ ).

On  $G_1$  holds:  $N_{C_1} = 1, U_{C_1} = 0, L_{P_1} = Lit_C, \sum NU_{Lit_1} = Lit_U$ , with  $Lit_C$  is the number of literals of  $C$ . Thus  $G_1$  satisfies:  $2 * (N_{C_1} \perp 1) + U_{C_1} + L_{P_1} = \sum NU_{Lit_1}$ .

Since all other graphs  $G_2, \dots, G_{v+1}$  can contain at most  $n$  non unit clauses, for them holds by induction hypothesis:  $2 * (N_{C_i} \perp 1) + U_{C_i} + L_{P_i} = \sum NU_{Lit_i}$  for  $2 \leq i \leq v + 1$ , respectively.

Adding these equation for  $G_1, \dots, G_{v+1}$  we obtain:

$$\begin{aligned} \sum_{i=1}^{v+1} (2 * (N_{C_i} \perp 1) + U_{C_i} + L_{P_i}) &= \sum_{i=1}^{v+1} (\sum NU_{Lit_i}) \\ \Rightarrow 2 * ((\sum_{i=1}^{v+1} N_{C_i}) \perp (v + 1)) + (\sum_{i=1}^{v+1} U_{C_i}) + (\sum_{i=1}^{v+1} L_{P_i}) &= \sum_{i=1}^{v+1} (\sum NU_{Lit_i}) (*) \end{aligned}$$

Since altogether no literals or clauses are really removed, for  $G$  holds:

$$\begin{aligned} N_C(G) &= \sum_{i=1}^{v+1} N_{C_i} \\ U_C(G) &= \sum_{i=1}^{v+1} U_{C_i} \\ \sum NU_{Lit}(G) &= \sum_{i=1}^{v+1} \sum (NU_{Lit_i}) \end{aligned}$$

Since  $G$  is binary-linked each removing of a link causes that exactly two more literals become pure. Thus, we have:

$$L_P(W) = (\sum_{i=1}^{v+1} L_{P_i}) \perp 2 * v$$

which is equivalent to

$$\sum_{i=1}^{v+1} L_{P_i} = L_P(W) + 2 * v$$

Using this in (\*) we obtain

$$\begin{aligned} \sum NU_{Lit}(G) &= 2 * (N_U(G) \perp (v + 1)) + U_C(G) + L_P(G) + 2 * v \\ &= 2 * (N_C(G) \perp 1) \perp 2 * v + U_C(G) + L_P(G) + 2 * v \\ &= 2 * (N_C(G) \perp 1) + U_C(G) + L_P(G) \end{aligned}$$

which proves the induction step.

□

From this lemma for minimal deduction graphs we obtain directly the following corollary for minimal refutation graphs:

**Corollary 5.2.7.**

Let  $G$  be a non empty, binary-linked, and minimal refutation graph. Let  $N_C, U_C$ , and  $\sum NU_{Lit}$  be defined as in Lemma 5.2.6.

Then holds:  $2 * (N_C \perp 1) + U_C = \sum NU_{Lit}$ .

**Proof:** Follows from Lemma 5.2.6 since a refutation graph has no pure literals ( $L_P = 0$ ).  $\square$

With this corollary we can prove the following theorem about the UCS-decomposability of binary-linked graphs.

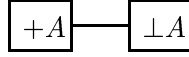
**Theorem 5.2.8 (UCS-Decomposition Theorem for Binary-Linked Ref. Graphs).**

Let  $G$  be a non empty, minimal, and binary-linked refutation graph. Then  $G$  is UCS-decomposable with a UCS-decomposition with  $n$  UCSs where  $n$  is the number of non unit clauses of  $G$ .

**Proof:** Induction over the number  $n$  of the non unit clauses of  $G$ .

**Induction Base Case:**  $n = 0$ 

Since  $G$  is non empty and minimal it has to be an end-step:



Thus, by definition of UCS-decomposability  $G$  is UCS-decomposable with a UCS-decomposition  $D(G) = \{([+A], [\perp A])\}$  consisting of 0 UCSs.

**Induction Step:**  $n \mapsto n + 1$ 

$G$  contains  $n + 1$  non unit clauses. As first we prove by contradiction that there is an UCS in  $G$ .

Assume there is no UCS in  $G$ .

Then each non unit clause is linked with at least two other non unit clauses.

Hence, the number of unit clauses ( $U_C$ ) in  $G$  is at most the sum of the literals of the non unit clauses ( $\sum NU_{Lit}$ ) minus  $2 * (n + 1)$ , since at least two literals of each non unit clause are linked with literals of other non unit clauses.

Thus, we have:  $U_C \leq \sum NU_{Lit} \perp 2 * (n + 1) (*)$

But by Corollary 5.2.7 we have that  $2 * ((n + 1) \perp 1) + U_C = \sum NU_{Lit}$  which is equivalent to  $U_C = \sum NU_{Lit} \perp 2 * n$

This is a contradiction to  $(*)$  and it follows that there has to be a UCS  $\mathcal{U}_1$  in  $G$ .

By replacing this UCS in  $G$  we obtain a graph  $G'$  which has  $n$  non unit clauses and which is still non empty, minimal, and binary-linked (see Lemma 5.1.3; the UCS-replacement creates no multi-links).

Thus, by induction hypothesis  $G'$  is UCS-decomposable with a UCS-decomposition with  $n$  UCSs. Let  $D(G') = \{\mathcal{U}_2, \dots, \mathcal{U}_{n+1}, E\}$  be a UCS-decomposition of  $G'$ , then by definition of UCS-decomposability  $D(G) = \{\mathcal{U}_1, \mathcal{U}_2, \dots, \mathcal{U}_{n+1}, E\}$  is a UCS-decomposition of  $G$  with  $n + 1$  UCSs.

$\square$

As a corollary of the UCS-decomposition theorem for binary-linked graphs we obtain the completeness of the UCS-decomposition Algorithm 5.1.9 on binary-linked refutation graphs.

**Corollary 5.2.9.**

*The UCS-decomposition Algorithm 5.1.9 is complete on the class of minimal, non empty, and binary-linked refutation graphs. If applied on a minimal, non empty, and binary-linked refutation graph  $G$  it computes a UCS-decomposition with  $n$  UCSs, where  $n$  is the number of the unit clauses of  $G$ .*

**Proof:** Follows directly from the proof of the UCS-decomposition theorem for binary-linked refutation graphs:

Since a binary-linked refutation graph is UCS-decomposable by Theorem 5.2.8 we can find a UCS in it.

By replacing this UCS we obtain a refutation graph in the class of binary-linked graphs (since the UCS-replacement creates no multi-links).

Thus, this new graph is again UCS-decomposable by Theorem 5.2.8 and we can proceed with the UCS-decomposition algorithm.  $\square$

### 5.3 Further Results on Binary-Linked Refutation Graphs

In the last section we showed that binary-linked refutation graphs are UCS-decomposable and that the UCS-decomposition Algorithm 5.1.9 is complete on the class of minimal, non empty, and binary-linked refutation graphs. Indeed, we can show even a stronger result on binary-linked refutation graphs and their UCS-decomposability.

Assume we want to obtain as end-step of a UCS-decomposition a particular pair of contradictory linked unit clauses. We shall show in this section that this is possible for binary-linked graphs. We start with introducing a slightly changed version of the UCS-decomposition Algorithm 5.1.9 which we call the *blocked UCS-decomposition algorithm*.

**Algorithm 5.3.1 (Blocked UCS-Decomposition Algorithm).**

Let  $G$  be a refutation graph and  $L_1$  and  $L_2$  two contradictory literals in  $G$  connected by a link.

**Initialization-step:** Assign  $D(G) = \{\}$  and  $G_{curr} = G$ .

**UCS-replacement-step:** As long as  $G_{curr}$  has not the form of an end step:

- Seek an UCS  $\mathcal{U}$  in  $G_{curr}$ , such that in the unit clauses of  $\mathcal{U}$  neither  $L_1$  nor  $L_2$  are contained.
- Replace  $\mathcal{U}$  in  $G_{curr}$  and assign  $G_{curr}$  to the resulting refutation graph.
- Assign  $D(G) = D(G) \cup \{\mathcal{U}\}$

If there is no UCS  $\mathcal{U}$  in  $G_{curr}$  stop with the message: Cannot terminate properly!

**Final-step:** If  $G_{curr}$  has the form of an end step  $S$ , assign  $D(G) = D(G) \cup S$ .

■

In this blocked UCS-decomposition algorithm the usage of two linked literals  $L_1$  and  $L_2$  in the unit clauses of an UCS is forbidden. The correctness of the algorithm on non empty and minimal refutation graphs follows directly from the correctness of the original

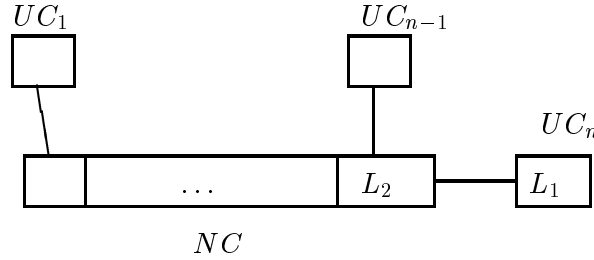
the UCS-decomposition algorithm proved in Theorem 5.1.10. Unclear is the completeness of this algorithm since we explicitly forbid some UCS-replacements. However, for binary-linked refutation graphs we can show the following important lemma we shall need later on.

**Lemma 5.3.2.**

*Let  $G$  be a non empty, minimal, and binary-linked refutation graph and let  $L_1$  and  $L_2$  be two contradictory literals in  $G$  connected by a link  $\Lambda$ . When called on  $G$  with blocked literals  $L_1$  and  $L_2$  the blocked UCS-Decomposition algorithm computes a UCS-decomposition  $D(G) = \{\mathcal{U}_1, \dots, \mathcal{U}_n, E\}$  where  $n$  is the number of non unit clauses in  $G$  and  $E = ([L_1], [L_2])$ .*

**Proof:** Technically, the proof is similar to the proof of the UCS-decomposition theorem for binary-linked refutation graphs: Induction over the number  $n$  of non unit clauses of  $G$ . Therefore, we give here no formal proof but present only the key steps.

1. The case that  $G$  has no non unit clauses is trivial, since then  $[L_1]$  and  $[L_2]$  are the unique clauses of  $G$  and hence  $G$  is directly the needed end step  $E = ([L_1], [L_2])$ .
2. If the number of non unit clauses in  $G$  is exactly one, then  $G$  is:



In this case either  $L_1$  or  $L_2$  is contained in one of the unit clauses  $UC_1, \dots, UC_n$ . We assume now without loss of generality that  $UC_n$  contains  $L_1$ . The other literal must be contained in the unique non unit clause  $NC$  (here  $L_2$ ).

$G$  contains – among other possible UCSs – the UCS  $\mathcal{U} = (NC, \{UC_1, \dots, UC_{n-1}\}, L_1)$ . If we replace  $\mathcal{U}$  we obtain a refutation graph with 0 non unit clauses but consisting of two linked unit clauses with the literals  $L_1$  and  $L_2$ . All other UCSs in  $G$  are blocked since they would contain  $UC_n$  as unit clause. Thus, the blocked UCS-decomposition creates for  $G$  the UCS-decomposition  $D(G) = \{\mathcal{U}, ([L_1], [L_2])\}$ .

3. If the number  $n$  of non unit clauses in  $G$  is bigger than one we can show that there are at least two UCSs in  $G$  such that at least one of them uses neither  $L_1$  nor  $L_2$  as unit clauses. The argumentation is similar to the argumentation in the proof of the UCS-decomposition theorem for binary-linked refutation graphs showing that there exists at least one UCS.

Assume  $G$  contains  $n + 1 \geq 2$  non unit clauses and there is only one UCS in  $G$  (that one has to exist we already showed in the proof of the UCS-decomposition theorem for binary-linked refutation graphs), then holds:

$U_C \leq \sum NU_{Lit} \pm 2 * n \pm 1$  (where  $U_C$  and  $\sum NU_{Lit}$  are defined as usual, see Lemma 5.2.6). This equation holds since  $n$  non unit clauses have at least two links to other non unit clauses (therefore  $\pm 2 * n$ ), one non unit clause has exactly one link to another non unit





clause, namely the non unit clause in the UCS (therefore  $\perp 1$ ).

This is a contradiction to  $U_C = \sum NU_{Lit} \perp 2 * ((n+1) \perp 1)$  known for  $G$  by Corollary 5.2.7.

Hence, our assumption was wrong and there are at least two UCSs  $\mathcal{U}_1$  and  $\mathcal{U}_2$  in  $G$ .

Now assume that both  $\mathcal{U}_1$  and  $\mathcal{U}_2$  contain either  $L_1$  or  $L_2$  in their unit clauses (that both UCSs contain the same literal in their unit clauses is not possible, since we have binary-linked graphs).

Then we have that both  $L_1$  and  $L_2$  are in unit clauses.

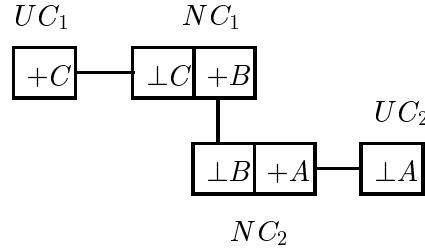
But this is impossible since we demanded that  $L_1$  and  $L_2$  are linked and by a UCS-replacement the link structure is not changed. But a graph containing only  $L_1$  and  $L_2$  in linked unit clauses would already be a refutation graph. Hence,  $G$  that contains at least 2 non unit clauses would not be minimal. This is a contradiction to the precondition that  $G$  is minimal.

Thus, we have that either  $\mathcal{U}_1$  or  $\mathcal{U}_2$  contains neither  $L_1$  nor  $L_2$  in its unit clauses. This UCS can then be chosen by the blocked UCS-Decomposition algorithm to proceed.

□

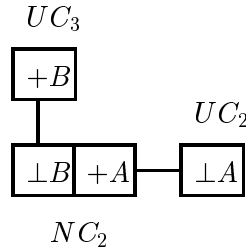
*Example 5.3.3.*

Given is the following non empty, minimal, and binary-linked refutation graph  $G$ .

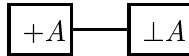


We block the literals  $+A$  and  $\perp A$  in the clauses  $NU_2$  and  $UC_2$  since we want to obtain an end step with two connected unit clauses containing exactly these literals, respectively. Then we apply the blocked UCS-Decomposition algorithm.

We have two UCSs in  $G$ :  $(NC_1, \{UC_1\}, +B)$  and  $(NC_2, \{UC_2\}, \perp B)$ . But the replacement of the second one is forbidden in the blocked UCS-Decomposition algorithm. Thus, we replace the first one and obtain:



Now we have again two UCSs:  $(NC_2, \{UC_3\}, +A)$  and  $(NC_2, \{UC_2\}, \perp B)$ . Again the replacement of the second one is forbidden in the blocked UCS-Decomposition algorithm. Thus, we replace again the first one and obtain the wished end step:



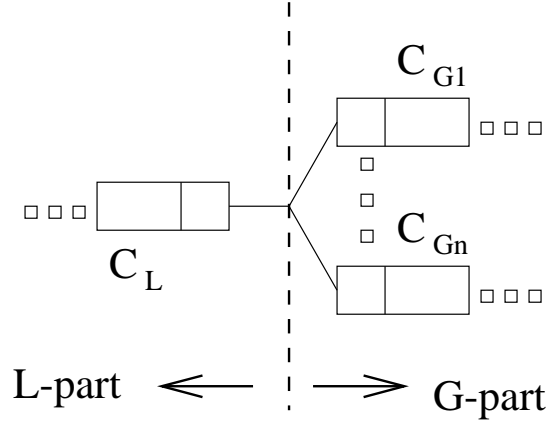
■

**Draft**  
9/2/2015

## 5.4 Contra-Links Free Graphs

In the last two sections we described important properties of binary-linked refutation graphs. In this section we present properties of refutation graphs with multi-links. As first we focus on graphs with simple multi-links. At the end of the section we shall transfer our results also to graphs with double multi-links.

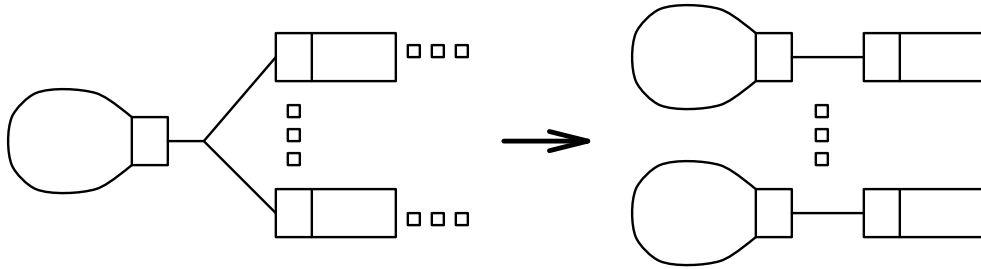
A single multi-link  $\Lambda$  looks as follows:



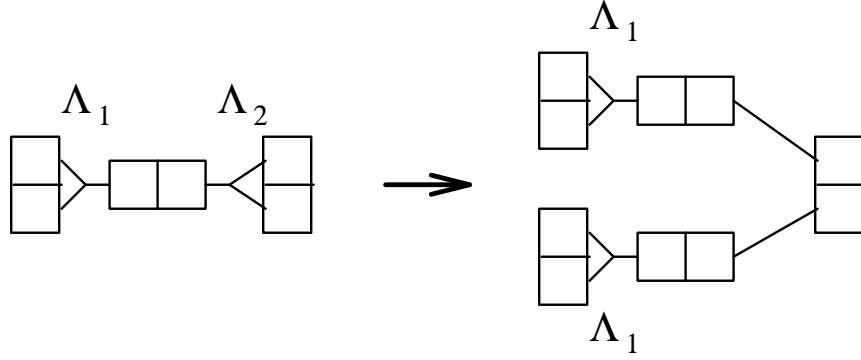
### Definition 5.4.1 (L-Clause and G-Clauses at a Simple Multi-Link).

Let  $G$  be a refutation graph and  $\Lambda$  a simple multi-link in  $G$ . Then the unique clause with a literal in the shore of  $\Lambda$  with cardinality 1 is called *L-clause* of  $\Lambda$ . The clauses with literals in the other shore of  $\Lambda$  are called the *G-clauses* of  $\Lambda$ . ■

The basic idea to handle graphs with multi-links is to reduce step-by-step the number of multi-links to obtain less-by-less a binary-linked graph, which we know how to handle. An intuitive idea to do so is to copy the L-part of a simple multi-link and to replace then the simple multi-link by a set of binary-links.

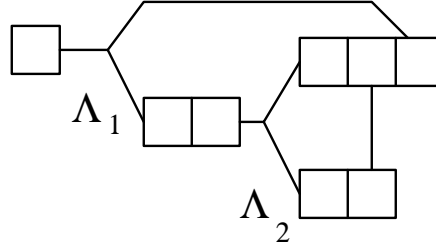


This naive approach suffers on the following problems: First, if the L-part is a large subgraph then this method would cause that the resulting graphs become larger and larger. Secondly, even if we would locally delete a multi-link we could not guarantee in general that we enhance the global situation. For instance, if the copied L-part contains again simple multi-links then these multi-links are also copied. Thus, in the resulting graph we would may have more multi-links as in the original one.

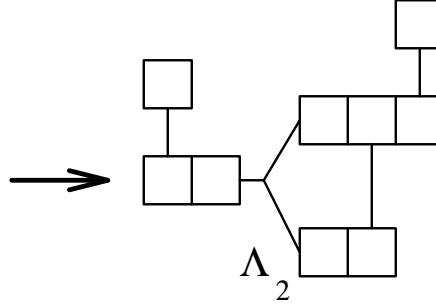


Hence, the question raises whether and in which graphs there is an order in which the simple multi-links can be deleted such that we obtain thereby continuously graphs with less-and-less multi-links. Look at the following example:

*Example 5.4.2.*



We should delete first  $\Lambda_1$  by copying its L-parts since the deletion of  $\Lambda_1$  decreases the number of multi-links in the graph, whereas we gain nothing when deleting first  $\Lambda_2$ .



Deleting afterwards  $\Lambda_2$  by copying its L-part results in a binary-linked refutation graph (but one that is more complex if we take the number of clauses and links as measure of complexity). ■

As in the last example we should delete first such simple multi-links that in their L-part is no other multi-link. This leads us to the following relation  $\leq_g$  on simple multi-links.

**Definition 5.4.3 (The  $\leq_g$  Relation).**

Let  $G$  be a refutation graph and  $\Lambda_1$  and  $\Lambda_2$  simple multi-links in  $G$ .

Then holds  $\Lambda_1 \leq_g \Lambda_2$  if and only if  $\Lambda_1 = \Lambda_2$  or there is a trail from the L-clause of  $\Lambda_2$  to one

Draft  
9/2/2006

of the clauses on  $\Lambda_1$  without using  $\Lambda_2$ . ■

In this definition, it is equal whether the trail reaches the L-clause or one of the G-clauses of  $\Lambda_1$  since by using  $\Lambda_1$  or (without using  $\Lambda_1$ ) we obtain a trail to the other one(s), respectively. We see now that the notation  $L$  and  $G$  (in L-part, G-part, L-clause, G-clauses) is reasonable since the L-part (G-part) is the *lesser* (*greater*) part with respect to  $\leq_g$ : in the L-part of a link  $\Lambda$  are the simple multi-links lesser than  $\Lambda$ , whereas in the G-part of  $\Lambda$  are the simple multi-links greater than  $\Lambda$ .

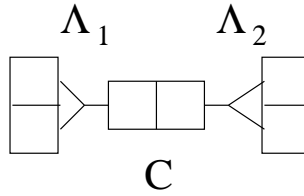
To give an intuition for the  $\leq_g$  ordering we present now some examples of relations between two with respect to  $\leq_g$ .

*Example 5.4.4.*

1.  $\Lambda_1 \leq_g \Lambda_2$  and  $\Lambda_2 \not\leq_g \Lambda_1$ .

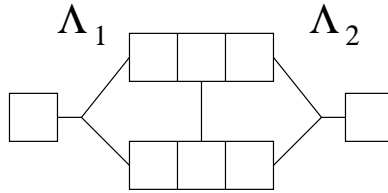
In the original graph of our last Example 5.4.2  $\Lambda_1 \leq_g \Lambda_2$  holds since the L-clause of  $\Lambda_2$  is directly one of the G-clauses of  $\Lambda_1$ . On the other hand, every trail from the L-clause of  $\Lambda_1$  to another clause has to use  $\Lambda_1$ . Therefore,  $\Lambda_2 \not\leq_g \Lambda_1$ .

2.  $\Lambda_1 \leq_g \Lambda_2$  and  $\Lambda_2 \leq_g \Lambda_1$



$\Lambda_1 \leq_g \Lambda_2$  and  $\Lambda_2 \leq_g \Lambda_1$  holds since  $C$  is both the L-clause of  $\Lambda_1$  and  $\Lambda_2$ .

3.  $\Lambda_1 \not\leq_g \Lambda_2$  and  $\Lambda_2 \not\leq_g \Lambda_1$



$\Lambda_1 \not\leq_g \Lambda_2$  and  $\Lambda_2 \not\leq_g \Lambda_1$  holds since there are no trails from one of the L-clauses to any other clauses without using the links  $\Lambda_1$  or  $\Lambda_2$ , respectively. ■

In these examples we can see that case 2 is obviously not UCS-decomposable, since it contains no unit clauses at all. Graphs with two links  $\Lambda_1$  and  $\Lambda_2$  where  $\Lambda_1 \leq_g \Lambda_2$  and  $\Lambda_2 \leq_g \Lambda_1$  are a very important class of graphs as we shall see later on. Therefore, we define them now formally.

**Definition 5.4.5 (Contra-Links, Contra-Links Free Graphs).**

Let  $G$  be a refutation graph and  $\Lambda_1$  and  $\Lambda_2$  simple multi-links in  $G$ . If  $\Lambda_1 \leq_g \Lambda_2$  and  $\Lambda_2 \leq \Lambda_1$  and  $\Lambda_1 \neq \Lambda_2$  holds then  $\Lambda_1$  and  $\Lambda_2$  are called *contra-directed links* or simply *contra-links*.

Does neither  $\Lambda_1 \leq_g \Lambda_2$  nor  $\Lambda_2 \leq_g \Lambda_1$  hold then  $\Lambda_1$  and  $\Lambda_2$  are called *undirected links*.

Does  $G$  contain no contra-directed links then  $G$  is called *contra-links free graph*. ■

In the rest of this section we shall show that contra-links free refutation graphs are UCS-decomposable and that our UCS-decomposition Algorithm 5.1.9 is complete on this class of refutation graphs. But first we need some lemmas.

**Lemma 5.4.6.**

Let  $G$  be a refutation graph and  $\Lambda_1$  and  $\Lambda_2$  simple multi-links in  $G$ , such that  $\Lambda_1 \leq_g \Lambda_2$  holds. Then there is a trail from  $\Lambda_1$  and  $\Lambda_2$  without using  $\Lambda_1$  to  $\Lambda_2$ .

**Proof:** By definition of  $\leq_g$ , it holds that there is a trail  $T_{21}$  from the L-clause of  $\Lambda_2$  to  $\Lambda_1$  without using  $\Lambda_2$ .

Assume this trail uses  $\Lambda_1$ .

Then let  $T'_{21}$  be the subset of  $T_{21}$  without  $\Lambda_1$  and all elements after  $\Lambda_1$ .

Then  $T'_{21}$  is a trail from  $\Lambda_1$  to  $\Lambda_2$  without using  $\Lambda_1$  and  $\Lambda_2$ . □

**Lemma 5.4.7 ( $\leq_g$  is a Partial Order).**

Let  $G$  be a contra-links free graph. Then the relation  $\leq_g$  is partial order on its simple multi-links.

**Proof:**

**Reflexivity:** holds by definition of  $\leq_g$ .

**Anti-Symmetric:** holds by the assumption that  $G$  is contra-links free.

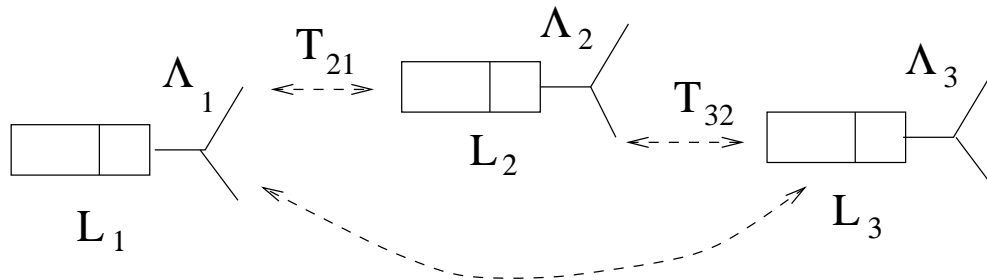
**Transitivity:** We have to show that from  $\Lambda_1 \leq_g \Lambda_2$  and  $\Lambda_2 \leq_g \Lambda_3$  follows that  $\Lambda_1 \leq_g \Lambda_3$ ;

When expanding the definition of  $\leq_g$  we have to show that

there is a trail  $T_{31}$  from the L-clause  $L_3$  of  $\Lambda_3$  to  $\Lambda_1$  without using  $\Lambda_3$  under the assumptions, that

1. there is a trail  $T_{21}$  from the L-clause  $L_2$  of  $\Lambda_2$  to  $\Lambda_1$  without using  $\Lambda_2$ , and
2. there is a trail  $T_{32}$  from the L-clause  $L_3$  of  $\Lambda_3$  to  $\Lambda_2$  without using  $\Lambda_3$ .

The idea is to connect these two trails  $T_{32}$  and  $T_{21}$  to the trail  $T_{31}$ , such that  $T_{31}$  does not contain  $\Lambda_3$ .



$T_{32}$  does not contain  $\Lambda_3$  but what is with  $T_{21}$ ? We show now that  $T_{21}$  does also not contain  $\Lambda_3$ .

Assume  $T_{21}$  contains  $\Lambda_3$ .

Then there would exist a trail  $T_{23}$  which is a subset of  $T_{21}$  starting with  $L_2$  and ending at  $\Lambda_3$ . Furthermore,  $T_{23}$  would not contain  $\Lambda_2$ , since  $T_{21}$  does not contain  $\Lambda_2$ .

Therefore, it would hold that  $\Lambda_3 \leq_g \Lambda_2$  because of this trail  $T_{23}$ .

This is a contradiction to the hypothesis that  $G$  is contra-links free since  $\Lambda_2 \leq_g \Lambda_3$  already holds.

Thus, we know that  $T_{21}$  does not contain  $\Lambda_3$  (\*).

Then there are the following two cases:

**Case 1:**  $T_{32}$  and  $T_{21}$  have no links in common.

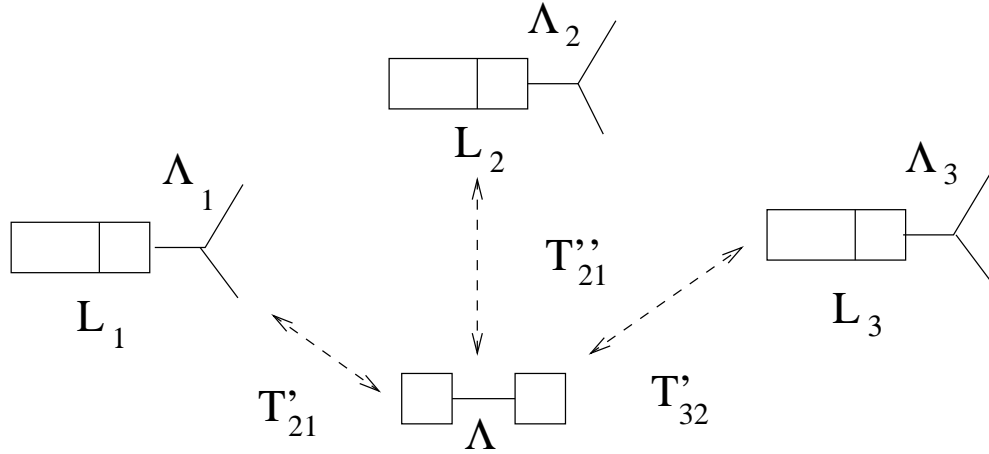
In this case we can create  $T_{31}$  simply by connecting  $T_{32}$  and  $T_{21}$ : Start with clause  $L_3$  use  $T_{32}$  to reach  $\Lambda_2$  and  $L_2$  (if  $T_{32}$  ends with a G-clause of  $\Lambda_2$  use  $\Lambda_2$  to reach  $L_2$ ). Then use  $T_{21}$  (starting with  $L_2$ ) to reach  $\Lambda_1$ . This trail does not use  $\Lambda_3$  since neither  $T_{21}$  nor  $T_{32}$  contains  $\Lambda_3$ .

Thus, with this  $T_{31}$  we have that  $\Lambda_1 \leq_g \Lambda_3$ .

**Case 2:**  $T_{32}$  and  $T_{21}$  share some common links.

Let be  $\Lambda$  the first link in  $T_{32}$  which is in both  $T_{32}$  and  $T_{21}$ . Let be  $T'_{32}$  the subset of  $T_{32}$  connecting  $L_3$  and  $\Lambda$ ,  $T'_{21}$  the subset of  $T_{21}$  connecting  $\Lambda_1$  and  $\Lambda$ , and  $T''_{21}$  the subset of  $T_{21}$  connecting  $L_2$  and  $\Lambda$ . Thus,  $\Lambda$  separates  $T_{21}$  into the disjunct trails  $T'_{21}$  and  $T''_{21}$ . It holds that  $T'_{21}$ ,  $T''_{21}$ , and  $T'_{32}$  share no links (\*\*) (since  $\Lambda$  is explicitly the *first* link of  $T_{32}$  also in  $T_{21}$ ).

Furthermore, neither  $T'_{32}$ , nor  $T'_{21}$ , nor  $T''_{21}$  contain  $\Lambda$  (\* \* \*).



We want to create the trail  $T_{31}$  as a connection of  $T'_{21}$  and  $T'_{32}$  via the link  $\Lambda$ . But what happens if  $T'_{21}$  and  $T'_{32}$  end with clauses in the same shore of  $\Lambda$ ? We have first to prove that this cannot happen.

Assume  $T'_{21}$  and  $T'_{32}$  would end with clauses in the same shore of  $\Lambda$ .

Then it would hold that  $T''_{21}$  ends at the other shore of  $\Lambda$ .

Then we could create a trail  $T_{23}$  starting at  $L_2$  using then  $T''_{21}$  to reach  $\Lambda$ , and using then  $\Lambda$  and  $T'_{32}$  to reach  $L_3$  and  $\Lambda_3$  ( $T_{23}$  would be a trail because of (\*\*) and (\* \* \*)).  $T''_{21}$  does not contain  $\Lambda_2$  by definition (subset of  $T_{21}$ ). Since  $T_{32}$  ends



by definition on  $\Lambda_2$  we can assume without loss of generality that  $T'_{32}$  (as subset of  $T_{32}$ ) does also not contain  $\Lambda_2$  (because of Lemma 5.4.6 we can assume that  $T_{32}$  does not contain  $\Lambda_2$ ). Thus,  $T_{23}$  does not contain  $\Lambda_2$ .

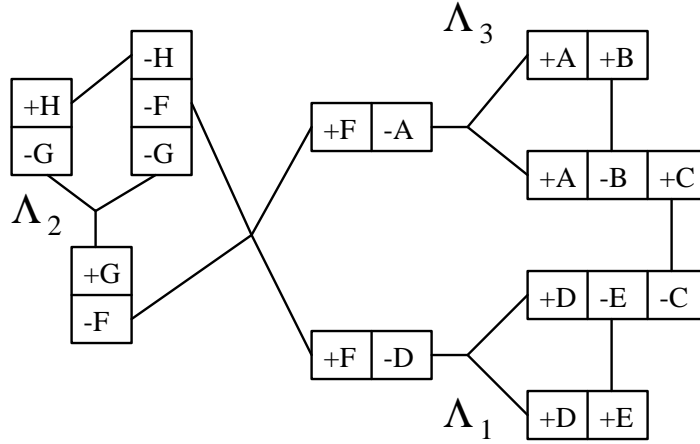
Therefore,  $\Lambda_3 \leq_g \Lambda_2$  would hold and this would be a contradiction to the hypothesis that  $G$  is contra-links free since  $\Lambda_2 \leq_g \Lambda_3$  holds already.

Hence, we know that  $T'_{21}$  and  $T'_{32}$  end with clauses in different shores of  $\Lambda$ . Then we can create  $T_{31}$  by connecting  $T'_{21}$  and  $T'_{32}$  via  $\Lambda$ : Start with clause  $L_3$  then use  $T'_{32}$  to reach  $\Lambda$ ; use then  $\Lambda$  and  $T'_{21}$  to reach  $\Lambda_1$ .  $T_{31}$  is a trail because of  $(**)$  and  $(***)$ . It does not contain  $\Lambda_3$  because of  $(*)$ .

Therefore, we have that  $\Lambda_1 \leq_g \Lambda_3$ .

□

Note that the relation  $\leq_g$  is a partial order only in contra-links free graphs since otherwise we cannot guarantee the transitivity. For instance, in the following graph hold  $\Lambda_1 \leq_g \Lambda_2$  and  $\Lambda_2 \leq_g \Lambda_3$  but not  $\Lambda_1 \leq_g \Lambda_3$ .



Since  $\leq_g$  is a partial order in contra-links free graphs and a graph is a finite thing, we can compute minimal elements with respect to  $\leq_g$ . This leads us to the following lemma:

**Lemma 5.4.8 (Minimal Elements).**

*Let be  $G$  a contra-links free refutation graph. If a simple multi-link is minimal in  $G$  with respect to  $\leq_g$  then it is also minimal in  $G$  with respect to  $<^*_L$ .*

**Proof:** Let  $\Lambda$  be a simple multi-link minimal in  $G$  with respect to  $\leq_g$ . Then there is no trail from its L-clause  $L$  to another simple multi-link that does not use  $\Lambda$ .

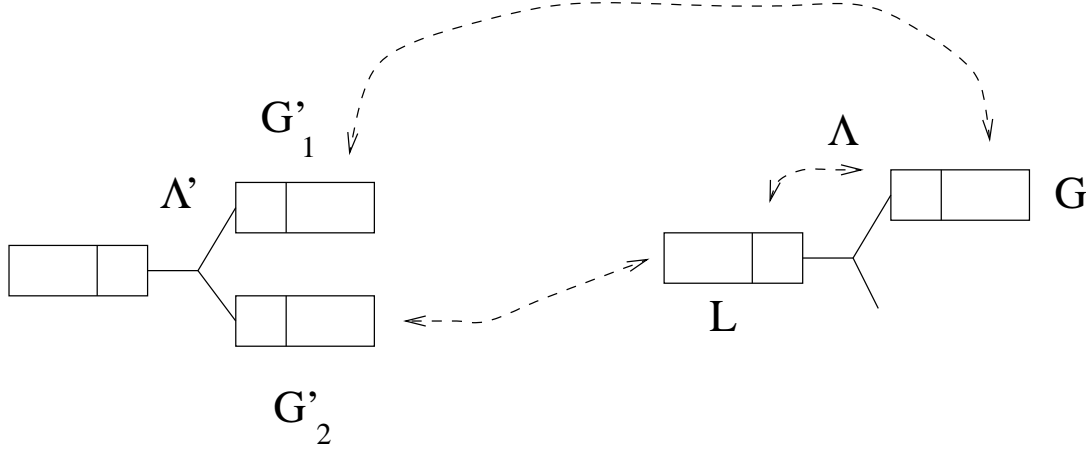
Assume there is a link  $\Lambda'$  such that  $\Lambda' <^*_L \Lambda$  and  $\Lambda' \neq \Lambda$ .

$<^*_L$  is the transitive and reflexive closure of  $<_L$ . Since we assume that  $\Lambda' \neq \Lambda$  there has to be a link which is  $<_L$  to  $\Lambda$ . We assume without loss of generality that  $\Lambda'$  is this link so that  $\Lambda' <_L \Lambda$  holds.

By definition of  $<_L$   $\Lambda'$  has to be also a multi-link; and since we assume in the whole section that we have only simple multi-links  $\Lambda'$  has to be a simple multi-link.

By definition of  $<_L$  we know from  $\Lambda' <_L \Lambda$  that there is a trail  $T_{G'_1, G'_2}$  from one G-clause  $G'_1$  of  $\Lambda'$  to another G-clause  $G'_2$  of  $\Lambda'$  using  $\Lambda$ . Since  $\Lambda$  is used we can split this trail into the following parts:

1. A trail  $T_{G'_1, G}$  from  $G'_1$  to a  $G$ -clause  $G$  of  $\Lambda$ .
2. A trail  $G, \Lambda, L$  from  $G$  to  $L$  using  $\Lambda$ .
3. A trail  $T_{L, G'_2}$  from  $L$  to  $G'_2$ .



Thereby, the trail  $T_{L, G'_2}$  cannot contain  $\Lambda$  (since otherwise it would be used twice in the trail  $T_{G'_1, G'_2}$ ; this would be a contradiction to the condition that  $T_{G'_1, G'_2}$  is a trail).

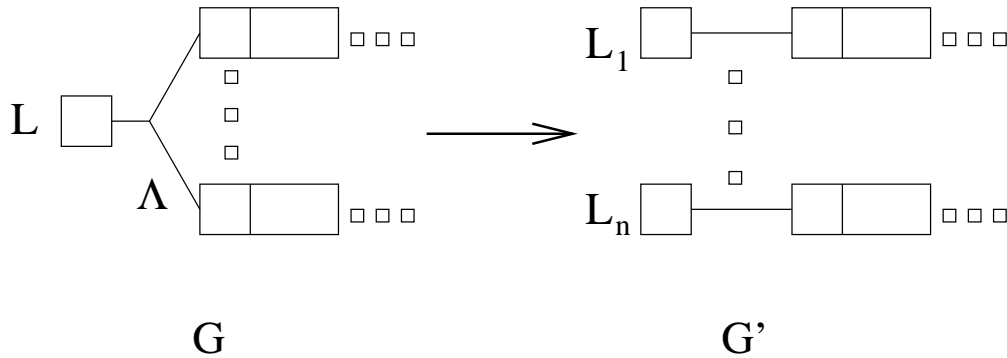
Thus, with  $T_{L, G'_2}$  we have a trail from  $\Lambda$  to  $G'_2$  and also to  $\Lambda'$  without using  $\Lambda$ .

Therefore, we would have  $\Lambda' \leq_g \Lambda$  what is contradiction to our hypothesis that  $\Lambda$  is minimal with respect to  $\leq_g$ .

So, our assumption was wrong and  $\Lambda$  is also minimal with respect to  $<^*_L$ .  $\square$

**Lemma 5.4.9 (Copying of Unit L-Clauses).**

Let  $G$  be a minimal and non empty refutation graph with a simple multi-link  $\Lambda$  such that the  $L$ -clause  $L$  of  $\Lambda$  is a unit clause. Let  $G'$  be the graph resulting from  $G$  if for each  $G$ -clause of  $\Lambda$  a copy of  $L$  is created and the  $G$ -clauses are connected by binary-links with these copies, respectively; then  $\Lambda$  itself is deleted.



It holds that if the resulting graph  $G'$  is UCS-decomposable, then  $G$  is UCS-decomposable, too.

**Proof:** If  $G$  is a minimal, non empty refutation graph then  $G'$  is also a minimal and non empty refutation graph (by Theorem 5.1.2).



We give here no formal proof of this lemma (a formal proof would exist of a induction over the number of the UCSs in a decomposition of  $G'$ ).

The idea is that from an UCS in  $G'$  using one of the copies of  $L$  as unit clause we can construct a corresponding UCS in  $G$  using  $L$  itself as unit clause.  $\square$

**Theorem 5.4.10 (UCS-Decomposition Theorem for Contra-Links Free Ref. Graphs).**

*Let  $G$  be a non empty, minimal, and contra-links free refutation graph. Then  $G$  is UCS-decomposable.*

**Proof:** Induction over the number  $n$  of simple multi-links in  $G$ .

**Induction Base Case:**  $n = 0$

Since  $G$  contains no simple multi-links it is a binary linked refutation graph which is UCS-decomposable by Theorem 5.2.8 (note that we still assume that we have no double multi-links).

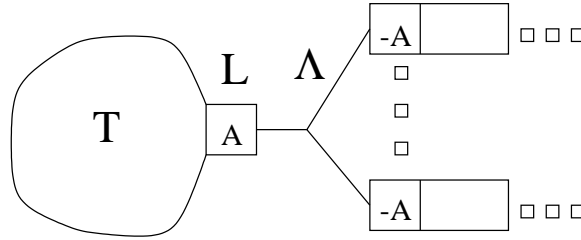
**Induction Step:**  $n \mapsto n + 1$

By Lemma 5.4.7  $\leq_g$  is a partial order in  $G$ . Since  $G$  is a finite object there have to be minimal links in  $G$  with respect to  $\leq_g$ .

Let  $\Lambda$  be such a simple multi-link minimal with respect to  $\leq_g$ . Then by Lemma 5.4.8  $\Lambda$  is also minimal in  $G$  with respect to  $<_L^*$ .

Then by Lemma 5.2.4  $\Lambda$  is separating in  $G$ .

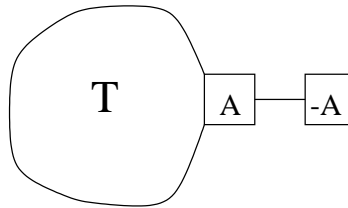
Furthermore, none of the G-clauses of  $\Lambda$  can be a unit clause (since otherwise  $G$  would not be minimal).



Let  $G'$  be the subgraph of  $G$  resulting from the deletion of the separating link  $\Lambda$  such that  $G'$  contains the L-clause  $L$  of  $\Lambda$ .

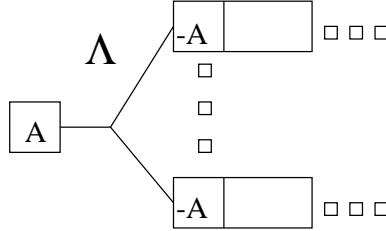
$G'$  cannot contain a multi-link. Otherwise there would be a trail from  $L$  to such an multi-link  $\Lambda'$  without using  $\Lambda$ . This would mean that  $\Lambda' \leq_g \Lambda$  what is a contradiction to the fact that  $\Lambda$  is minimal with respect to  $\leq_g$ .

$G'$  contains one pure literal, namely the literal of  $L$  previously in  $G$  connected by  $\Lambda$ . We add a unit clause to  $G'$  with the contradictory literal. Furthermore, we add a link connecting this two literals. Let  $G''$  be the resulting graph.



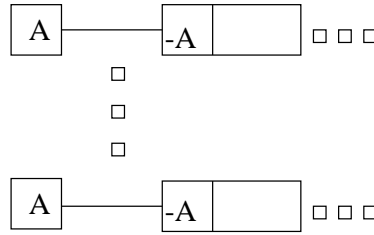
Then  $G''$  is again a refutation graph and it is binary-linked. By Theorem 5.2.8  $G''$  is UCS-decomposable. By applying the blocked UCS-decomposition Algorithm 5.3.1 and the according Lemma 5.3.2 we obtain that there is a UCS-decomposition  $D(G'') = \{\mathcal{U}_1, \dots, \mathcal{U}_m, E\}$  of  $G''$  with the end step  $E = ([A], [\perp A])$ .

These UCSs  $\mathcal{U}_1, \dots, \mathcal{U}_m$  exist also in  $G$ . If we replace them in  $G$  we obtain the following graph  $G'''$ :



Since the end step  $E = ([A], [\perp A])$  is not replaced in  $G$ ,  $\Lambda$  remains as simple multi-link. Next we perform the construction introduced in Lemma 5.4.9:

Let  $G''''$  be the graph resulting from  $G'''$  if for each G-clause of  $\Lambda$  a copy of  $L$  is created and the G-clauses of  $\Lambda$  are connected by binary-links with these copies, respectively. Afterwards, we delete  $\Lambda$ .



Then  $G''''$  is a refutation graph with  $n$  simple multi-links.

Thus, by induction hypothesis  $G''''$  is UCS-decomposable.

So, by Lemma 5.4.9  $G'''$  is UCS-decomposable, too.

Let  $D(G''') = \{\mathcal{U}'_1, \dots, \mathcal{U}'_l, E'\}$  be a UCS-decomposition of  $G'''$ .

Combining the decompositions of  $G''$  and  $G'''$  we obtain that  $G$  is UCS-decomposable with the decomposition  $D(G) = \{\mathcal{U}_1, \dots, \mathcal{U}_m, \mathcal{U}'_1, \dots, \mathcal{U}'_l, E'\}$ .

□

#### Corollary 5.4.11.

*The UCS-decomposition Algorithm 5.1.9 is complete on the class of minimal, non empty, and contra-links free refutation graphs.*

**Proof:** Follows directly from the proof of the UCS-decomposition theorem for contra-links free refutation graphs:

Since a contra-links free refutation graph is UCS-decomposable by Theorem 5.4.10 we can find an UCS in it.

By replacing this UCS we obtain a refutation graph that is again in the class of contra-links

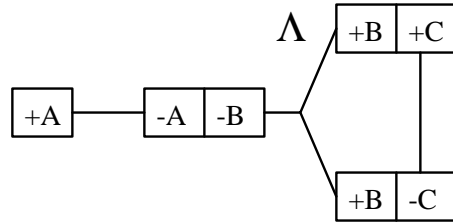
free graphs (since by the UCS-replacement no new multi-links are created).

Thus, this new graph is again UCS-decomposable by Theorem 5.4.10 and we can proceed with the UCS-decomposition algorithm.  $\square$

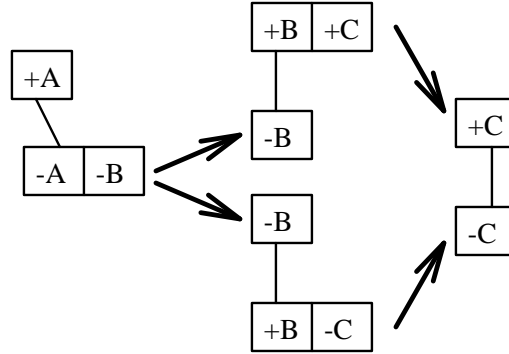
Note that the key point in this proof is the same as in the proof for completeness of the UCS-decomposition Algorithm 5.1.9 on the class of binary-linked refutation graphs (Corollary 5.2.9): It is an invariant of the application of a UCS-replacement that the graph resulting from the replacement is again in the same class as the original graph (here in this proof in the class of contra-links free graphs, in the proof of Corollary 5.2.9 in the class of binary-linked graphs).

*Example 5.4.12.*

We have the following refutation graph:

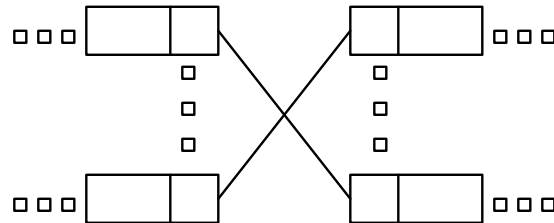


A UCS-decomposition for it is:

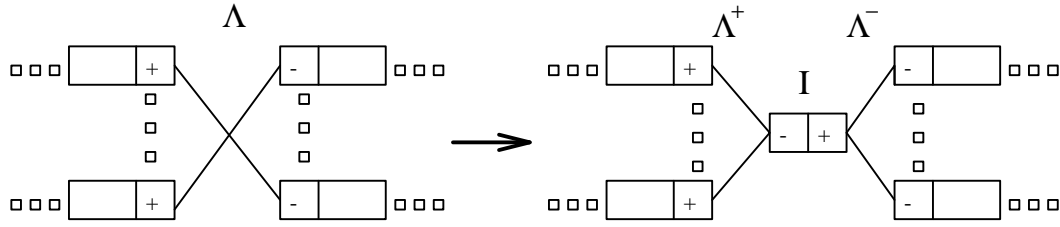


■

So far, we assumed that our refutation graphs contain only simple multi-links but no double multi-links. At the end of this section we extend our ideas to graphs containing also double multi-links. The following is a double multi-link:



If add into this double multi-link a clause with two appropriate literals we obtain from the one double multi-link two simple multi-links:



It is directly visible, that  $\Lambda^+$  and  $\Lambda^-$  are contra-directed links. With this construction in mind we can regard a double multi-link as two simple multi-links that are contra-directed. Therefore, We extend our definition of contra-directed links such that double multi-links are contra-directed to themselves.

Thus, we obtain that all lemmas of this section, the UCS-decomposition theorem for contra-links free refutation graphs and the corollary about the completeness of the UCS-decomposition algorithm, are still valid, since they contain explicitly the hypothesis that the graphs are contra-links free. In the next section we shall prove that refutation graphs with contra-directed links are not UCS-decomposable.

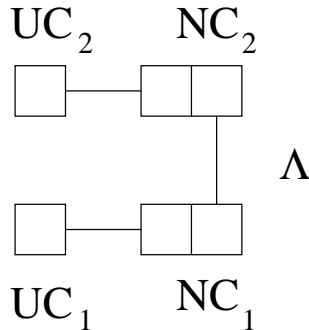
## 5.5 Not UCS-Decomposable Graphs

In the last section we showed that graphs with simple multi-links are UCS-decomposable if they are contra-links free. However, there is a fundamental difference between binary-links and simple multi-links: A binary-link can be used in a UCS-decomposition in both direction, that means each of its two literals can be used in a unit clause of a UCS. This is not the case for simple multi-links. We shall now formalize this intuitive observations.

### Definition 5.5.1 (Unit Offering).

Let  $G$  be a refutation graph and  $\Lambda$  a link in  $G$ . Then  $\Lambda$  is called *unit offering* from its one shore  $S_1$  to its other shore  $S_2$  if there exists a UCS-decomposition of  $G$  such that one of the literals of  $S_1$  is used in a unit clause of the UCS-decomposition. ■

*Example 5.5.2.*



The first possibility for a UCS-decomposition is to produce first a UCS with  $NC_2$  and  $UC_2$ . In this case  $\Lambda$  would afterwards offer the result literal (previously contained in  $NC_2$ ) as new unit clause to  $NC_1$  for further UCSs.

The second possibility for a UCS-decomposition is to produce first a UCS with  $NC_1$  and

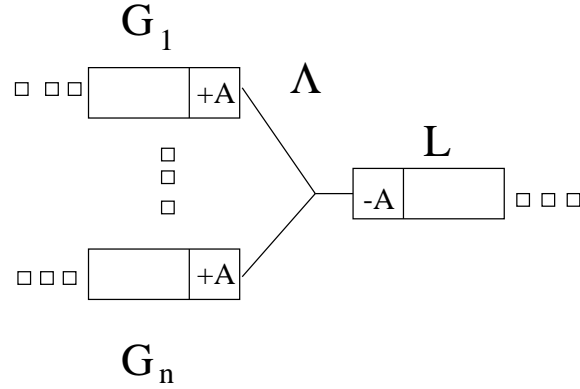
$UC_1$ . Then  $\Lambda$  would offer afterwards the result literal (previously contained in  $NC_1$ ) as new unit clause to  $NC_2$  for further UCSs. ■

In this example we see that a binary-link can be unit offering in both directions. In the next lemma we shall prove that this is not the case in a simple multi-link.

**Lemma 5.5.3.**

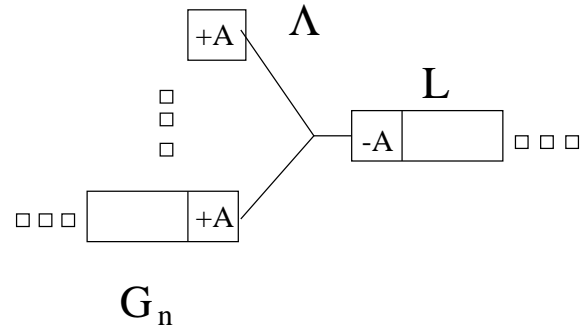
*Let  $G$  be a minimal refutation graph and  $\Lambda$  a simple multi-link in  $G$ . Then  $\Lambda$  can not be unit offering from its  $G$ -part to its  $L$ -part.*

**Proof:** Assume we have the following situation at  $\Lambda$ :



Neither  $G$ -clause  $G_1, \dots, G_n$  can be a unit clauses, since otherwise  $G$  would not be minimal.

Assume that we would obtain during the UCS-decomposition a situation in that  $+A$  of  $G_1$  is the result literal of a UCS. If we would then perform the UCS-replacement with this UCS we would obtain the following graph:



In this graph  $\Lambda$  could offer a unit clause from its  $G$ -part to its  $L$ -part, but this graph would not be minimal. Hence, we have a contradiction to Lemma 5.1.3 about the correctness of the UCS-replacement (stating that by a UCS-replacement we obtain from an minimal refutation graph again a minimal refutation graph).

Thus, during a UCS-decomposition a simple multi-link can never offer a unit clause from its  $G$ -part to its  $L$ -part. □

From this lemma we obtain the following corollary.

**Corollary 5.5.4.**

*Let  $G$  be a minimal refutation graph and  $\Lambda$  a double multi-link in  $G$ . Then  $\Lambda$  is not unit offering in any direction and hence  $G$  is not UCS-decomposable.*

**Proof:** That a double multi-link  $\Lambda$  is not unit offering in any direction follows from Lemma 5.5.3.

Since in a UCS-decomposition *each* link has to be unit offering in (at least) one direction (since we are ending with an end step), it follows that graphs with double multi-links are not UCS-decomposable.  $\square$

**Example 5.5.5.**

Look again at Example 5.4.12. The simple multi-link  $\Lambda$  is unit offering from its L-part to its G-part. But there is no UCS-decomposition such that  $\Lambda$  is unit offering in the other direction.  $\blacksquare$

In the following we shall extend the result of Corollary 5.5.4 to all graphs with contra-directed links by proving that a graph with contra-directed links is not UCS-decomposable. We start with proving a needed lemma.

**Lemma 5.5.6.**

*Let  $G$  be a refutation graph. Two simple multi-links  $\Lambda_1$  and  $\Lambda_2$  in  $G$  with L-clauses  $L_1$  and  $L_2$ , respectively, are contra-directed if and only if there is a trail from  $L_1$  to  $L_2$  without using  $\Lambda_1$  and  $\Lambda_2$ .*

**Proof:**

$\Leftarrow$ : This direction holds by definition of contra-directed links and the relation  $\leq_g$ .

$\Rightarrow$ : It holds that  $\Lambda_1 \leq_g \Lambda_2$  and  $\Lambda_2 \leq_g \Lambda_1$ . By expanding the definition of  $\leq_g$  we obtain that there is a trail  $T_{21}$  from  $L_2$  to  $\Lambda_1$  without using  $\Lambda_2$  and a trail  $T_{12}$  from  $L_1$  to  $\Lambda_2$  without using  $\Lambda_1$ .

Furthermore, by Lemma 5.4.6 we can assume without loss of generality that  $T_{21}$  does also not contain  $\Lambda_1$  and that  $T_{12}$  does also not contain  $\Lambda_2$ .

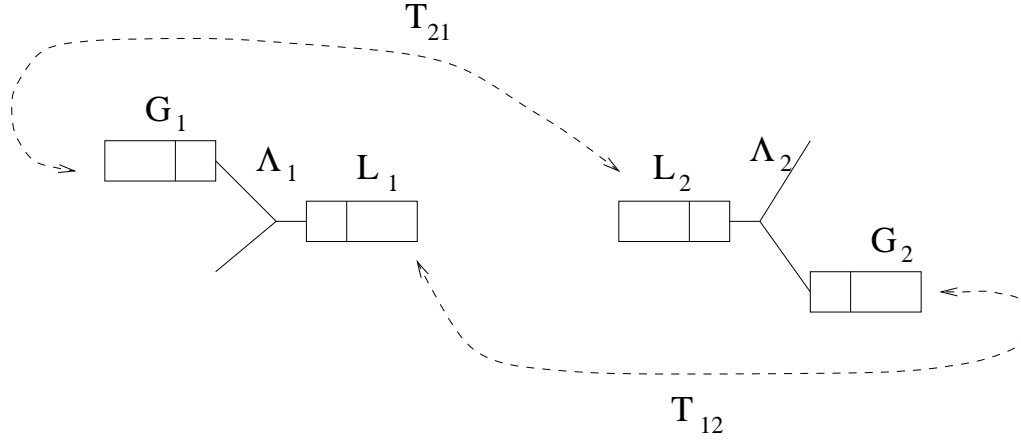
**Case 1:**  $T_{21}$  or  $T_{12}$  are such a trail from  $L_1$  to  $L_2$ .

Then we are finished.

**Case 2:** Neither  $T_{21}$  nor  $T_{12}$  are such a trail from  $L_1$  to  $L_2$ .

Then  $T_{21}$  connects  $L_2$  and a G-clause  $G_1$  of  $\Lambda_1$ ; and  $T_{12}$  connects  $L_1$  and a G-clause  $G_2$  of  $\Lambda_2$ .

Furthermore, neither  $T_{21}$  nor  $T_{12}$  uses  $L_1$  or  $L_2$  (\*).



Assume  $T_{21}$  and  $T_{12}$  share no links.

Then we would have a cycle by connecting  $T_{21}$  and  $T_{12}$  over  $\Lambda_1$  and  $\Lambda_2$ .

This would be a contradiction to the hypothesis that  $G$  is a refutation graph (and therefore acyclic).

So,  $T_{21}$  and  $T_{12}$  have at least one common link.

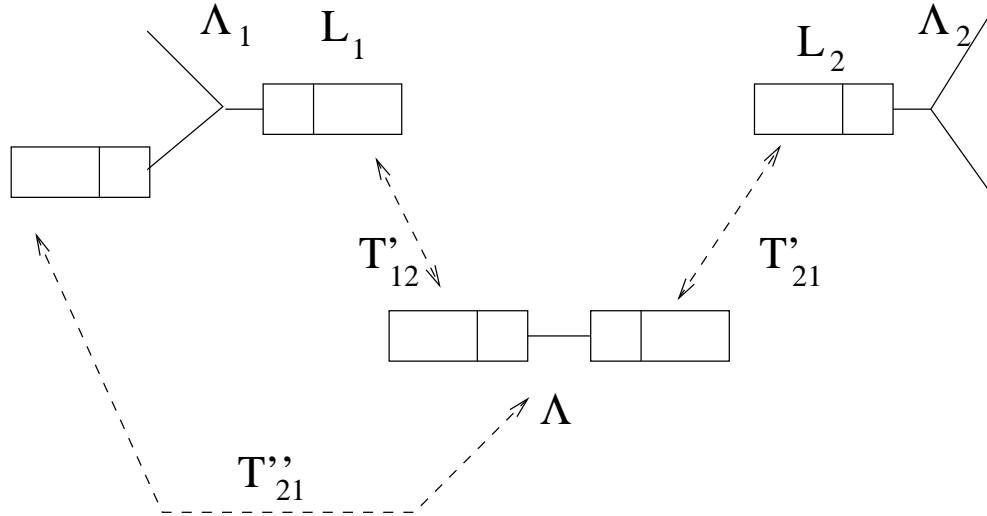
Let  $\Lambda$  be the first such common link in  $T_{12}$ .

Then  $\Lambda$  can be neither  $\Lambda_1$  nor  $\Lambda_2$  (see (\*)) (\*\*).

Let be  $T'_{21}$  the subtrail of  $T_{21}$  from  $L_2$  to  $\Lambda$ ,  $T''_{21}$  the subtrail of  $T_{21}$  from  $\Lambda$  to  $G_1$ , and  $T'_{12}$  the subtrail of  $T_{12}$  from  $L_1$  to  $\Lambda$ .

Since  $\Lambda$  is the *first* common link in  $T_{12}$ ,  $T'_{12}$  shares no link with  $T'_{21}$  or  $T''_{21}$  (\*\*\*) .

Moreover, neither  $T'_{12}$ , nor  $T'_{21}$ , nor  $T''_{21}$  contain  $\Lambda$  (\*\*\*\*).



Assume  $T'_{12}$  and  $T'_{21}$  would end at the same shore of  $\Lambda$ .

Then  $T''_{21}$  would end at the opposite shore of  $\Lambda$  and we could create the following trail:

Starting at  $L_1$  we could reach  $\Lambda$  via  $T'_{12}$ , then we could reach  $G_2$  via  $T'_{21}$  and then  $L_1$  via  $\Lambda_1$  (this would be a trail since no link would be used double, see (\*), (\*\*), (\*\*\*)).

Thus, we would have a cycle what would be a contradiction to the fact that  $G$  is

a refutation graph.

Therefore, we have that  $T'_{12}$  and  $T'_{21}$  end at opposite shores of  $\Lambda$ .

Then we can create the following trail:

Starting at  $L_1$  we can reach  $\Lambda$  via  $T'_{12}$ , then we can reach  $L_2$  via  $T'_{21}$  (this is a trail since no link is used double, see  $(*)$ ,  $(***)$ ,  $(****)$ ).

So, we have a trail from  $L_1$  to  $L_2$  without using  $\Lambda_1$  and  $\Lambda_2$  (see also  $(*)$ ,  $(**)$ ).

□

With this lemma we can prove the following theorem:

**Theorem 5.5.7 (Not UCS-Decomposable Graphs).**

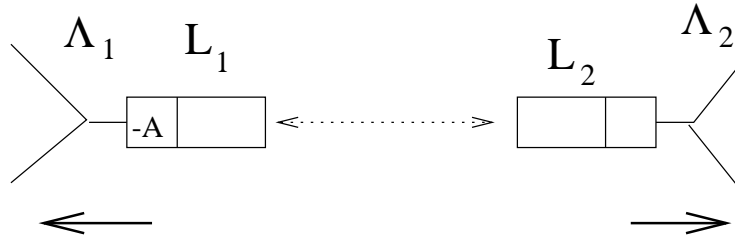
*Let  $G$  be a non empty and minimal refutation graph. If  $G$  contains contra-directed links then  $G$  is not UCS-decomposable.*

**Proof:** Let  $\Lambda_1$  and  $\Lambda_2$  be two contra-directed links in  $G$ . We assume – without loss of generality – that  $\Lambda_1$  and  $\Lambda_2$  are both simple multi-links and not double multi-links (the case for double multi-links is already handled in Corollary 5.5.4).

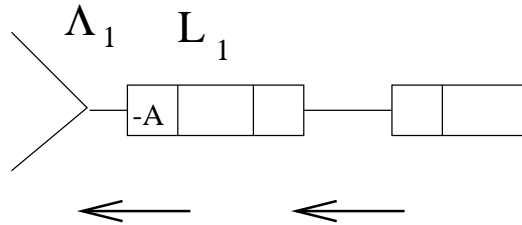
Let  $L_1$  and  $L_2$  be the L-clauses of  $\Lambda_1$  and  $\Lambda_2$ , respectively. By Lemma 5.5.6 there is a trail  $T$  connecting  $L_1$  to  $L_2$  without using  $\Lambda_1$  or  $\Lambda_2$ .

Assume there is a UCS-decomposition  $D(G) = \{\mathcal{U}_1, \dots, \mathcal{U}_n, E\}$  of  $G$ .

Then by Lemma 5.5.3 both  $\Lambda_1$  and  $\Lambda_2$  have to be unit offering in the decomposition from their L-clauses to their G-clauses, respectively.



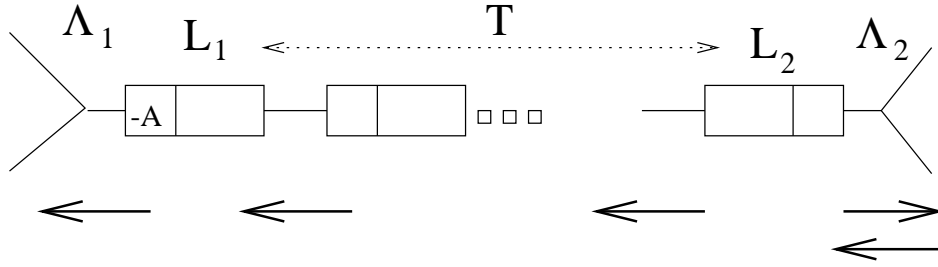
That  $\Lambda_1$  can be unit offering from  $L_1$  to its G-clauses,  $\perp A$  has to be the result literal of a UCS with the assertion clause  $L_1$ . Then all links (except  $\Lambda_1$ ) on  $L_1$  have to be unit offering to  $L_1$ .



Following the trail  $T$  from  $L_1$  to  $L_2$  this direction in which the links have to be unit offering is propagated along  $T$ .

But then  $\Lambda_2$  would have to be unit offering in both directions in one UCS-decomposition, since for  $\Lambda_2$  holds the same as for  $\Lambda_1$ .



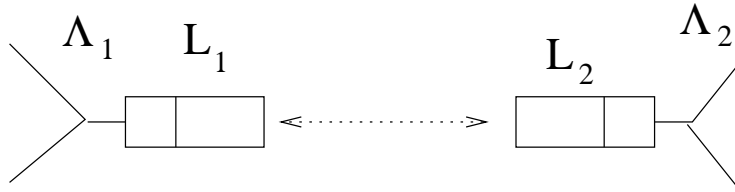


This would mean that  $\Lambda_2$  is unit offering from its G-clauses to its L-clause what is not possible because of Lemma 5.5.3.

So, we have a contradiction to the assumption that we have a UCS-decomposition for  $G$  and obtain that  $G$  is not UCS-decomposable.  $\square$

## 5.6 Conclusion

We will summarize now the content of the last sections. In the previous sections we proved on the one hand that refutation graphs containing no contra-directed links are UCS-decomposable (Theorem 5.4.10) and on the other hand that refutation graphs with contra-directed links are not UCS-decomposable (Theorem 5.5.7). Thereby, there are two cases of contra-directed links: (1) a link whose shores have both a cardinality bigger than one is contra-directed to itself. (2) Two links  $\Lambda_1$  and  $\Lambda_2$  are contra-directed if both have one shore with a cardinality bigger than one and one shore with cardinality one, and their clauses at the shore with cardinality one, respectively, are connected by a trail.



By combining Theorem 5.5.7 and Theorem 5.4.10 we obtain the following theorem:

### Theorem 5.6.1 (UCS-Decomposition Theorem).

*Let  $G$  be a non empty and minimal refutation graph. Then  $G$  is UCS-decomposable if and only if  $G$  is contra-links free.*

From this theorem follows that the class of contra-links free graphs is the maximal class of UCS-decomposable graphs. Since we proved in Corollary 5.4.11 that the UCS-decomposition Algorithm 5.1.9 is complete on this class, we have that the UCS-decomposition algorithm is complete on all UCS-decomposable refutation graphs.

### Theorem 5.6.2 (Completeness of the UCS-Decomposition Algorithm).

*Let  $G$  be a UCS-decomposable graph. Then the UCS-decomposition Algorithm 5.1.9 computes a UCS-decomposition for  $G$ .*

We found in this chapter a criterion to decide whether a refutation graph is UCS-decomposable or not. If the refutation graph contains contra-directed links it is not UCS-decomposable; if

the graph contains no contra-directed links it is UCS-decomposable and our UCS-decomposition algorithm can compute a UCS-decomposition for it. We call this criterion the *UCS-decomposition criterion*. Moreover, note that it was a essential premise in many proofs of this chapter that the refutation graphs are minimal. This emphasizes again the necessity of transformation invariant 1.

In the next chapter we present a transformation algorithm based on UCS-decomposable graphs with a UCS-decomposition in that each UCSs represents an assertion applications. In this transformation algorithm it is important to recognize when a graph is reached that is UCS-decomposable (whether then all their UCSs represent assertion applications is another question) and to reduce not UCS-decomposable graphs to UCS-decomposable graphs. We shall see in the next chapter how the results of this chapter are used in this transformation algorithm.

## Chapter 6

# A UCS-Based Transformation Algorithm

We presented in the last but one chapter the assertion level steps both in ND-proofs and in the refutation graph. In particular, we were described the UCS, a structure in a refutation graph that represents under certain conditions (see UCS-Theorem 4.3.3) an assertion application. Thus, the idea for a new transformation algorithm was to translate such UCSs directly into assertion applications in the ND-proof. In the last chapter we described which graphs consist only of a sequence of UCSs, we call such graphs UCS-decomposable graphs (see Definition 5.1.5) and how we obtain such a UCS-decomposition of a graph (see Algorithm 5.1.9).

In this chapter we present the transformation algorithm based on the translation of UCSs representing assertion application into assertion applications in the ND-proof. We construct this algorithm like the literal transformation algorithm (see Algorithm 3.4.1. We describe first the base case of our new algorithm and how it is transformed into the ND-proof (see the next section) and then how an arbitrary case is reduced to a set of these base cases (see Section 6.2 and Section 6.3). The base case of our new algorithm consists of UCS-decomposable refutation graphs such that each UCS in a UCS-decomposition represents an assertion application. These UCSs are then translated stepwise into corresponding assertion applications in the ND-proof. Furthermore, we describe how we can reduce an arbitrary transformation problem into a transformation problem consisting only of a set of base cases by applying transformation rules in a goal-directed way. These new base cases can be much more complex as the literal base cases of the literal transformation algorithm, so, in general, we need to apply significantly less transformation rules to reach these base cases than in the literal transformation algorithm. Finally, in the last section we compose the single parts developed in this chapter to a complete transformation algorithm based on transformation of UCSs at the assertion level.

### 6.1 The UCS-Decomposable Base Case

In the UCS-theorem we showed that a UCS that satisfies two conditions – called  $C_1$  and  $C_2$  in the UCS-theorem – represents an assertion application. Assume we have a UCS  $\mathcal{U} = (AC, \{UC_1, \dots, UC_n\}, R_{Lit})$  satisfying the conditions  $C_1$  and  $C_2$  of the UCS-Theorem 4.3.3. Then we know (by the UCS-theorem) that  $\mathcal{U}$  represents an assertion application

$$\frac{(R_{Lit})\sigma}{L_1 \dots L_n} A$$

where  $L_i$  ( $1 \leq i \leq n$ ) is the unique literal of  $UC_i$ , respectively,  $\sigma$  is the ground substitution of  $AC$ , and  $A$  is a formula, such that  $AC$  is in the CNF of  $A$ .  $C_1$  demands that  $UC_1, \dots, UC_n$  are ground and  $C_2$  demands that in the clause normalization from  $A$  to  $AC$  the  $\delta$ -rule is not used. In a transformation problem the clauses  $AC, UC_1, \dots, UC_n$  are connected by a delta-relation with ND-lines  $N_A, N_{U_1}, \dots, N_{U_n}$  such that the clauses are in the CNF of the ND-lines, respectively (invariant 3). Then, not unexpectedly, we want to transform this single UCS by an assertion application of its corresponding ND-lines:

$$\frac{(R_{Lit})\sigma}{N_{U_1} \dots N_{U_n}} N_A$$

A whole UCS-decomposable refutation graph we want to transform in two steps: First, we compute an UCS-decomposition of it (by Algorithm 5.1.9) and then we translate these UCSs – that should all represent assertion applications – stepwise into corresponding assertion steps in the ND-proof. We give first an algorithm that can transform a UCS-decomposable graphs in this way and explain then later under which conditions it works correctly (i.e., conditions guaranteeing that each UCS in the UCS-decomposition represents an assertion application and that we can be translate this UCS into an assertion application of the ND-lines connected with its clauses).

So far, we did not care about the difference between such clauses connected with closed ND-lines and such clauses connected with open lines. Assume in a UCS  $\mathcal{U}$  one of the clauses is connected with the open node justified by the refutation graphs containing  $\mathcal{U}$ . Then we cannot transform this UCS into an assertion step in the ND-proof since then one of our premises or the assertion would be the open line we want to close. Therefore, we use in the first approach of the algorithm handling the transformation of UCS-decomposable graphs an initialization step that creates indirect proof parts by introducing the negation of the open line as hypothesis. After this initialization step all clauses are connected with closed ND-lines. We can now formulate the so-called *base algorithm for UCS-decomposable graphs*.

Case 1: Formula of open line is not a negation.

$$\begin{array}{llll} L. & \mathcal{A} & \vdash F & (W) \\ & & \text{we obtain} & \\ L'. & L' & \vdash \neg F & (Hyp) \\ L''. & \mathcal{A}, L' & \vdash \perp & (W) \\ L. & \mathcal{A} & \vdash F & (IP_2 \ L'') \end{array}$$

Case 2: Formula of open line is a negation.

$$\begin{array}{llll} L. & \mathcal{A} & \vdash \neg F & (W) \\ & & \text{we obtain} & \\ L'. & L' & \vdash F & (Hyp) \\ L''. & \mathcal{A}, L' & \vdash \perp & (W) \\ L. & \mathcal{A} & \vdash F & (IP_1 \ L'') \end{array}$$

Figure 6.1: Indirect proofs by negation of the open line.

**Algorithm 6.1.1 (Base Algorithm for UCS-Decomposable Graphs).**

Let  $G$  be a UCS-decomposable graph such that  $G$  justifies an open ND-line  $L$  in a not closed ND-proof  $\mathcal{N}$ . Furthermore, let  $\Delta$  be the  $\Delta$ -relation connecting the clauses of  $G$  and ND-lines in  $\mathcal{N}$ .

**Initialization:** If the formula of  $L$  is

1.  $\perp$  then do nothing,
2. **a negation** then apply  $IP_2$  backwardly on  $L$  to obtain an indirect proof part with a new open line  $L'$  (see Figure 6.1),
3. **a negation** then apply  $IP_1$  backwardly on  $L$  to obtain an indirect proof part with a new open line  $L'$  (see Figure 6.1).

In the case of 2 or 3 change  $\Delta$  such that clauses previously connected with  $L$  are afterwards connected with the new hypothesis line.

**UCS-Transformation:** Compute a UCS-decomposition  $D(G) = \{\mathcal{U}_1, \dots, \mathcal{U}_n, E\}$  of  $G$  (use Algorithm 5.1.9).

For  $i = 1$  to  $n$  do:

Translate the UCS  $\mathcal{U}_i = (AC, \{UC_{1_i}, \dots, UC_{n_i}\}, R_{Lit_i})$  ( $1 \leq i \leq n$ ) into an assertion application of the ND-lines  $N_{A_i}, N_{U_{1_i}}, \dots, N_{U_{n_i}}$  connected with the clauses of  $\mathcal{U}_i$ , respectively ( $N_{A_i}$  with  $AC$ ,  $N_{U_{1_i}}, \dots, N_{U_{n_i}}$  with  $UC_{1_i}, \dots, UC_{n_i}$ ):

$$\frac{N_{U_{1_i}} \dots N_{U_{n_i}}}{(R_{Lit_i})\sigma_i} N_{A_i}$$

Connect in  $\Delta$  the resulting unit clause consisting of the result literal  $(R_{Lit_i})\sigma_i$  with the new ND-line justified by the assertion application.

**End-Step-Transformation:** Transform the end step  $E = ([+A], [\perp A])$  of the UCS-decomposition into a justification for the open line using the rule *Contradiction* on the lines  $L_{E_1}$  and  $L_{E_2}$  connected with the clauses of this end step:

$$L'. \quad \mathcal{A} \quad \vdash \perp \quad (Contradiction \ L_{E_1} \ L_{E_2})$$

■

To have a literal base case it is not enough that the refutation graph consists only of two linked unit clauses, but the ND-lines connected with these clauses have also to satisfy some conditions (the formulas of the ND-lines have to be literal, see Section 3.1). Similarly, to be able to apply the base algorithm for UCS-decomposable graphs we need not only to have a UCS-decomposable graph, but the ND-lines connected with the clauses of the graph have also to satisfy some conditions that we shall describe in the following. We start with regarding only a single UCS in a UCS-decomposition and describe which conditions the ND-lines connected with the clauses of the UCS have to satisfy that we can translate the UCS into an assertion application of these ND-lines. Then we describe the conditions that the ND-lines connected

with the clauses of a whole UCS-decomposable graph have to satisfy so that we can apply (correctly) the base algorithm for UCS-decomposable graphs.

Assume we have a single UCS  $\mathcal{U} = (AC, \{UC_1, \dots, UC_n\}, R_{Lit})$  in a UCS-decomposition that satisfies the conditions  $C_1$  and  $C_2$  of the UCS-theorem. The clauses of  $\mathcal{U}$  are connected with the ND-lines  $N_A, N_{U_1}, \dots, N_{U_n}$  by a  $\Delta$ -relation. Which conditions have to hold for  $N_A, N_{U_1}, \dots, N_{U_n}$  that we can derive  $(R_{Lit})\sigma$  with the assertion application

$$\frac{N_{U_1} \dots N_{U_n}}{(R_{Lit})\sigma} N_A \quad ?$$

We can extend the conditions  $C_1$  and  $C_2$  of the UCS-theorem to these ND-lines:

$C_1 \mapsto C'_1$  The formulas of the ND-lines  $N_{U_i}$  ( $1 \leq i \leq n$ ) have to be the ground literals  $L_i$ , respectively. Note that this condition subsumes the condition  $C_1$  on the literals  $L_i$  since, if the formula of  $N_{U_i}$  is ground and literal, then all clauses connected with this ND-line are unit clauses and the literals are ground (by invariant 3).

$C_2 \mapsto C'_2$  In the clause normalization from the formula of  $N_A$  to the clause  $AC$  the  $\delta$ -rule is not used.

If these conditions are satisfied the formulas in the ND-lines correspond directly to the formulas in the UCS. Hence, by the UCS-theorem we have also an assertion application in the ND-proof. In line-notation we obtain the following ND-line:

$$L. \quad \mathcal{A} \quad \vdash (R_{Lit})\sigma \quad (N_A \ N_{U_1} \dots N_{U_n})$$

where the justification  $(N_A \ N_{U_1} \dots N_{U_n})$  is the application of the assertion  $N_A$  on the premises  $N_{U_1}, \dots, N_{U_n}$ . This new line can be connected in the  $\Delta$ -relation with the new unit clause containing the literal  $(R_{Lit})\sigma$  since this connection satisfies invariant 3. Beside the conditions  $C_1$  and  $C_2$  we have to demand another condition forbidding the usage of skolem terms in the ND-proof. Since we avoid always the use of skolem term in the ND-lines, the formulas of the lines selves contain no skolem terms. Hence, if condition  $C'_1$  is justified  $L_1, \dots, L_n$  can contain no skolem terms. Furthermore, since the application of the  $\delta$ -rule is forbidden in the clause normalization from  $N_A$  to  $AC$ ,  $AC$  can contain no skolem terms. The only possible source for skolem terms is the ground substitution of  $AC$   $\sigma$ . Therefore, we demand:

$C'_3$  The ground substitution of  $AC$  contains no skolem terms.

Next, we examine which conditions the ND-lines connected with a whole UCS-decomposable refutation graph have to satisfy so that we can apply correctly the base algorithm. Obviously, each UCS and its corresponding ND-lines has to satisfy the conditions  $C'_1$ ,  $C'_2$ , and  $C'_3$  so that we can translate it into a corresponding assertion application in the ND-proof. Therefore, we extend the conditions  $C'_1, C'_2, C'_3$  to the conditions  $GC_1, GC_2$ , and  $GC_3$  on a whole graph  $G$ :

$GC_1$  All unit clauses of  $G$  are connected by the  $\Delta$ -relation with ND-lines such that the formula of the ND-line is the unique literal of the unit clause and ground, respectively.

*GC<sub>2</sub>* All non unit clauses of  $G$  are connected by the  $\Delta$ -relation with ND-lines such that during the clause normalization from the formula of the ND-line to the clause the  $\delta$ -rule is not used.

*GC<sub>3</sub>* No ground substitution of one of the non unit clauses contains skolem terms (the ground substitutions of the unit clauses are already empty, because of *GC<sub>1</sub>*).

These conditions guarantee that for each single UCS the conditions  $C'_1, C'_2, C'_3$  hold, because of the following reasons:

*GC<sub>1</sub>*  $\mapsto C'_1$  During the UCS-transformation of the UCS-decomposition of a graph  $G$  there are two possibilities for a unit clauses  $UC$  of a UCS  $\mathcal{U}$ :

1.  $UC$  is also in the graph  $G$  a unit clause. Then by *GC<sub>1</sub>*  $C'_1$  is satisfied for  $UC$ .
2. We derived  $UC$  as result of the translation of a UCS  $\mathcal{U}'$  earlier in the UCS-transformation. Then  $UC$  is connected – because of the construction of the base algorithm – with the ND-line  $L$  that results from the transformation of  $\mathcal{U}'$ . But then the formula of  $L$  is ground and literal and equals the unique literal of  $UC$ . Hence,  $C'_1$  is satisfied for  $UC$  and  $L$ .

*GC<sub>2</sub>*  $\mapsto C'_2$   $C'_2$  follows directly for each single UCS from *GC<sub>2</sub>*.

*GC<sub>3</sub>*  $\mapsto C'_3$   $C'_3$  follows directly for each single UCS from *GC<sub>3</sub>*.

For a whole graph  $G$  an explicit demand for *GC<sub>3</sub>* is not necessary since *GC<sub>3</sub>* is implicitly satisfied if *GC<sub>1</sub>* and *GC<sub>2</sub>* are satisfied. All skolem terms both in the clauses and the ground substitution arise by clause normalization of the formulas of the ND-lines connected with the clauses of the refutation graph. Precisely, they arise by applications of the  $\delta$ -rule during the clause normalization. But, *GC<sub>1</sub>* and *GC<sub>2</sub>* guarantee already that in the clause normalizations producing both the unit and non unit clauses of  $G$  the  $\delta$ -rule is not applied, respectively. Thus, a graph satisfying *GC<sub>1</sub>* and *GC<sub>2</sub>* can contain no skolem terms, neither in the clauses nor in the ground substitutions.

Using the graph conditions *GC<sub>1</sub>* and *GC<sub>2</sub>* we can state the following theorem about the correctness of the base algorithm:

**Theorem 6.1.2 (Correctness of the Base Algorithm).**

*Given a transformation problem  $(\mathcal{N}, \{G_1, \dots, G_{i-1}, G_i, G_{i+1}, \dots, G_n\}, \Delta)$  such that  $G_i$  is a minimal, UCS-decomposable refutation graph justifying a ND-line  $L$  in  $\mathcal{N}$ . Furthermore,  $G_i$  and the ND-lines of  $\mathcal{N}$  connected with the clauses of  $G_i$  satisfy the following conditions (called graph conditions):*

*GC<sub>1</sub>* All unit clauses of  $G_i$  are connected by  $\Delta$  with ND-lines of  $\mathcal{N}$  such that the formula of the ND-line is the unique literal of the unit clause and ground, respectively.

*GC<sub>2</sub>* All non unit clauses of  $G_i$  are connected by  $\Delta$  with ND-lines of  $\mathcal{N}$  such that during the clause normalization from the formula of the ND-line to the clause the  $\delta$ -rule is not used.

Then the application of the base algorithm for UCS-decomposable graphs (Algorithm 6.1.1) on  $G_i$  computes a sequence of correctly justified ND-steps closing  $L$ . The resulting transformation problem is  $(N', \{G_1, \dots, G_{i-1}, G_{i+1}, \dots, G_n\}, \Delta')$  where  $\Delta'$  results from  $\Delta$  by removing all connections with clauses of  $G_i$ .

**Proof:** In a sequence of UCSs resulting from the UCS-decomposition of a graph we call a unit clause *internal* if it is the result of another UCS in this sequence. The other unit clauses we call *external*. When a UCS is translated during the UCS-transformation step then the unit clause resulting from this UCS becomes external in the remaining sequence of UCSs.

By induction over the number  $m$  ( $0 \leq m \leq n$ ) of transformed UCSs during the UCS-transformation step we can show that:

- (\*) each unit clause that is external in the remaining UCS-decomposition is connected by  $\Delta$  with a ND-line containing the unique literal of the clause as formula.

**Induction Base Case:**  $m = 0$

(\*) holds because of two reasons: First, since  $GC_1$  holds; secondly, since the initialization-step introduces the new hypothesis such that also for this new ND-line and the clauses connected with it (\*) holds. Note that the negation-step explicitly distinguishes two cases to guarantee (\*).

**Induction Step:**  $m \mapsto m + 1$

How described earlier in this section, for each remaining external unit clause  $UC$  there exist two possibilities:

1.  $UC$  was also in the original UCS-decomposition an external unit clause. Then (\*) holds for  $UC$  for the same reasons as described in the induction base case (since we do not change the connection of  $UC$  during the UCS-transformation step).
2.  $UC$  is derived as result of the translation of a UCS  $U'$  earlier in the UCS-transformation. Then – by the construction of the base algorithm –  $UC$  is connected by  $\Delta$  with the ND-line  $L$  resulting from the translation of  $U'$ . But then the formula of  $L$  is ground and literal and equals the unique literal of  $UC$ . Hence, (\*) is satisfied for  $UC$  and  $L$ .

Because of (\*),  $GC_1$ , and  $GC_2$  each single UCS in the UCS-decomposition satisfies the conditions of the UCS-Theorem 4.3.3. Hence, by the UCS-Theorem the translation of the UCSs into the ND-proof produces correctly justified ND-lines.

Moreover, the ND-lines created in the initialization-step and the end-step-transformation are also correctly justified ((\*) guarantees that the lines connected with the clauses of the end-step contain contradictory formulas).  $\square$

The base cases of the UCS-based transformation algorithm we present in this chapter are refutation graphs and corresponding ND-lines that satisfy the conditions of Theorem 6.1.2. We call this new base case the *UCS base case*. We demonstrate now the use of the base algorithm for UCS-decomposable graphs by applying it on our standard example.

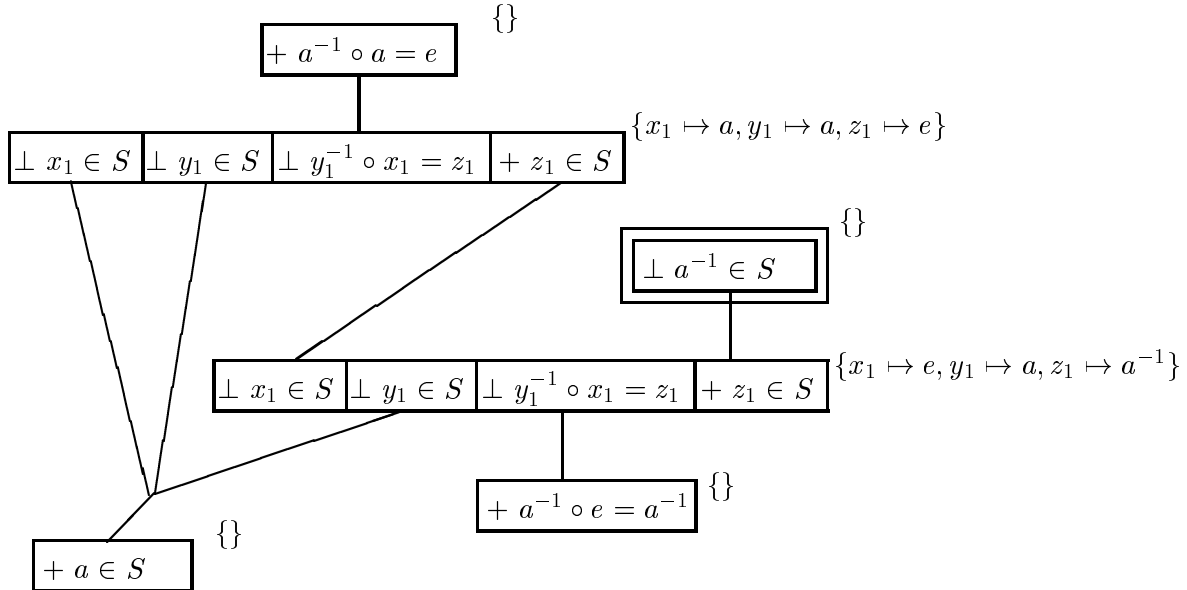


*Example 6.1.3 (Standard Example (Continuation)).*

In the initial transformation problem of the standard example the conditions of Theorem 6.1.2 are not satisfied (we presented the initial refutation graph in Figure 2.1, the initial ND-proof in Example 2.5.2, and the initial  $\Delta$ -relation is in Example 2.5.3). The refutation graph is UCS-decomposable (it contains no contra-directed links) and  $GC_2$  is satisfied by the two non unit clauses and the ND-line  $L_3$  connected with them, but there are violations of  $GC_1$ . In fact, no unit clause is connected with a ND-line that contains exactly the literal of the unit clause as formula, for instance the unit clause  $[+sk_1 \in S]$  is connected with the ND-line  $L_4$ .

Therefore, we have to apply first some transformation rules transforming this initial transformation problem into a UCS base case. To be able to apply here the base algorithm for UCS-decomposable graphs we assume that we obtained here directly a transformation problem consisting of the following ND-proof, refutation graph  $G''$ , and the obvious  $\Delta$ -relation connecting them. We describe in the next section which steps we have to apply and why we apply exactly these steps to transfer the initial transformation problem into this transformation problem (see Example 6.2.3).

$L_6.$	$L_6$	$\vdash a \in S$	(Hyp)
$L_1.$	$L_1$	$\vdash \forall u.(u \circ e = u)$	(Hyp)
$L_8.$	$L_1$	$\vdash a^{-1} \circ e = a^{-1}$	( $\forall E$ $L_1$ )
$L_2.$	$L_2$	$\vdash \forall w.(w^{-1} \circ w = e)$	(Hyp)
$L_9.$	$L_2$	$\vdash a^{-1} \circ a = e$	( $\forall E$ $L_2$ )
$L_3.$	$L_3$	$\vdash \forall x, y, z.((x \in S \wedge (y \in S \wedge y^{-1} \circ x = z)) \Rightarrow (z \in S))$	(Hyp)
$L_7.$	$L_6, L_1, L_2, L_3$	$\vdash a^{-1} \in S$	( $G''$ )
$L_5.$	$L_1, L_2, L_3$	$\vdash a \in S \Rightarrow a^{-1} \in S$	( $\Rightarrow I$ $L_6$ $L_7$ )
$L_4.$	$L_1, L_2, L_3$	$\vdash \forall v.(v \in S \Rightarrow v^{-1} \in S)$	( $\forall I$ $L_5$ )

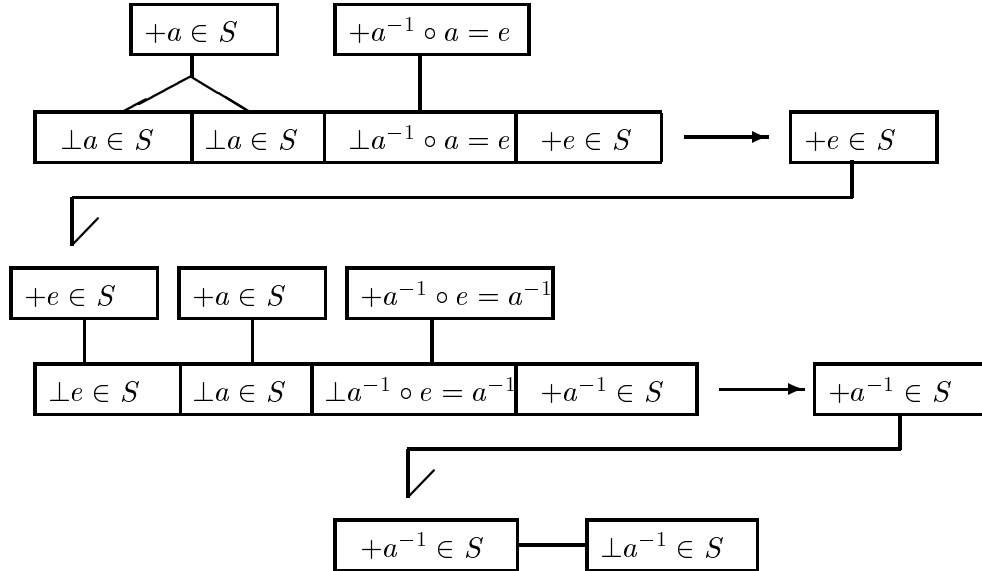


It is left to the reader to verify that this transformation problem satisfies all conditions of Theorem 6.1.2. We start with the application of the base algorithm. As first we apply the

initialization-step and obtain in the ND-proof:

$L_6.$	$L_6$	$\vdash a \in S$	$(Hyp)$
$L_1.$	$L_1$	$\vdash \forall u.(u \circ e = u)$	$(Hyp)$
$L_8.$	$L_1$	$\vdash a^{-1} \circ e = a^{-1}$	$(\forall E L_1)$
$L_2.$	$L_2$	$\vdash \forall w.(w^{-1} \circ w = e)$	$(Hyp)$
$L_9.$	$L_2$	$\vdash a^{-1} \circ a = e$	$(\forall E L_2)$
$L_3.$	$L_3$	$\vdash \forall x, y, z.((x \in S \wedge (y \in S \wedge y^{-1} \circ x = z)) \Rightarrow (z \in S))$	$(Hyp)$
$L_{10}.$	$L_{10}$	$\vdash \neg a^{-1} \in S$	$(Hyp)$
$L_{11}.$	$L_{10}, L_6, L_1, L_2, L_3$	$\vdash \perp$	$(G'')$
$L_7.$	$L_6, L_1, L_2, L_3$	$\vdash a^{-1} \in S$	$(IP_2 L_{11})$
$L_5.$	$L_1, L_2, L_3$	$\vdash a \in S \Rightarrow a^{-1} \in S$	$(\Rightarrow I L_6 L_7)$
$L_4.$	$L_1, L_2, L_3$	$\vdash \forall v.(v \in S \Rightarrow v^{-1} \in S)$	$(\forall I L_5)$

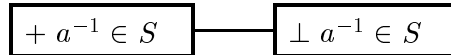
From the UCS-decomposition Algorithm 5.1.9 we obtain the following UCS-decomposition for  $G''$ :



During the UCS-transformation-step we translate these two UCSs into corresponding assertion applications in the ND-proof. Thereby we obtain the following two assertion level lines:

$L_{12}.$	$L_6, L_1, L_2, L_3$	$\vdash e \in S$	$(L_3 L_6 L_6 L_9)$
$L_{13}.$	$L_6, L_1, L_2, L_3$	$\vdash a^{-1} \in S$	$(L_3 L_{12} L_6 L_8)$

Then we transform the remaining end-step



into an justification for  $L_{11}$  using the ND-rule *Contradiction*. Altogether we obtain the following closed ND-proof:

$L_6.$	$L_6$	$\vdash a \in S$	$(Hyp)$
$L_1.$	$L_1$	$\vdash \forall u.(u \circ e = u)$	$(Hyp)$
$L_8.$	$L_1$	$\vdash a^{-1} \circ e = a^{-1}$	$(\forall E L_1)$
$L_2.$	$L_2$	$\vdash \forall w.(w^{-1} \circ w = e)$	$(Hyp)$
$L_9.$	$L_2$	$\vdash a^{-1} \circ a = e$	$(\forall E L_2)$
$L_3.$	$L_3$	$\vdash \forall x, y, z.((x \in S \wedge (y \in S \wedge y^{-1} \circ x = z)) \Rightarrow (z \in S))$	$(Hyp)$
$L_{10}.$	$L_{10}$	$\vdash \neg a^{-1} \in S$	$(Hyp)$
$L_{12}.$	$L_6, L_1, L_2, L_3$	$\vdash e \in S$	$(L_3 L_6 L_6 L_9)$
$L_{13}.$	$L_6, L_1, L_2, L_3$	$\vdash a^{-1} \in S$	$(L_3 L_{12} L_6 L_8)$
$L_{11}.$	$L_{10}, L_6, L_1, L_2, L_3$	$\vdash \perp$	$(Contradiction L_{10} L_{13})$
$L_7.$	$L_6, L_1, L_2, L_3$	$\vdash a^{-1} \in S$	$(IP_2 L_{11})$
$L_5.$	$L_1, L_2, L_3$	$\vdash a \in S \Rightarrow a^{-1} \in S$	$(\Rightarrow I L_6 L_7)$
$L_4.$	$L_1, L_2, L_3$	$\vdash \forall v.(v \in S \Rightarrow v^{-1} \in S)$	$(\forall I L_5)$

■

The base algorithm creates – by construction as described in Algorithm 6.1.1 – always indirect proof parts if it is applied on a UCS base case. But this is not always necessary, for instance, we could obviously construct also a direct ND-proof for the standard example in Example 6.1.3. We did not use the introduced hypothesis  $L_{10}$  except in the transformation of the end step. Hence, we could also derive  $L_7$  directly from  $L_{13}$  without an indirect part. We describe in Section ?? a refined version of the base algorithm and further techniques to avoid indirect proof parts.

Note that the literal base cases (see Section 3.1) satisfy the conditions of Theorem 6.1.2. In fact, a literal base case is the 'smallest' (with respect to the complexity of the refutation graph) possible UCS base case

The content of this section corresponds to similar results on resolution proofs in [Hua96a]. [Hua96a] describes structures in resolution proofs, so-called SSPU-resolutions, that correspond to UCS-decomposable refutation graphs. Hence, the base algorithm for UCS-decomposable graphs corresponds directly to the basic procedure to handle SSPU-resolutions in [Hua96a]. But [Hua96a] contains no algorithm to reduce the general case to a set of SSPU-resolutions. Hence, it is not able to state a complete algorithm based on transformations of SSPU-resolutions. In contrast thereto, we describe in the next two sections how we can apply transformation rules in a goal-directed way to (1) delete each violation of the graph conditions (see in the next section) and (2) to reduce arbitrary refutation graphs to UCS-decomposable refutation graphs (next but one section). Then we can reduce an arbitrary transformation problem into a transformation problem consisting of a set of UCS base cases that we can transform with the base algorithm for UCS-decomposable graphs 6.1.1. Finally, in the last section of this chapter we state the complete UCS-based transformation algorithm, that is composed of the base algorithm for UCS-decomposable graphs and the algorithms we develop in the next two sections.

## 6.2 Satisfying the Graph Conditions

We described in the last section the so-called UCS base case. Thereby, a UCS base case consists of a refutation graph and ND-lines connected with the clauses of the refutation graph by a  $\Delta$ -relation. To be a UCS base case a refutation graph and the ND-lines have to satisfy the following conditions (also stated in Theorem 6.1.2): First, the refutation graph has to be UCS-decomposable. Secondly, the clauses and the ND-lines have to satisfy the graph conditions  $GC_1$  and  $GC_2$ . Whereas the UCS-decomposability of the refutation graph is a property of the graph only,  $GC_1$  and  $GC_2$  concern the refutation graph, the delta-relation, and the ND-lines. We present in this section an algorithm that takes as input an arbitrary transformation problem and produces as output a transformation problem in that each refutation graph and the corresponding ND-lines satisfy the graph conditions  $GC_1$  and  $GC_2$ . How to obtain UCS-decomposable graphs we describe in the next section.

We observed already in the last section, that the literal base case from Section 3.1 is also a UCS base case. A naive approach to satisfy  $GC_1$  and  $GC_2$  (and to obtain UCS-decomposable graphs) would be to apply transformation rules until we would obtain accidentally UCS base cases. This approach would be correct and complete since in the worst case we would apply as many transformation rules as in the literal transformation-algorithm and would obtain a set of literal bases cases. There are indeed problems where we have to decompose until literal base cases are reached. Moreover, it is an important argument in the completeness proofs of this chapter that we can reach in the worst case literal base cases. However, we want to avoid the application of many transformation rules as needed to reach literal base because of the following reasons:

- By each application of a transformation rule there is the danger to make 'bad' decisions about which transformation rule to apply (see Section 3.5). In contrast thereto, during the transformation of the UCS base cases there is no danger to produce 'bad' results.
- Each application of a transformation rule produces new basic ND-lines. Thus, if we apply many transformation rules we obtain long and uncomprehensible ND-proofs. One assertion application step replaces a sequence of basic ND-steps (see Chapter 4). Therefore, we should try to obtain as complex UCS base cases with as many UCSs as possible.

Hence, we describe in this section and the next section algorithms applying as less transformation rules as possible to reach UCS base cases that are as complex as possible.

We present now an algorithm that chooses goal-directed transformation rules to delete violations of the graph conditions  $GC_1$  and  $GC_2$ . The idea is to decompose only such clauses and ND-lines that do not satisfy  $GC_1$  and  $GC_2$ .

### Algorithm 6.2.1 (GC-Algorithm).

Let  $G_1, \dots, G_n$  be refutation graphs justifying open ND-lines  $L_1, \dots, L_n$  in a not closed ND-proof  $\mathcal{N}$ .

**Initialization:** Compute the set  $S_C$  of all clauses in  $G_1, \dots, G_n$  such that the clauses and the ND-lines connected with them do not satisfy  $GC_1$  or  $GC_2$ .

**Decomposition:** Repeat until  $S_C$  is empty:

1. Compute the set  $S_L$  of all ND-lines connected with a clause in  $S_C$ .



2. Choose in  $S_L$  a ND-line  $L$  such that the corresponding transformation rule is applicable.
3. Apply the rule on  $L$ .
4. Let  $C_1, \dots, C_n$  be the clauses connected with  $L$  and let  $C'_1, \dots, C'_m$  be the clauses resulting from these clauses after the application of the transformation rule (it can happen that some clauses are not changed, then it holds that  $C'_i = C_j$  for a pair  $i, j$ ,  $1 \leq i \leq n, 1 \leq j \leq m$ ).  
Remove  $C_1, \dots, C_n$  from  $S_C$  and add each of the clauses  $C'_1, \dots, C'_m$  for that holds that the clause and the ND-line connected with it does not satisfy  $GC_1$  or  $GC_2$ .

■

If an application of this algorithm terminates with  $S_C = \emptyset$  then we obtain because of the construction of the algorithm as output a transformation problem in that each clause and corresponding ND-lines satisfy  $GC_1$  and  $GC_2$ . But does this algorithm always terminate with  $S_C = \emptyset$ ? The critical point is the  $\gamma$ -restriction forbidding some applications of transformation rules (see Section 3.3). Can it happen that we reach during the application of this algorithm a situation such that all ND-lines in  $S_L$  are blocked by the  $\gamma$ -restriction? We show in the following theorem stating the correctness and the completeness of the algorithm that this can not happen.

**Theorem 6.2.2 (Correctness and Completeness of the GC-Algorithm).**

*Let  $(\mathcal{N}, \{G_1, \dots, G_n\}, \Delta)$  be a transformation problem. If we apply the GC-algorithm on this transformation problem then holds:*

**Termination:** *The application of the algorithm terminates with  $S_C = \emptyset$ .*

**Correctness:** *The result is a transformation problem  $(\mathcal{N}', \{G'_1, \dots, G'_m\}, \Delta')$  such that all clauses of the refutation graphs  $\{G'_1, \dots, G'_m\}$  and the ND-lines of  $\mathcal{N}'$  connected with these clauses by  $\Delta'$  satisfy the graph conditions  $GC_1$  and  $GC_2$ .*

**Proof:** During the application of the algorithm  $S_L$  and  $S_C$  consist always of exactly the clauses and ND-lines that do not satisfy  $GC_1$  or  $GC_2$  (\*).

We omit to give a formal proof of this statement (e.g., by induction over the number  $n$  of applied transformation rules) since it obviously follows by the construction of the algorithm.

**Termination:** If the algorithm is not blocked – if all ND-lines in  $S_L$  are blocked by the  $\gamma$ -restriction – then it terminates because in the worst case we have to decompose until literal base cases are reached (the completeness of the literal-transformation algorithm guarantees this). Since literal base cases satisfy  $GC_1$  and  $GC_2$   $S_C$  is then empty.

We have to prove now that it cannot happen that the decomposition of all lines in  $S_L$  is blocked by the  $\gamma$ -restriction.

The  $\gamma$ -restriction forbids to instantiate gamma quantified variables by skolem terms. Each skolem function in the ground substitutions of the clauses of the refutation graphs was created by the application of the  $\delta$ -rule during the clause normalization of a ND-line connected with some clauses in the current refutation graphs.

$S_L$  contains – among others – all ND-lines connected with clauses in whose clause normalization the  $\delta$ -rule is applied (because of  $GC_2$ ,  $GC_1$  and (\*)).



But then we obtain by a proof similar to the proof of Lemma 3.3.2 that there has to be at least one ND-line in  $S_L$  not blocked by the  $\gamma$ -restriction.

**Correctness:** All we do in this algorithm is applying transformation rules. Hence, we reach again a transformation problem and produce correctly justified ND-lines. That in this transformation problem there are no further violations of  $GC_1$  and  $GC_2$  follows from the fact that the algorithm terminates with  $S_L = S_C = \emptyset$  (see the first part of this proof) and  $(*)$ .

□

*Example 6.2.3 (Standard Example (Continuation)).*

We apply the GC-algorithm on the initial transformation problem of the standard example (we presented the initial refutation graph in Figure 2.1, the initial ND-proof in Example 2.5.2), and the initial  $\Delta$ -relation in Example 2.5.3).

In this initial transformation problem we have the following violations against  $GC_1$  and  $GC_2$ :

The formula of line  $L_1$  is not literal but it is connected with the unit clause  $[+u_1 \circ e = u_1]$ . The formula of line  $L_2$  is not literal but it is connected with the unit clause  $[+w_1^{-1} \circ w = e]$ . The formula of line  $L_4$  is not literal but it is connected with the unit clauses  $[+sk_1 \in S]$ ,  $[\perp sk_1^{-1} \in S]$ .

We have here three violations against  $GC_1$  and zero violations against  $GC_2$ . Hence, our initial  $S_C$  is:

$$S_C = \{[+u_1 \circ e = u_1], [+w_1^{-1} \circ w = e], [+sk_1 \in S], [\perp sk_1^{-1} \in S]\},$$

the corresponding  $S_L$  is:

$$S_L = \{L_1, L_2, L_4\}.$$

The appropriate transformation rule for  $L_1, L_2$  would be  $TI\forall$ , but since the corresponding ground substitutions contain skolem terms ( $sk_1, sk_1^{-1}$ ), respectively, we have to decompose first  $L_4$ . We do this by applying the transformation rule  $TE\forall$  on  $L_4$ . The resulting ND-proof is:

$L_1.$	$L_1$	$\vdash \forall u. (u \circ e = u)$	(Hyp)
$L_2.$	$L_2$	$\vdash \forall w. (w^{-1} \circ w = e)$	(Hyp)
$L_3.$	$L_3$	$\vdash \forall x, y, z. ((x \in S \wedge (y \in S \wedge y^{-1} \circ x = z)) \Rightarrow (z \in S))$	(Hyp)
$L_5.$	$L_1, L_2, L_3$	$\vdash a \in S \Rightarrow a^{-1} \in S$	(G')
$L_4.$	$L_1, L_2, L_3$	$\vdash \forall v. (v \in S \Rightarrow v^{-1} \in S)$	( $\forall I$ $L_5$ )

$G'$  results from the original refutation graph by replacing each occurrence of  $sk_1$  both in the clauses and the ground substitutions by an occurrence of the new constant  $a$ .

The clauses  $[+a \in S]$  and  $[\perp a^{-1} \in S]$  are now connected with  $L_5$  but the clauses and the ND-line still violate  $GC_1$ . Hence, we obtain the following  $S_C$  and  $S_L$ :

$$S_C = \{[+u_1 \circ e = u_1], [+w_1^{-1} \circ w = e], [+a \in S], [\perp a \in S]\},$$

$$S_L = \{L_1, L_2, L_5\}.$$

Now the  $\gamma$ -restriction blocks no application of a transformation rule. So we can choose between applying  $TI\forall$  on  $L_1$  or  $L_2$  or applying  $TE\Rightarrow$  on  $L_5$ . We decide for applying  $TE\Rightarrow$  on  $L_5$ . The resulting ND-proof is:

**Draft**  
9/2/2000

$L_6.$	$L_6$	$\vdash a \in S$	$(Hyp)$
$L_1.$	$L_1$	$\vdash \forall u.(u \circ e = u)$	$(Hyp)$
$L_2.$	$L_2$	$\vdash \forall w.(w^{-1} \circ w = e)$	$(Hyp)$
$L_3.$	$L_3$	$\vdash \forall x, y, z.((x \in S \wedge (y \in S \wedge y^{-1} \circ x = z)) \Rightarrow (z \in S))$	$(Hyp)$
$L_7.$	$L_6, L_1, L_2, L_3$	$\vdash a^{-1} \in S$	$(G')$
$L_5.$	$L_1, L_2, L_3$	$\vdash a \in S \Rightarrow a^{-1} \in S$	$(\Rightarrow I \ L_6 \ L_7)$
$L_4.$	$L_1, L_2, L_3$	$\vdash \forall v.(v \in S \Rightarrow v^{-1} \in S)$	$(\forall I \ L_5)$

After this step only  $L_1$  and  $L_2$  are connected with clauses such that they violate  $GC_1$ , thus we obtain the following  $S_C$  and  $S_L$ :

$$S_C = \{[+u_1 \circ e = u_1], [+w_1^{-1} \circ w = e]\},$$

$$S_L = \{L_1, L_2\}.$$

Applying then (successively)  $TI\forall$  on  $L_1$  and  $L_2$  we obtain the following ND-proof:

$L_6.$	$L_6$	$\vdash a \in S$	$(Hyp)$
$L_1.$	$L_1$	$\vdash \forall u.(u \circ e = u)$	$(Hyp)$
$L_8.$	$L_1$	$\vdash a^{-1} \circ e = a^{-1}$	$(\forall E \ L_1)$
$L_2.$	$L_2$	$\vdash \forall w.(w^{-1} \circ w = e)$	$(Hyp)$
$L_9.$	$L_2$	$\vdash a^{-1} \circ a = e$	$(\forall E \ L_2)$
$L_3.$	$L_3$	$\vdash \forall x, y, z.((x \in S \wedge (y \in S \wedge y^{-1} \circ x = z)) \Rightarrow (z \in S))$	$(Hyp)$
$L_7.$	$L_6, L_1, L_2, L_3$	$\vdash a^{-1} \in S$	$(G'')$
$L_5.$	$L_1, L_2, L_3$	$\vdash a \in S \Rightarrow a^{-1} \in S$	$(\Rightarrow I \ L_6 \ L_7)$
$L_4.$	$L_1, L_2, L_3$	$\vdash \forall v.(v \in S \Rightarrow v^{-1} \in S)$	$(\forall I \ L_5)$

$G''$  results from  $G'$  by applying the ground substitutions on the clauses  $[+u_1 \circ e = u_1]$  and  $[+w_1^{-1} \circ w = e]$ , respectively, such that we obtain the clauses  $[+a^{-1} \circ e = a^{-1}]$  and  $[+a^{-1} \circ a = e]$  instead.

Then all clauses and the ND-lines connected with them satisfy  $GC_1$  and  $GC_2$  and the GC-algorithm terminates since  $S_C$  is empty.  $\blacksquare$

Note that the transformation problem produced in this example as output of the GC-algorithm, is exactly the transformation problem that we used in Example 6.1.3 as input for the base algorithm for UCS-decomposable graphs. Thus, the GC-algorithm and the base algorithm for UCS-decomposable graphs compute together a closed ND-proof at assertion level for the standard example. Thereby, they use together less applications of transformation rules than the application of the literal transformation algorithm (compare with Example 3.4.3).

We described in this section an algorithm applying goal-directed transformation rules only on ND-lines and clauses violating  $GC_1$  and  $GC_2$  until all such violations are deleted. The result is a transformation problem containing no violations of  $GC_1$  and  $GC_2$ . In the next section we describe how we can obtain additionally UCS-decomposable graphs also by applying less transformation rules in a goal-directed manner.

**Draft**  
9/2/2006

### 6.3 Obtaining UCS-decomposable Graphs

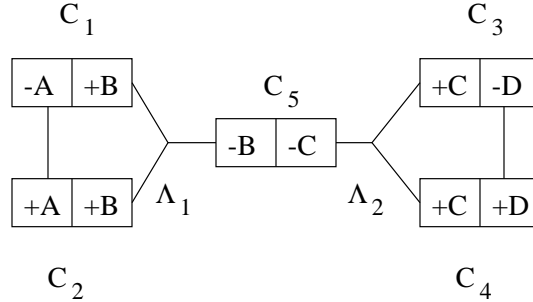
In this section we regard the following case: We assume that we applied the GC-algorithm and obtained a transformation problem in that  $GC_1$  and  $GC_2$  are satisfied. But we cannot apply the base algorithm for UCS-decomposable graphs, since we have not UCS-decomposable refutation graphs (also a premise of Theorem 6.1.2). We have here the same problem as in the last section. A naive approach to obtain UCS-decomposable graphs would be to apply arbitrarily applicable transformation rules, until we would obtain accidentally UCS base cases. This approach would be correct and complete since in the worst case we would apply transformation rules until we would have literal base cases which are also UCS-base cases. But this is not what we want (see also the introduction of the last section). Hence, we present in this section methods how to obtain UCS-decomposable graphs in a goal-directed way such that we have to apply as less transformation rules as possible. Regard the following example:

*Example 6.3.1 (Not UCS-decomposable Graph).*

We have the following initial ND-proof  $\mathcal{N}$ :

$L_1.$	$L_1$	$\vdash A \Rightarrow B$	$(Hyp)$
$L_2.$	$L_2$	$\vdash \neg A \Rightarrow B$	$(Hyp)$
$L_3.$	$L_3$	$\vdash D \Rightarrow C$	$(Hyp)$
$L_4.$	$L_4$	$\vdash \neg D \Rightarrow C$	$(Hyp)$
$L_5.$	$L_1, L_2, L_3, L_4$	$\vdash B \wedge C$	$(G)$

where  $G$  is the following refutation graph:



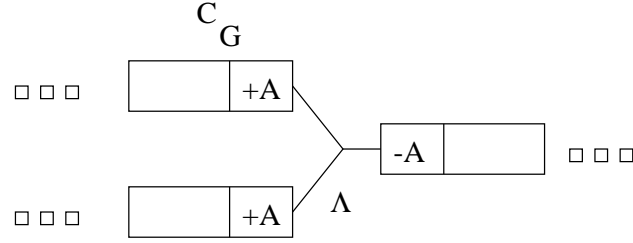
and  $\Delta$  is the corresponding  $\Delta$ -relation between  $G$  and  $\mathcal{N}$ . This transformation problem satisfies  $GC_1$  and  $GC_2$  but  $G$  is not UCS-decomposable, since it contains with  $\Lambda_1$  and  $\Lambda_2$  contra-directed links. ■

The results of Chapter 5 provide us with the knowledge to reach UCS-decomposable graphs in a goal-directed manner. By Theorem 5.6.1 we know that a refutation graph is UCS-decomposable if and only if it contains no contra-directed links. Hence, to obtain UCS-decomposable graphs we can apply goal-directed such decompositions breaking or separating contra-directed links. In the following we present four strategies to do this: the *liquidation-strategy with the direct-decomposition-method*, the *separation-strategy with the direct-decomposition-method*, the *liquidation-strategy with the symmetrical-simplification-method*, and the *separation-strategy with the symmetrical-simplification-method*. The idea of

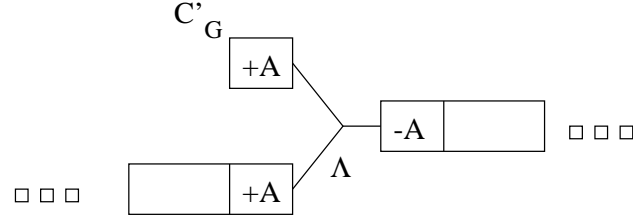


the *direct-decomposition-method* is to apply transformation rules on ND-lines and clauses that are in the current transformation problem, whereas the symmetrical-simplification method first introduces a new ND-line and a new clause and applies then transformation rules on the new ND-line and clause.

**Liquidation-Strategy with the Direct-Decomposition-Method:** If we have a refutation graph which is not UCS-decomposable then it contains pairs of contra-directed multi-links. The liquidation-strategy liquidates one of these multi-links. Assume we have the following situation:

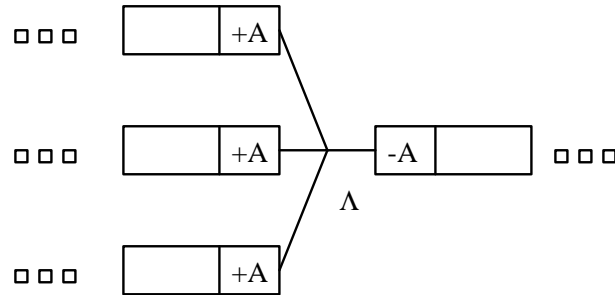


where  $C_G$  is one of the clauses of the G-part of link  $\Lambda$ . If we apply transformation rules on the line connected with  $C_G$ ,  $C_G$  becomes decomposed little-by-little. We can repeat decomposing the clauses resulting from  $C_G$  and the ND-lines connected with them, until we obtain a unit clause  $C'_G$  for  $C_G$ .



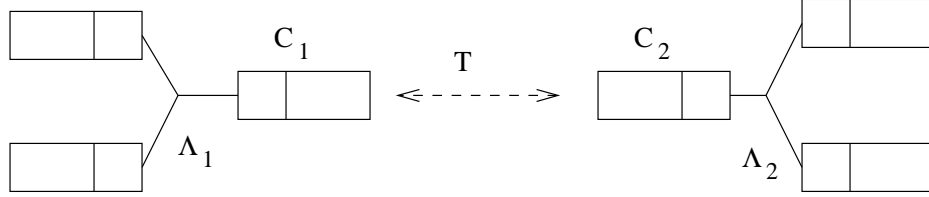
But this situation is not possible since this refutation graph is not minimal whereas we guarantee to obtain always minimal refutation graphs by decompositions with transformation rules (invariant 1). Therefore,  $\Lambda$  as multi-link becomes liquidated during the decompositions on  $G$  and its successors.

If we have multi-links with more than two clauses in the G-part, we have to decompose all clauses in the G-part except one to have this effect.

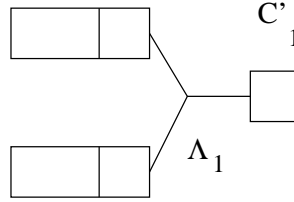


**Separation Strategy with the Direct-Decomposition-Method:** If we have a refutation graph which is not UCS-decomposable then it contains pairs of contra-directed multi-links. Two contra-directed multi-links are connected by a trail between their L-clauses. The separation strategy breaks this trail such that the previously contra-directed links are afterwards separated.

Assume we have the following situation:



Here  $\Lambda_1$  and  $\Lambda_2$  are two contra-directed links connected by a trail  $T$  between their L-clauses. By Lemma 5.5.6 holds that  $\Lambda_1$  and  $\Lambda_2$  are contra-directed if and only if there is a trail between their L-clauses  $C_1$  and  $C_2$  without using  $\Lambda_1$  and  $\Lambda_2$ . We apply on  $C_1$  (or  $C_2$ ) transformation rules until we obtain a unit clause  $C'_1$  for  $C_1$ .



Then in the resulting situation there can be obviously no trail between the L-clause  $C'_1$  of  $\Lambda_1$  and any other clause without using  $\Lambda_1$ . The trail  $T$  is broken and the links are separated.

Note that the separation strategy with the direct-decomposition-method is not applicable on double multi-links, since – regarding the double multi-link as two simple multi-links (see Section 5.4) – there is no clause on a trail between them. Furthermore, it is maybe not enough to break an arbitrary clause on the trail between the L-clauses of two contra-directed links, since then it can happen that there is still another trail. Therefore, we decompose in the separation strategy with the direct-decomposition-method one of the L-clauses. Another remark holds for both strategies presented so far: It is not necessary to decompose always until the decomposed clause is really a unit clause. This is only the worst case. Often we obtain already after less decompositions that the contra-directed links become liquidated or separated. We can stop as soon as this happens.

We demonstrate now the use of these two strategies on the situation of Example 6.3.1.

**Example 6.3.2. Liquidation-Strategy with the Direct-Decomposition-Method**  $\Lambda_1$  and  $\Lambda_2$  are contra-directed; hence, we try to liquidate  $\Lambda_1$ . The G-clauses of  $\Lambda_1$  are  $C_1$  and  $C_2$ . We want to decompose  $C_1$ ; hence, we apply on the ND-line  $L_1$  connected with  $C_1$  the transformation rule  $TI \Rightarrow$ . We obtain a new ND-line  $L_6$ .

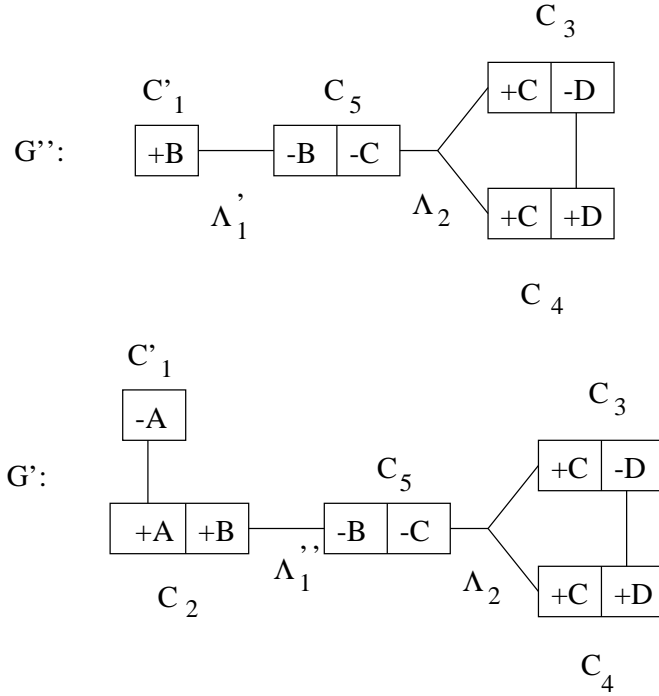


$L_1.$	$L_1$	$\vdash A \Rightarrow B$	$(Hyp)$
$L_6.$	$L_1$	$\vdash \neg A \vee B$	$(Taut L_1)$

$C_1$  is now connected with  $L_6$  but the refutation graph  $G$  did not change. Therefore, we apply the transformation rule *TMCases* on  $L_6$ . We obtain the following ND-proof:

$L_1.$	$L_1$	$\vdash A \Rightarrow B$	$(Hyp)$
$L_2.$	$L_2$	$\vdash \neg A \Rightarrow B$	$(Hyp)$
$L_3.$	$L_3$	$\vdash D \Rightarrow C$	$(Hyp)$
$L_4.$	$L_4$	$\vdash \neg D \Rightarrow C$	$(Hyp)$
$L_6.$	$L_1$	$\vdash \neg A \vee B$	$(Taut L_1)$
$L_7.$	$L_7$	$\vdash \neg A$	$(Hyp)$
$L_8.$	$L_1, L_2, L_3, L_4, L_7$	$\vdash B \wedge C$	$(G')$
$L_9.$	$L_9$	$\vdash B$	$(Hyp)$
$L_{10}.$	$L_1, L_2, L_3, L_4, L_9$	$\vdash B \wedge C$	$(G'')$
$L_5.$	$L_1, L_2, L_3, L_4$	$\vdash B \wedge C$	$(Cases L_6 L_7 L_9)$

with the refutation graphs  $G'$  and  $G''$



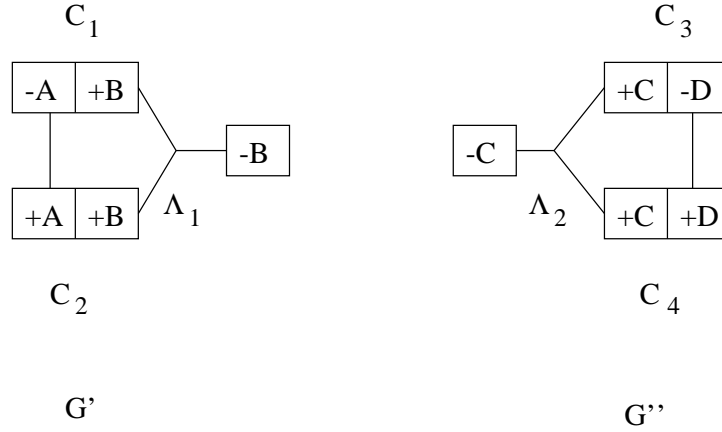
We liquidated  $\Lambda_1$  as multi-link, since in the resulting refutation graphs  $G'$  and  $G''$  the links  $\Lambda_1'$  and  $\Lambda_1''$  corresponding to  $\Lambda_1$  are binary links. Thus,  $G'$  and  $G''$  are both contra-links free and UCS-decomposable.<sup>1</sup>

<sup>1</sup>Notice that during the decomposition of the original graph the whole subgraph connected with the negative shore of  $\Lambda_1$  became duplicated. Would we apply now on  $G'$  and  $G''$  the base algorithm for UCS-decomposable graphs, then we would have to work on this part twice. This would not only be inefficient but it would also increase the ND-proof. We shall see in the next chapter in Section ?? how we can avoid such duplications.

**Separation-Strategy with the Direct-Decomposition-Method** We decompose the clause  $C_5$  which is the L-clause of both  $\Lambda_1$  and  $\Lambda_2$ . Since  $C_5$  is connected with line  $L_5$  we apply the appropriate transformation rule  $TE\wedge$  on  $L_5$  and obtain the following ND-proof:

$L_1.$	$L_1$	$\vdash A \Rightarrow B$	$(Hyp)$
$L_2.$	$L_2$	$\vdash \neg A \Rightarrow B$	$(Hyp)$
$L_3.$	$L_3$	$\vdash D \Rightarrow C$	$(Hyp)$
$L_4.$	$L_4$	$\vdash \neg D \Rightarrow C$	$(Hyp)$
$L_6.$	$L_1, L_2, L_3, L_4$	$\vdash B$	$(G')$
$L_7.$	$L_1, L_2, L_3, L_4$	$\vdash C$	$(G'')$
$L_5.$	$L_1, L_2, L_3, L_4$	$\vdash B \wedge C$	$(\wedge I \ L_6 \ L_7)$

with the refutation graphs  $G'$  and  $G''$



Here  $\Lambda_1$  and  $\Lambda_2$  are separated in different graphs. Hence,  $G'$  and  $G''$  are both contra-links free and UCS-decomposable. ■

Both strategies of the direct-decomposition-method work with the decomposition of given clauses and ND-lines. A problem thereof is that the formula of an ND-line can be very complex, such that we have to apply a lot of transformation rules to obtain the wished decomposition of the chosen clause. An approach to avoid this problem is the so-called *symmetrical-simplification-method*. The base idea is taken from [PN90]. There, the symmetrical simplification is used during the transformation of machine-found proofs into ND-proofs to avoid the production of indirect proof parts and to introduce lemmas. We use this idea to obtain UCS-decomposable graphs. We allow the introduction of additional ND-lines justified by  $TND$  and with formulas  $A \vee \neg A$

$$L. \quad \vdash A \vee \neg A \quad (TND)$$

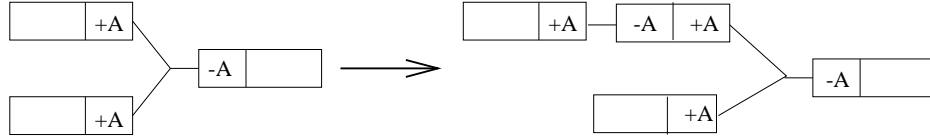
where  $A$  is a closed, atom formula that contains no skolem terms. If this formula is clause normalized we obtain a clause  $[+A, \perp A]$ . In the following we explain how we can insert this

clause into the refutation graph and can then split it by applying exactly *one* transformation rule on its corresponding ND-line  $L$ .

**Liquidation-Strategy with the Symmetrical-Simplification-Method:** We assume that we are in the same situation as in the liquidation-strategy with the direct-decomposition-method. We add a new line  $L$  to the ND-proof:

$$L. \quad \vdash A \vee \neg A \quad (TND)$$

where  $A$  is the atom of the literal of the L-clause which is connected with the multi-link. In the refutation graph we add the corresponding clause  $[+A, \perp A]$  between the link  $\Lambda$  and the G-clause of  $\Lambda$ .

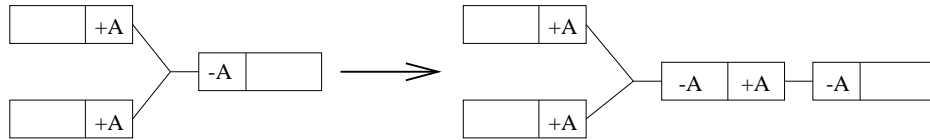


If we decompose now this clause by applying a transformation rule on  $L$  we have the same effect as with the liquidation-strategy with the direct-decomposition-method, but we can guarantee, that we break the clause  $[+A, \perp A]$  in the wished manner with exactly *one* application of a transformation rule.

**Separation-Strategy with the Symmetrical-Simplification-method:** We assume that we are in the same situation as in the separation-strategy with the direct-decomposition-method. We add the new line  $L$  to the ND-proof:

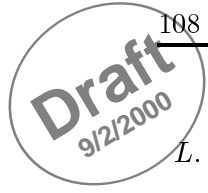
$$L. \quad \vdash A \vee \neg A \quad (TND)$$

where  $A$  is the atom of that literal of the L-clause of  $\Lambda_1$  (or similarly  $\Lambda_2$ ) connected by  $\Lambda_1$ . We add the new clause  $[+A, \perp A]$  between the L-clause and  $\Lambda_1$ .



Afterwards, we can decompose this new clause and the line  $L$  with the application of *one* transformation rule. We obtain the same effect as with the separation-strategy with the direct-decomposition-method: the contra-directed links become separated.

The separation-strategy with the symmetrical-simplification-method is also applicable if the contra-directed link is a double multi-link. In this case we add the new clause into the double multi-link such that the two shores of the link are separated by the new clause. If we choose to introduce a new clause between literals that are not ground, we have to take not the atom of one of the literals but the ground substituted atom to construct the new ND-line:

 $L.$ 

$$\vdash (A\sigma) \vee \neg(A\sigma)$$

 $(TND)$ 

Hence, there is the following general restriction for the symmetrical-simplification-method: We can apply it only if the ground substitution  $\sigma$  contains no skolem terms. If we assume that our transformation problem satisfies  $GC_1$  and  $GC_2$  then there can be no further skolem terms in the refutation graphs (compare with the proof of Theorem 6.2.2).

We apply now both liquidation-strategy and separation-strategy with the symmetrical-simplification-method on Example 6.3.1.

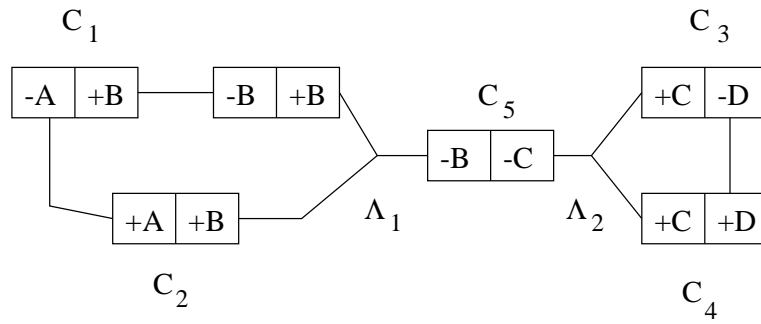
**Example 6.3.3. Liquidation-Strategy with the Direct-Decomposition-Method** We add the new ND-line  $L_6$

 $L_6.$ 

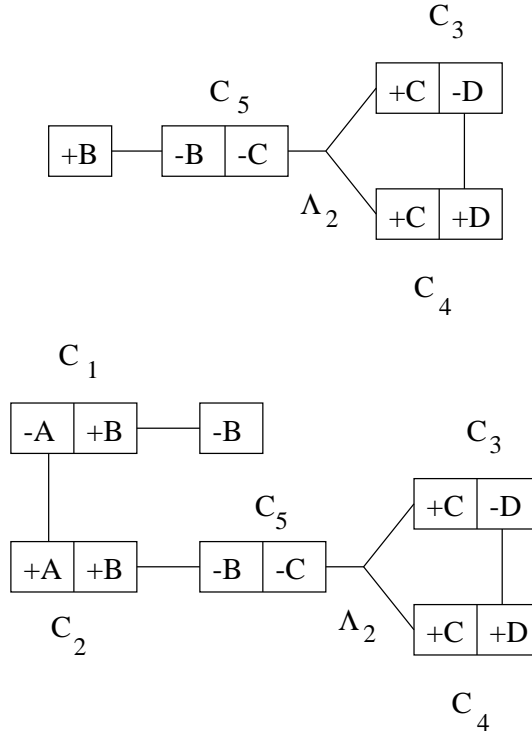
$$\vdash B \vee \neg B$$

 $(TND)$ 

Furthermore, we add the corresponding clause  $[+B, \perp B]$  into the refutation graph



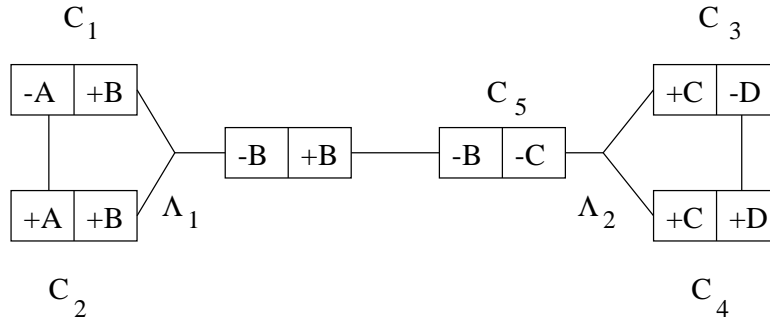
We apply *TMCases* on  $L_6$  and obtain (with one decomposition step) the following UCS-decomposable graphs:



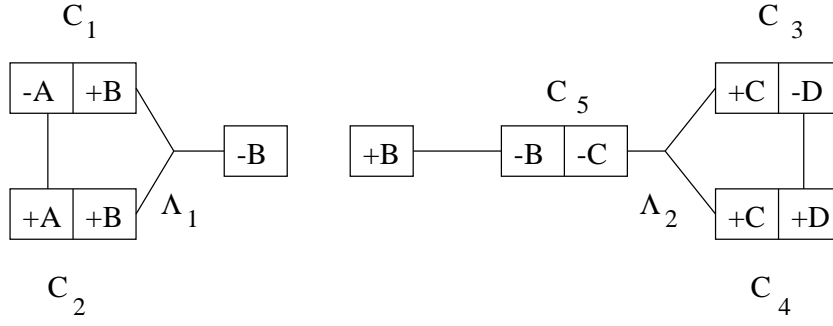
**Separation-Strategy with the Direct-Decomposition-Method** We add the new ND-line  $L_6$  with

$$L_6. \quad \vdash B \vee \neg B \quad (TND)$$

Furthermore, we add the corresponding clause  $[+B, \perp B]$  into the refutation graphs



We break this new clause  $[+B, \perp B]$  by an application of *TMCases* on  $L_6$ . Thus, we obtain after *one* application of a transformation rule the following UCS-decomposable graphs:



■

We presented four strategies to delete contra-directed links. The question is now which strategy should be preferred in which situation? We found the following heuristics:

**Prefer symmetrical-simplification-method before direct-decomposition-method:** In general, we suggest to prefer the symmetrical-simplification-method before the direct-decomposition-method. First, the symmetrical-simplification-method guarantees that the wished breaking of a clause is done with exactly *one* application of a transformation rule. Secondly, it guarantees that no clauses and UCSs are destroyed unnecessarily since the effects of these method are restricted on the clause that should be broken. In contrast thereto, the application of transformation rules during the direct-decomposition-method can effect also other clauses not mentioned in the method but connected with the ND-line on that we call the transformation rule.

**Prefer separation-strategy before liquidation-strategy:** If there are multi-links with more than two G-clauses we suggest to prefer the separation-strategy before the liquidation-strategy. The separation-strategy guarantees that we have to break only one clause, whereas we have to break in the liquidation-strategy (in the worst case)  $n + 1$  clauses (where  $n$  is the number of clauses in the G-part of the link).

**Prefer small clause and lines during the direct-decomposition-method:** During an application of the direct-decomposition-method we suggest to choose such clauses and corresponding ND-lines, that the breaking of the clause can be achieved by less applications of transformation rules. As approximative measure we suggest the complexity of the clause or the ND-line.

**Prefer links which are multiple contra-directed:** In a graph containing several pairs of contra-directed multi-links we suggest to choose first such links for separation or liquidation that are contained in the most pairs. Then we can remove many pairs of contra-directed links by one application of a liquidation- or separation-method.

We conclude this section with two final remarks about our strategies:

1. The completeness of the strategies introduced in this section with respect to the achievement of UCS-decomposable graphs follows – as usual – from the argument that we can decompose in the worst case until we reach literal base cases which are UCS-decomposable.



2. If we apply these strategies on a refutation graph and corresponding ND-lines that satisfy already the graph conditions  $GC_1$  and  $GC_2$ , then there can be afterwards again violations against  $GC_1$  (since during the applications of transformation rules new unit clauses can arise for that we can not guarantee  $GC_1$ ).

We introduced in this section four strategies allowing to obtain UCS-decomposable graphs by applying only a few transformation rules in a goal-directed way. In the next section we present the complete UCS-based transformation algorithm which uses these strategies.

## 6.4 The UCS-Based Transformation Algorithm

Combining the algorithms introduced in the different sections of this chapter we can construct the following transformation algorithm based on the translation of UCSs.

### Algorithm 6.4.1 (The UCS-Based Transformation Algorithm).

Given an initial transformation problem  $(\mathcal{N}, \{G\}, \Delta)$  where  $\mathcal{N}$  is the ND-proof whose unique open line  $L$  is justified by the minimal refutation graph  $G$ ;  $\Delta$  is the  $\Delta$ -relation connecting the clauses of  $G$  and the ND-lines of  $\mathcal{N}$ .

**GC-Satisfaction-Step:** Apply the GC-Algorithm 6.2.1 on the initial transformation problem. It shall produce a transformation problem  $(\mathcal{N}', \{G'_1, \dots, G'_{n'}\}, \Delta')$  satisfying  $GC_1$  and  $GC_2$ .

**UCS-Decompositions-Step:** Apply the strategies introduced in the last section until all refutation graphs are UCS-decomposable. The resulting transformation problem is  $(\mathcal{N}'', \{G''_1, \dots, G''_{n''}\}, \Delta'')$ .

**Guarantee- $GC_1$ -Step:** If the UCS-decompositions-step produced new unit clauses, apply again the GC-algorithm to guarantee  $GC_1$  (compare with the concluding remark of the last section). Let  $(\mathcal{N}''', \{G'''_1, \dots, G'''_{n'''}\}, \Delta''')$  be the resulting transformation problem.

**Base-Step:**  $(\mathcal{N}''', \{G'''_1, \dots, G'''_{n'''}\}, \Delta''')$  satisfies  $GC_1$  and  $GC_2$  and all refutation graphs  $G'''_1, \dots, G'''_{n'''}$  are UCS-decomposable. Hence, we can apply the base algorithm for UCS-decomposable graphs (Algorithm 6.1.1) to obtain an empty transformation problem with a closed ND-proof.

■

The correctness and completeness of this algorithm follow directly from the correctness and completeness results of its single components described in the last sections. Note the following remark about the order of the steps: We could make first the UCS-decompositions-step and then the GC-satisfaction-step. In this case we would need no extra step to guarantee  $GC_1$  again. We decided for this order of the steps (including an extra step to guarantee  $GC_1$  again) since we found that during the GC-satisfaction-step often not UCS-decomposable graphs become decomposed into UCS-decomposable graphs. In contrast thereto, if we make first the UCS-decompositions-step, in general, this will not help to achieve the satisfaction of  $GC_1$  and  $GC_2$ .

The UCS-based transformation algorithm is based on translating steps at the assertion level from refutation graphs into ND-proofs. Hence, in general, we have not to decompose

first all structures until we reach literal level, to translate then steps at the literal level from the refutation graphs into the ND-proofs, and to abstract then the ND-proofs. In fact, this algorithm produces directly ND-proofs at the assertion level. Since the resulting assertion steps are macro-steps and we have to apply only a few (in comparison to the literal transformation algorithm) transformation rules in a goal-directed way, we obtain short, well structured, and comprehensible ND-proofs that contain rarely indirect proof parts (if we add some extensions to the base algorithm for UCS-decomposable graphs that avoid indirect proofs, see next chapter). Therefore, in general, the results of this transformation algorithm are superior to the combination of the literal transformation algorithm (see Section 3.4 and the subsequent abstraction to assertion level (see Section 4.2). Only in the worst case – if the refutation graphs contain no UCSs representing assertion applications – we have to decompose to the literal level. Then this algorithm suffers on the same problems as the literal transformation algorithm (see Section 3.5). Moreover, this algorithm is not only applicable to obtain ND-proofs at assertion level, but that it can provide us – combined with an expansion of the assertion applications (see Section 4.1) – also with base ND-proofs. Also in this case the UCS-based transformation algorithm produces better results as the literal transformation algorithm, since also then the resulting ND-proofs contain rarely indirect proof parts and are well structured.

A complete example demonstrating the use of the UCS-based transformation algorithm is the transformation of the standard example distributed to Example 6.1.3 and Example 6.2.3 (the UCS-decompositions-step and the Guarantee- $GC_1$ -step are empty since we have in the standard example directly a UCS-decomposable graph). We omit here to present further examples since we want to introduce first in the next chapter some extensions of the UCS-based transformation algorithm. In particular, we describe how we can avoid the creation of indirect proof parts by the base algorithm for UCS-decomposable graphs. Furthermore, we describe how we can present paramodulation steps in the refutation graph and how we can handle such structures in refutation graphs representing paramodulation steps in the UCS-based transformation algorithm. More examples we present in the next but one chapter.



## Chapter 7

# Extensions of the UCS-Based Transformation Algorithm

This chapter will contain some extensions of the UCS-Based Transformation Algorithm:

1. An extension of the base algorithm for UCS-decomposable graphs that avoids the production of indirect proof parts.
2. An extension that avoids duplications when a refutation graph is split.
3. Extensions to present and handle paramodulation steps.





## Chapter 8

# Examples

This chapter will contain the detailed explanation of the handling of further examples by the UCS-based transformation algorithm.



## Appendix A

# The Transformation Rules

In this chapter we present complete set of transformation rules we need and use in our transformation algorithms. Except  $TM \vee left$ ,  $TM \vee right$ , and  $TMContra$  all these transformation rules are taken from [Lin90]. The rules are divided into two sets:

**The Set of Standard Rules** contains for each possible connective and quantifier in ND-lines exactly two rules, one to decompose a closed ND-line and one to decompose an open ND-lines. This set of rules is complete for the transformation algorithms described in this report.

**The Set of Alternative Rules** contains additional rules for certain formulas, which produce shorter and better comprehensible proof parts as the corresponding standard rules (see Section 3.5).

The alternative rules are not always applicable (see also Section 3.5). Thus, their usage is controlled by heuristics. In [Lin90], in particular, application heuristics for  $TMInf$  are examined. We present here a more general heuristic which is also expandable to the rules  $TM \vee right$ ,  $TM \vee left$  and  $TMContra$ .

The names of the rules are also taken from [Lin90] (we named the new rules  $TM \vee left$ ,  $TM \vee right$ , and  $TMContra$  consistently). The first letter of each transformation rule is  $T$  to divide them clearly from the ND-rules. For the next letter there are the following possibilities:

- $I$ , for instance in  $TI\wedge$ , indicates that the rule is an *internal rule*. Internal rules work only on closed (ND-calculus internal) ND-lines.
- $E$ , for instance in  $TE\wedge$ , indicates that the rule is an *external rule*. External rules work only on open (ND-calculus external) ND-lines.
- $M$ , for instance in  $TMInf$ , indicates that the rule is a *mixed rule*. Mixed rules affect simultaneously both closed and open ND-lines.

In the following we present the single transformation rules. We assume always that we have a transformation problem  $(\mathcal{N}, \{G_1, \dots, G_n\}, \Delta)$  with a ND-proof  $\mathcal{N}$ , a set of refutation graphs  $\{G_1, \dots, G_n\}$  justifying the open lines in  $\mathcal{N}$ , and a delta-relation  $\Delta$  connecting the formulas of  $\mathcal{N}$  and the clauses of the refutation graphs. The transformation rules can change simultaneously the ND-proof, some refutation graphs, and the delta-relation. To describe the

changes in the refutation graph we need an algorithm breaking one refutation graph into two refutation graphs by breaking some clauses in the original refutation graph (see Appendix 3.2). This algorithm is described in the next chapter.

## A.1 The Standard Rules

$TI \wedge$ :

$$L_1. \mathcal{A} \quad \vdash F \wedge G \quad (\mathcal{R}) \quad \rightarrow \quad \left\{ \begin{array}{ll} L_1. \mathcal{A} \quad \vdash F \wedge H & (\mathcal{R}) \\ L_2. \mathcal{A} \quad \vdash F & (\wedge E_L L_1) \\ L_3. \mathcal{A} \quad \vdash H & (\wedge E_R L_2) \end{array} \right.$$

We need not to change any refutation graphs.

We change  $\Delta$  such that clauses previously connected with  $L_1$  are afterwards connected with  $L_2$  or  $L_3$ , respectively.

$TM Cases$ :

$$\left. \begin{array}{ll} L_1. \mathcal{A} \quad \vdash F \vee H & (\mathcal{R}) \\ L_2. \mathcal{A} \quad \vdash I & (G) \end{array} \right\} \rightarrow \left\{ \begin{array}{ll} L_1. \mathcal{A} \quad \vdash F \vee H & (\mathcal{R}) \\ L_3. L_3 \vdash F & (Hyp) \\ L_4. \mathcal{A}, L_3 \vdash I & (G') \\ L_5. L_5 \vdash H & (Hyp) \\ L_6. \mathcal{A}, L_5 \vdash I & (G'') \\ L_2. \mathcal{A} \quad \vdash I & (Cases L_1 L_4 L_6) \end{array} \right.$$

$G'$  and  $G''$  result from  $G$  by breaking the clauses connected with  $F \vee H$  into two clauses, respectively, one clause in the CNF of  $F$  and one in the CNF of  $H$ .

We change  $\Delta$  such that the new clauses resulting from the breaking are connected with  $L_3$  or  $L_5$ , respectively, depending on whether they are in the CNF of  $F$  or  $H$ . Furthermore, the clauses previously (in  $G$ ) connected with  $L_2$  are connected afterwards with  $L_4$  (if the clause is in  $G'$ ) or  $L_6$  (if the clause is in  $G''$ ), respectively.

$TI \Rightarrow$ :

$$L_1. \mathcal{A} \quad \vdash F \Rightarrow H \quad (\mathcal{R}) \quad \rightarrow \quad \left\{ \begin{array}{ll} L_1. \mathcal{A} \quad \vdash F \Rightarrow H & (\mathcal{R}) \\ L_2. \mathcal{A} \quad \vdash \neg F \vee H & (Taut L_1) \end{array} \right.$$

We need not to change any refutation graphs.

We change  $\Delta$  such that clauses previously connected with  $L_1$  are afterwards connected with  $L_2$ .

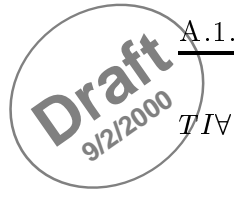
$TI \Leftrightarrow$ :

$$L_1. \mathcal{A} \quad \vdash F \Leftrightarrow G \quad (\mathcal{R}) \quad \rightarrow \quad \left\{ \begin{array}{ll} L_1. \mathcal{A} \quad \vdash F \Leftrightarrow G & (\mathcal{R}) \\ L_2. \mathcal{A} \quad \vdash (F \Rightarrow G) \wedge (G \Rightarrow F) & (Taut L_1) \end{array} \right.$$

We need not to change any refutation graphs.

We change  $\Delta$  such that clauses previously connected with  $L_1$  are afterwards connected with  $L_2$ .





*TI∀*

$$\begin{array}{lcl} L_1. \mathcal{A} & \vdash \forall x.F & (\mathcal{R}) \\ L_2. \mathcal{A} & \vdash H & (G) \end{array} \rightarrow \left\{ \begin{array}{lcl} L_1. \mathcal{A} & \vdash \forall x.F & (\mathcal{R}) \\ L_3. \mathcal{A} & \vdash F[t] & (\forall E) \\ L_2. \mathcal{A} & \vdash H & (G') \end{array} \right.$$

where  $t$  has to be a ground term. Furthermore,  $t$  has to occur in the ground substitution of some clauses of  $G$  that are connected with  $L_1$  such that the ground substitution contains a pair  $x' \mapsto t$  (where  $x'$  is the renaming of  $x$  in the clause).

$G'$  results from  $G$  by applying at each clause which is connected to  $L_1$  and whose ground substitution contains the pair  $x' \mapsto t$  the substitution  $\{x' \mapsto t\}$  and by removing the pair  $x' \mapsto t$  from the ground instantiation.

We change the  $\Delta$ -relation such that the changed clauses (which were previously connected with  $L_1$  and contain a pair  $x' \mapsto t$  in their ground substitution) are afterwards connected with  $L_3$ .

*TMChoice*

$$\begin{array}{lcl} L_1. \mathcal{A} & \vdash \exists x.F & (\mathcal{R}) \\ L_2. \mathcal{A} & \vdash H & (G) \end{array} \rightarrow \left\{ \begin{array}{lcl} L_1. \mathcal{A} & \vdash \exists x.F & (\mathcal{R}) \\ L_3. L_2 & \vdash F[c] & (Hyp) \\ L_4. \mathcal{A}, L_2 & \vdash H & (G') \\ L_2. \mathcal{A} & \vdash H & (Choice\ L_1\ L_4) \end{array} \right.$$

where  $c$  has to be a new constant.

$G'$  results from  $G$  by replacing in both the clauses and the ground substitutions the skolem term that corresponds to  $x$  by  $c$ .

We change the  $\Delta$ -relation such that the clauses previously connected with  $L_1$  are afterwards connected with  $L_3$ .

*TI¬:*

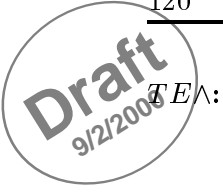
$$L_1. \mathcal{A} \vdash F \quad (\mathcal{R}) \rightarrow \left\{ \begin{array}{lcl} L_1. \mathcal{A} & \vdash G & (\mathcal{R}) \\ L_2. \mathcal{A} & \vdash G^\neg & (Taut\ L_1) \end{array} \right.$$

where  $G$  and  $G^\neg$  are one of the following pairs:

	$F$	$F^\neg$
(1)	$\neg(F \vee H)$	$\neg F \wedge \neg H$
(2)	$\neg(F \wedge H)$	$\neg F \vee \neg H$
(3)	$\neg\neg F$	$F$
(4)	$\neg(F \Rightarrow H)$	$F \wedge \neg H$
(5)	$\neg\forall x.F$	$\exists x.\neg F$
(6)	$\neg\exists x.F$	$\forall x.\neg F$
(7)	$\neg(F \Leftrightarrow H)$	$\neg(F \Rightarrow H) \vee \neg(H \Rightarrow F)$

We need not to change any refutation graphs.

We change  $\Delta$  such that clauses previously connected with  $L_1$  are afterwards connected with  $L_2$ .



$TE\wedge$ :

$$L_1. \mathcal{A} \quad \vdash F \wedge H \quad (G) \quad \rightarrow \quad \left\{ \begin{array}{ll} L_2. \mathcal{A} \quad \vdash F & (G') \\ L_3. \mathcal{A} \quad \vdash H & (G'') \\ L_1. \mathcal{A} \quad \vdash F \wedge H & (\wedge I \ L_2 \ L_3) \end{array} \right.$$

$G'$  and  $G''$  result from  $G$  by breaking the clauses connected with  $F \wedge H$  into two clauses, respectively, one clause in the CNF of  $\neg F$  and one in the CNF of  $\neg H$  ( $L_2$  and  $L_3$  are both open lines, thus, their formulas are negated for clause normalization).

We change  $\Delta$  such that the new clauses resulting from the breaking are connected with  $L_2$  and  $L_3$ , respectively, depending on whether they are in the CNF of  $\neg F$  or  $\neg H$ .

$TE\vee$ :

$$L_1. \mathcal{A} \quad \vdash F \vee H \quad (G) \quad \rightarrow \quad \left\{ \begin{array}{ll} L_2. \mathcal{A} \quad \vdash \neg F & (Hyp) \\ L_3. \mathcal{A}, L_2 \vdash H & (G) \\ L_4. \mathcal{A} \quad \vdash \neg F \Rightarrow H & (\Rightarrow I \ L_3) \\ L_1. \mathcal{A} \quad \vdash F \vee H & (Taut \ L_4) \end{array} \right.$$

We need not to change any refutation graphs.

We change  $\Delta$  such that clauses previously connected with  $L_1$  are afterwards connected with  $L_2$  or  $L_3$ , respectively, depending on whether they are in the CNF of  $\neg F$  or  $\neg H$  ( $L_3$  is an open ND-line).

$TE\Rightarrow$ :

$$L_1. \mathcal{A} \quad \vdash F \Rightarrow H \quad (G) \quad \rightarrow \quad \left\{ \begin{array}{ll} L_2. \mathcal{A} \quad \vdash F & (Hyp) \\ L_3. \mathcal{A}, L_2 \vdash H & (G) \\ L_1. \mathcal{A} \quad \vdash F \Rightarrow H & (\Rightarrow I \ L_3) \end{array} \right.$$

We need not to change any refutation graphs.

We change  $\Delta$  such that clauses previously connected with  $L_1$  are afterwards connected with  $L_2$  or  $L_3$ , respectively, depending on whether they are in the CNF of  $\neg F$  or  $\neg H$  ( $L_3$  is an open ND-line).

$TE\Leftrightarrow$ :

$$L_1. \mathcal{A} \quad \vdash F \Leftrightarrow H \quad (G) \quad \rightarrow \quad \left\{ \begin{array}{ll} L_2. \mathcal{A} \quad \vdash (F \Rightarrow H) \wedge (H \Rightarrow F) & (G) \\ L_1. \mathcal{A} \quad \vdash F \Leftrightarrow H & (Taut \ L_2) \end{array} \right.$$

We need not to change any refutation graphs.

We change  $\Delta$  such that clauses previously connected with  $L_1$  are afterwards connected with  $L_2$ .

$TE\forall$ :

$$L_1. \mathcal{A} \quad \vdash \forall x.F \quad (G) \quad \rightarrow \quad \left\{ \begin{array}{ll} L_2. \mathcal{A} \quad \vdash F[c] & (G') \\ L_1. \mathcal{A} \quad \vdash \forall x.F & (\forall I \ L_2) \end{array} \right.$$

**Draft**  
9/2/2000

where  $c$  has to be a new constant.

$G'$  results from  $G$  by replacing in both the clauses and the ground substitutions the skolem term that corresponds to  $x$  by  $c$ .

We change the  $\Delta$ -relation such that all clauses previously connected with  $L_1$  are afterwards connected with  $L_2$ .

$TE\exists$ :

$$L_1. \mathcal{A} \quad \vdash \exists x.F \quad (G) \quad \rightarrow \quad \left\{ \begin{array}{ll} L_2. \quad L_2 \quad \vdash \neg \exists x.F & (Hyp) \\ L_3. \quad L_2 \quad \vdash \forall x. \neg F & (Taut \ L_2) \\ L_4. \quad \mathcal{A}, L_2 \vdash \perp & (G) \\ L_1. \quad \mathcal{A} \quad \vdash \exists x.F & (Indirect \ L_4) \end{array} \right.$$

We need not to change any refutation graphs.

We change  $\Delta$  such that clauses previously connected with  $L_1$  are afterwards connected with  $L_3$ .

$TE\neg$ :

$$L_1. \mathcal{A} \quad \vdash F \quad (G) \quad \rightarrow \quad \left\{ \begin{array}{ll} L_2. \mathcal{A} \quad \vdash G^\neg & (G) \\ L_1. \mathcal{A} \quad \vdash G & (Taut \ L_2) \end{array} \right.$$

where  $G$  and  $G^\neg$  are one of the following pairs:

	$F$	$F^\neg$
(1)	$\neg(F \vee H)$	$\neg F \wedge \neg H$
(2)	$\neg(F \wedge H)$	$\neg F \vee \neg H$
(3)	$\neg\neg F$	$F$
(4)	$\neg(F \Rightarrow H)$	$F \wedge \neg H$
(5)	$\neg\forall x.F$	$\exists x. \neg F$
(6)	$\neg\exists x.F$	$\forall x. \neg F$
(7)	$\neg(F \Leftrightarrow H)$	$\neg(F \Rightarrow H) \vee \neg(H \Rightarrow F)$

We need not to change any refutation graphs.

We change  $\Delta$  such that clauses previously connected with  $L_1$  are afterwards connected with  $L_2$ .

## A.2 The Alternative Rules

$TMInf$ :

$$\begin{array}{ll} L_1. \mathcal{A} & \vdash F \Rightarrow H \\ L_2. \mathcal{A} & \vdash I \end{array} \quad \begin{array}{l} (\mathcal{R}) \\ (G) \end{array} \quad \rightarrow \quad \left\{ \begin{array}{ll} L_3. \mathcal{A} & \vdash F \\ L_1. \mathcal{A} & \vdash F \Rightarrow H \\ L_4. \mathcal{A} & \vdash H \\ L_2. \mathcal{A} & \vdash I \end{array} \right. \quad \begin{array}{l} (G') \\ (\mathcal{R}) \\ (\Rightarrow E \ L_3 \ L_2) \\ (G'') \end{array}$$

$G'$  and  $G''$  result from  $G$  by breaking the clauses connected with  $L_1$  into two clauses, respectively, one clause in the CNF of  $\neg F$  ( $L_3$  is an open node) and one clause in the CNF of  $H$ .

We change  $\Delta$  such that the new clauses resulting from the breaking are connected with  $L_3$  and  $L_4$ , respectively, depending on whether they are in the CNF of  $\neg F$  or  $H$ .

*TMContra:*

$$\begin{array}{llll} L_1. & \mathcal{A} & \vdash F \Rightarrow H & (\mathcal{R}) \\ L_2. & \mathcal{A} & \vdash I & (G) \end{array} \rightarrow \left\{ \begin{array}{llll} L_3. & \mathcal{A} & \vdash \neg H & (G') \\ L_1. & \mathcal{A} & \vdash F \Rightarrow H & (\mathcal{R}) \\ L_4. & \mathcal{A} & \vdash \neg F & (MT \ L_3 \ L_2) \\ L_2. & \mathcal{A} & \vdash I & (G'') \end{array} \right.$$

$G'$  and  $G''$  result from  $G$  by breaking the clauses connected with  $L_1$  into two clauses, respectively, one clause in the CNF of  $\neg F$  and one clause in the CNF of  $\neg\neg H$  ( $L_3$  is an open line).

We change  $\Delta$  such that the new clauses resulting from the breaking are connected with  $L_3$  and  $L_4$ , respectively, depending on whether they are in the CNF of  $\neg F$  or  $\neg\neg H$ .

*TM  $\vee$  left:*

$$\begin{array}{llll} L_1. & \mathcal{A} & \vdash F \vee H & (\mathcal{R}) \\ L_2. & \mathcal{A} & \vdash I & (G) \end{array} \rightarrow \left\{ \begin{array}{llll} L_3. & \mathcal{A} & \vdash \neg F & (G') \\ L_1. & \mathcal{A} & \vdash F \vee H & (\mathcal{R}) \\ L_4. & \mathcal{A} & \vdash H & (\vee E_L \ L_3 \ L_2) \\ L_2. & \mathcal{A} & \vdash I & (G'') \end{array} \right.$$

$G'$  and  $G''$  result from  $G$  by breaking the clauses connected with  $L_1$  into two clauses, respectively, one clause in the CNF of  $\neg\neg F$  ( $L_3$  is an open line) and one clause in the CNF of  $H$ .

We change  $\Delta$  such that the new clauses resulting from the breaking are connected with  $L_3$  and  $L_4$ , respectively, depending on whether they are in the CNF of  $\neg\neg F$  or  $H$ .

*TM  $\vee$  right:*

$$\begin{array}{llll} L_1. & \mathcal{A} & \vdash F \vee H & (\mathcal{R}) \\ L_2. & \mathcal{A} & \vdash I & (G) \end{array} \rightarrow \left\{ \begin{array}{llll} L_3. & \mathcal{A} & \vdash \neg H & (G') \\ L_1. & \mathcal{A} & \vdash F \vee H & (\mathcal{R}) \\ L_4. & \mathcal{A} & \vdash F & (\vee E_R \ L_3 \ L_2) \\ L_2. & \mathcal{A} & \vdash I & (G'') \end{array} \right.$$

$G'$  and  $G''$  result from  $G$  by breaking the clauses connected with  $L_1$  into two clauses, respectively, one clause in the CNF of  $\neg\neg H$  ( $L_3$  is an open line) and one clause in the CNF of  $F$ .

We change  $\Delta$  such that the new clauses resulting from the breaking are connected with  $L_3$  and  $L_4$ , respectively, depending on whether they are in the CNF of  $\neg\neg H$  or  $F$ .

In Section 3.5 we illustrate with an example that the alternative rules are not always applicable. We examine now when the rule *TM  $\vee$  left* is applicable.

$$\begin{array}{lcl}
 L_1. \mathcal{A} & \vdash F \vee H & (\mathcal{R}) \\
 L_2. \mathcal{A} & \vdash I & (G)
 \end{array}
 \rightarrow
 \left\{
 \begin{array}{lcl}
 L_3. \mathcal{A} & \vdash \neg F & (G') \\
 L_1. \mathcal{A} & \vdash F \vee H & (\mathcal{R}) \\
 L_4. \mathcal{A} & \vdash H & (\vee E_L \ L_3 \ L_2) \\
 L_2. \mathcal{A} & \vdash I & (G'')
 \end{array}
 \right.$$

The question is whether  $\neg F$  can still be proved by the hypotheses  $\mathcal{A}$ . To check this, we can try the following: We break the refutation graph  $G$  into the two graphs  $G'$  and  $G''$ , such that  $G'$  contains instead of clauses in the CNF of  $F \vee H$  only clauses in the CNF of  $\neg F$  and  $G''$  contains only clauses in the CNF of  $H$ . If  $G'$  does not contain any clauses connected with  $L_2$  then  $G'$  proves

$$\mathcal{A}, F \vdash \perp$$

since  $G'$  depends then only of clauses derivable from  $\mathcal{A}$  and  $F$ . Therefore, we can deduce:

$$\mathcal{A} \vdash \neg F$$

Thus, in this case we can prove  $\neg F$  from hypotheses  $\mathcal{A}$  and the application of  $TM \vee left$  is correct.

On the other hand, would  $G''$  contain no clause connected with  $L_2$ , we could apply for the same reason  $TM \vee right$ . Would both  $G'$  and  $G''$  contain clauses connected with  $L_2$  we would have to use the standard rule  $TM Cases$ . The same schema (first break the refutation graph, then check in which of the resulting graphs clauses are still connected with the open line; then choose the appropriate rule) is also applicable on the rules  $TM Contra$ ,  $TM Inf$ , and  $TI \Rightarrow$ . If the one resulting refutation graphs – the one with clauses in the CNF of  $\neg F$  – contains no clauses connected with the corresponding open line, we can chose  $TM Inf$ ; if the other resulting refutation graph – the one with clauses in the CNF of  $H$  – contains no clauses connected with the corresponding open line, we can chose  $TM Contra$ ; if both resulting refutation graphs contain clauses connected with the corresponding open line, we have to chose  $TI \Rightarrow$ . Note that this breaking of the refutation graphs to check which rule is applicable is no extra effort. To apply one of the rules we would have to split the refutation graph anyway. We change only the order by first breaking the graph and then deciding which rule to apply. Furthermore, note that this heuristic subsumes the heuristics for possible applications of  $TM Inf$  given in [Lin90].

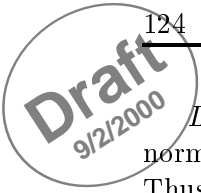
$TE \exists \perp direct$ :

$$\begin{array}{lcl}
 L_1. \mathcal{A} & \vdash \exists x.F & (G)
 \end{array}
 \rightarrow
 \left\{
 \begin{array}{lcl}
 L_2. \mathcal{A} & \vdash F[t] & (G') \\
 L_1. \mathcal{A} & \vdash \exists x.F & (\exists I \ L_2)
 \end{array}
 \right.$$

where  $t$  has to be a ground term. Furthermore,  $t$  has to occur in the ground substitution of some clauses of  $G$  that are connected with  $L_1$  such that the ground substitution contains a pair  $x' \mapsto t$  ( where  $x'$  is the renaming of  $x$  in the clause).

$G'$  results from  $G$  by applying at each clause which is connected to  $L_1$  and whose ground substitution contains the pair  $x' \mapsto t$  the substitution  $\{x' \mapsto t\}$  and by removing the pair  $x' \mapsto t$  from the ground instantiation.

We change the  $\Delta$ -relation such that the changed clauses (which were previously connected with  $L_1$  and contain a pair  $x' \mapsto t$  in their ground substitution) are afterwards connected with  $L_3$ .



$L_1$  is an open line, so its formula is negated for clause normalization and we start the clause normalization with  $\neg\exists x.F$  which is equivalent to the universal quantified formula  $\forall x\neg F$ . Thus, in the clauses connected with this line we have free variables for  $x$  and there can exist several ground instantiations of these clauses in  $G$ . The rule  $TE\exists \perp direct$  is applicable only if there exists at most one such term  $t$ . This heuristic is directly taken from [Lin90].

## Appendix B

# Breaking Refutation Graphs

In this section we describe an algorithm breaking clauses and refutation graphs. If we apply transformation rules on a ND-line whose formula used for clause normalization is a  $\beta$ -formula, we have to split clauses and refutation graphs (e.g., in the transformation rules *TMCases*, *TE $\wedge$* , or *TMInf*).

Since for the CNF of a  $\beta$ -formula holds  $CNF(\beta) = CNF(\beta_1) \times CNF(\beta_2)$  each clause in  $CNF(\beta)$  consists of a part in  $CNF(\beta_1)$  and a part in  $CNF(\beta_2)$ . If we decompose the  $\beta$ -formula into  $\beta_1$  and  $\beta_2$  we have to break each clause  $C$  in  $CNF(\beta)$  into a clause  $C'$  in  $CNF(\beta_1)$  and a clause  $C''$  in  $CNF(\beta_2)$ . This is necessary to guarantee invariant 3.

This breaking of clauses effects also the refutation graphs. First, after the breaking of a clause the resulting graph is not minimal (look at Example 3.2.1 in Section 3.2, we can regard the *two* graphs resulting from a splitting also as *one* non-minimal graph). Secondly, we need in the transformation rules after the decomposition of a  $\beta$ -formula two refutation graphs instead of one. For these two new refutation graphs has to hold that the one graph does not contain any of the clauses  $C'_1, \dots, C'_m$  in  $CNF(\beta_1)$  and the other graph does not contain any of the clauses  $C''_1, \dots, C''_m$  in  $CNF(\beta_2)$  (since the one refutation graph only depends on  $\beta_1$  and the other only on  $\beta_2$ ).

In this section we give a complete and correct algorithm for the breaking of clauses and refutation graphs. Complete in the sense, that each minimal refutation graph is splitted by splitting a set of clauses  $C_1, \dots, C_m$  that are splitted into  $C'_1, \dots, C'_m$  and  $C''_1, \dots, C''_m$ , respectively. Correct in the sense that we obtain as result two minimal refutation graphs satisfying the condition that the one does not contain any of the clauses  $C'_1, \dots, C'_m$  and the other does not contain any of the clauses  $C''_1, \dots, C''_m$ .

We give first an algorithm breaking exactly one clause  $C$  into two disjunct clauses  $C'$  and  $C''$ . Thereby two refutation graphs are produced, one not containing  $C'$  and one not containing  $C''$ . Then we use this algorithm to describe an algorithm that is able to handle the breaking of an arbitrary number of clauses.

### B.1 Breaking One Clause

We have the following situation: We have a minimal refutation graph  $G$  containing a clause  $C$  and a disjunct partition of this clause into two non-empty clauses  $C'$  and  $C''$ . Our goal is to have two refutation graphs  $G'$  and  $G''$  such that  $G'$  contains  $C'$  but not  $C''$  (or  $C$ ) and  $G''$  contains  $C''$  but not  $C'$  (or  $C$ ).



The idea is to start the construction of the new graphs with  $C'$  and  $C''$ , respectively, and to add then step-by-step such links and other clauses from  $G$  that still pure literal nodes in the new graphs are inked.

**Algorithm B.1.1 (Single-clause-Breaking Algorithm).**

Let  $G$  be a minimal refutation graph and let  $C$  be a clause in  $G$ . Let  $C'$  and  $C''$  be a disjunct partition of  $C$  such that  $C'$  and  $C''$  are non empty clauses.

**Initialization-step:** Assign the new graph  $G_{new} = C'$  or  $C''$ , respectively.

Assign the set of the pure literal nodes of  $G_{new}$ ,  $S_{Lp} = \{L | L \text{ is literal of } C' \text{ or } C''\}$ , respectively.

**Linking-step:** As long as  $S_{Lp}$  is not empty:

Choose a literal  $L$  in  $S_{Lp}$

Compute the set of literals linked with  $L$  in the original graph  $G$  by a link  $\Lambda$ .

Choose one literal  $L_{contra}$  of these contradictional literals which is contained in a clause  $C_{contra}$  of  $G$ , such that:

**Case 1** There is yet no copy of  $C_{contra}$  and there is yet no copy of link  $\Lambda$  in  $G_{new}$ .

**Case 2** There is already a copy of  $C_{contra}$  and there is already a copy of  $\Lambda$  in the new refutation graph and  $L_{contra}$  is already connected with this copy of  $\Lambda$ .

Then do:

**Case 1** Add a copy of  $C_{contra}$  and  $\Lambda$  into the new refutation graph and connect the literals  $L$  and  $L_{contra}$  in the new refutation graph by  $\Lambda$ .

$$S_{Lp} = S_{Lp} \cup \{L' | L' \text{ is literal of } C_{contra}\} \perp \{L, L_{contra}\}$$

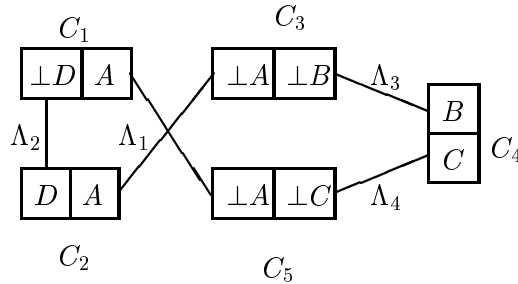
**Case 2** Add  $L$  to the copy of  $\Lambda$  in the new refutation graph.

$$S_{Lp} = S_{Lp} \perp \{L\}.$$

■

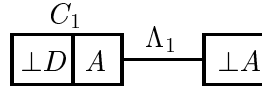
Before we prove the completeness and the correctness of this algorithm we demonstrate its use in a small example.

*Example B.1.2.* Given is the refutation graph  $G$ :

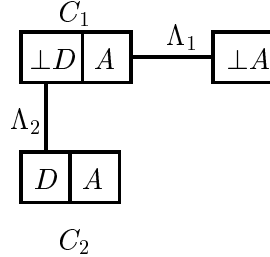


We break the clause  $C_3 = [\neg A, \neg B]$  into  $[\neg A]$  and  $[\neg B]$ . We start our algorithm with clause  $[\neg A]$  as new graph. To link literal  $\neg A$  we can either copy  $C_1$  or  $C_2$  into the new graph and link the literals by a copy of  $\Lambda_1$  (both is an application of Case 1 of our algorithm). We choose here  $C_1$  and obtain the following graph:

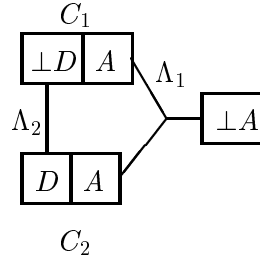




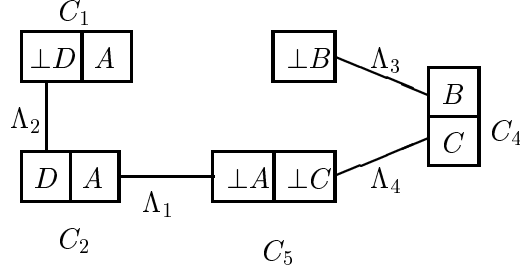
The only pure literal in the resulting graph is  $\neg D$ . The only possibility to link it is to apply again Case 1 such that  $C_2$  and  $\Lambda_2$  are copied into the new graph. We obtain as resulting graph:



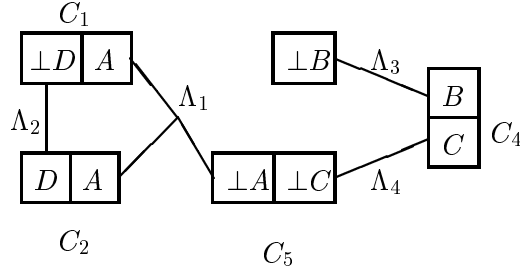
If we apply now Case 2 on the pure literal  $A$  we obtain as final result the following minimal refutation graph:



If we start to construct a refutation graph with the other clause  $[\neg B]$  resulting from the breaking, we have to introduce first a copy of clause  $C_4$  and link  $\Lambda$  (by an application of Case 1); then we can apply three times Case 1 copying  $C_5$  and  $\Lambda_4$ ,  $C_2$  and  $\Lambda_1$ , and  $C_1$  and  $\Lambda_2$  into the new graph, such that we obtain the following graph:



The last pure literal is  $A$  in  $C_1$ . Applying Case 2 we can link it by  $\Lambda_1$ . Thus, we obtain as final result the following minimal refutation graph:



■

We prove now some lemmas before we prove the completeness and the correctness of our algorithm.

**Lemma B.1.3.**

Let  $G$  be a minimal refutation graph and  $C$  a clause in  $G$ . Let  $C'$  and  $C''$  be a disjunct partition of  $C$  such that  $C'$  and  $C''$  are two non empty clauses. Then during the whole application of Algorithm B.1.1 for the new graph  $G_{new}$  under construction holds:

- (1) There is a trail from each clause in  $G_{new}$  to each other clause in  $G_{new}$  and the same trail exists also in the original graph  $G$ . Thereby, 'same' means that the original clauses of the clauses in  $G_{new}$  are connected by a trail using the original links and clauses of the links and clauses of the trail in  $G_{new}$ , respectively. Furthermore,  $C$  is regarded as the original clause in  $G$  to  $C'$  or  $C''$  in  $G_{new}$ .
- (2)  $G_{new}$  is a minimal deduction graph.

**Proof:** We prove both propositions by induction about the number  $n$  of applied linking-steps.

**Induction Base Case:**  $n = 0$ 

We are in the situation after the initialization-step.

- (1) holds, since we have only one clause.
- (2) holds by Theorem 5.1.2, since we have only one clause and no links in the new graph.

**Induction Step:**  $n \rightarrow n + 1$ 

We call the graph before the application of the linking-step  $G'_{new}$  and the graph after the application of the linking step  $G''_{new}$ .

1. We apply Case 1:

- (1) We add a new clause  $C_{new}$  to  $G'_{new}$  and link a literal of  $C_{new}$  with a literal of a clause  $C'_{new}$  (already in  $G'_{new}$ ) by a link  $\Lambda$  that we also add to  $G'_{new}$ . By induction hypothesis there is a trail from each clause in  $G'_{new}$  to each other clause in  $G'_{new}$ . In particular, there is a trail from each clause to  $C'_{new}$ . Thus, we have also a trail from each clause to  $C_{new}$  in  $G'_{new}$ , by using the trail to  $C'_{new}$  and then using the *new* link  $\Lambda$  (new and hence not yet contained in the trail to  $C'_{new}$ ). Thus we have that in  $G''_{new}$  there is a trail from each clause to each other clause.

By induction hypothesis all trails in  $G'_{new}$  are also trails in  $G$ . Since both  $\Lambda$  and  $C_{new}$  are copies of a link or a clause of  $G$ , respectively, all trails in  $G''_{new}$  are also trails in  $G$ .

- (2) Assume  $G''_{new}$  is cyclic. Then there would be a trail from a clause to itself. From (1) we know that all trails in  $G''_{new}$  are also in  $G$ . Thus, we would have a cycle in  $G$ , too. But this is a contradiction to the precondition that  $G$  is a refutation graph. Therefore,  $G''_{new}$  is acyclic. Furthermore,  $G''_{new}$  can not be empty since it contains at least  $C'$  or  $C''$ . Thus, we have that  $G''_{new}$  is a deduction graph.

By induction hypothesis  $G'_{new}$  is a minimal deduction graph. Thus,  $G'_{new}$  contains By Theorem 5.1.2 one clause more than links. We add exactly one clause and one link, hence  $G''_{new}$  has still one clause more than links. Hence, we have by Theorem 5.1.2 that  $G''_{new}$  is minimal, too.



2. We apply Case 2:

- (1) We add neither a new clause nor a new link to  $G'_{new}$ . Thus, it follows directly by induction hypothesis from  $G'_{new}$  that in  $G''_{new}$  each clause is connected with each other clause by a trail.  
By induction hypothesis all trails in  $G'_{new}$  are also trails in  $G$ . Since  $\Lambda$  is a copy of a link of  $G$ , all new trails in  $G''_{new}$  using the updated  $\Lambda$  are also trails in  $G$ .
- (2) That  $G''_{new}$  is a minimal deduction graph follows by Theorem 5.1.2 and by the induction hypothesis on  $G'_{new}$  (since we add neither a clause nor a link there are still one clause more than links in  $G''_{new}$ ).

□

**Lemma B.1.4.**

*Let  $G$  be a minimal refutation graph and  $C$  a clause in  $G$ . Let  $C'$  and  $C''$  be a disjunct partition of  $C$  such that  $C'$  and  $C''$  are two non empty clauses. Let  $G_{new}$  be the graph resulting by the construction that starts with clause  $C'$ . Then during the Algorithm B.1.1 it can not happen that for a pure literal  $L$  in  $G_{new}$  there exists only a contradictory literal  $L_{contra}$  which is in clause  $C''$  (this would force us to introduce  $C''$  into  $G_{new}$  to link  $L$  with  $L_{contra}$ ). The analogous statement holds if we start with  $C''$  to construct the new graph.*

**Proof:** Assume that for a pure literal  $L$  there exists only a literal  $L_{contra}$  linked with  $L$  such that  $L_{contra}$  is in clause  $C''$ . Furthermore, assume that we would introduce  $C''$  and the corresponding link into  $G_{new}$  by an application of Case 1.

Then by Lemma B.1.3 there would be a trail between  $C'$  to  $C''$  in  $G_{new}$ . By Lemma B.1.3 we would have also that this trail is in  $G$ , too. Since  $C'$  and  $C''$  are in  $G$  united together to  $C$ , we would have a cycle from  $C$  to itself in  $G$ . But this is a contradiction to the hypothesis that  $G$  is a refutation graph. □

We prove now the completeness of the Algorithm B.1.1.

**Theorem B.1.5 (Completeness of Algorithm B.1.1).**

*Let  $G$  be a minimal refutation graph and  $C$  a clause in  $G$ . Let  $C'$  and  $C''$  be a disjunct partition of  $C$  such that  $C'$  and  $C''$  are two non empty clauses. Then the Algorithm B.1.1 terminates and produces two graphs with no pure literals.*

**Proof:** The algorithm transfers clauses and links from the original refutation graph into the new graph as long as there are pure literals. Since both Case 1 and Case 2 allow only the addition of at most one copy of a clause or a link of the original graph into the new graph the algorithm has to terminate. The question is whether the two possibilities *Case1* and *Case2* are enough to add clauses and links until there are no pure literal nodes.

Assume we have a pure literal  $L$  in the new graph. Since  $L$  is also in the original graph, we can compute the set of literals that are linked with  $L$  in the original graph by a link  $\Lambda$ . If we choose one literal  $L_{contra}$  of these contradictional literals which is contained in a clause  $C_{contra}$  there can be the following situations:

**Case 1:** There is yet no copy of  $C_{contra}$  and there is yet no copy of link  $\Lambda$  in the new refutation graph. Then we can copy  $C_{contra}$  into the new refutation graph and can connect the literals  $L$  and  $L_{contra}$  in the new refutation graph by a copy of link  $\Lambda$ .

**Draft**  
9/2/2009

**Case 2:** There is already a copy of  $C_{contra}$  and there is already a copy of  $\Lambda$  in the new refutation graph and  $L_{contra}$  is already connected with this copy of  $\Lambda$ . Then we can add  $L$  to this copy of  $\Lambda$  in the new refutation graph.

These two cases are exactly the two cases also used in the Algorithm B.1.1.

**Case 3:** There is already a copy of  $C_{contra}$  and there is already a copy of  $\Lambda$  in the new refutation graph but  $L_{contra}$  is not yet connected with this copy of  $\Lambda$ .

**Case 4:** There is yet no copy of  $C_{contra}$  and there is already a copy of  $\Lambda$  in the new refutation graph.

These two cases we can reduce to Case 2 by choosing another literal  $L'_{contra}$  linked with  $L$  in the original refutation graph instead of  $L_{contra}$ . Since Case 1 is in our algorithm the only case that adds new links into the new refutation graph we guarantee that a link in the new refutation graph has never empty shores (since Case 1 always constructs links with non empty shores). Therefore, if we have Case 3 or Case 4 there has to be also another literal  $L'_{contra}$  in another clause  $C'_{contra}$  such that Case 2 is applicable.

**Case 5:** There is already a copy of  $C_{contra}$  and there is yet no copy of  $\Lambda$  in the new refutation graph.

In this situation we could add a copy of  $\Lambda$  that links  $L_{contra}$  and  $L$  in the new refutation graph. Indeed, we prove now that this situation of Case 5 can not happen during an application of the Algorithm B.1.1.

Assume we would reach during an application of Algorithm B.1.1 for the first time a situation in which case 5 would be applicable. Then we would have clauses  $C_1$  and  $C_2$  in  $G_{new}$  with contradictory literals  $L_1$  and  $L_2$ . Both  $L_1$  and  $L_2$  would be pure (since there is no copy of the link  $\Lambda$ ). Furthermore,  $G_{new}$  would be constructed so far by (maybe zero) applications of case 1 and case 2.

Then by Lemma B.1.3 there is a trail between  $C_1$  and  $C_2$ . If introduce the *new* link  $\Lambda$  we would obtain a cycle. Since all trails in the new graph  $G_{new}$  are also in the original graph this there would be a cycle in  $G$ , too. But this is a contradiction to the hypothesis that  $G$  is a refutation graph.

Thus, Case 1 and Case 2 used in the Algorithm B.1.1 are enough to link every pure literal in the new graph such that the Algorithm B.1.1 computes after a finite sequence of linking-steps two graphs  $G'$  and  $G''$  which contain no pure literals.  $\square$

**Theorem B.1.6 (Correctness of Algorithm B.1.1).**

*Let  $G$  a minimal refutation graph and  $C$  a clause in  $G$ . Let  $C'$  and  $C''$  be a disjunct partition of  $C$  such that both  $C'$  and  $C''$  are non empty clauses. Then the Algorithm B.1.1 creates two graphs  $G'$  and  $G''$  such that:*

1.  $G'$  and  $G''$  are minimal refutation graphs;
2.  $G'$  contains the clause  $C'$  but not the clause  $C''$ , whereas  $G''$  contains the clause  $C''$  but not the clause  $C'$ .

Draft  
9/2/2000

**Proof:** By the completeness Theorem B.1.5 we know that the algorithm terminates with two graphs with no pure literals. By Lemma B.1.3 we know that these graphs are minimal deduction graphs. Thus, we have that both graphs are minimal refutation graphs.

By construction (see initialization-step)  $G'$  contains  $C'$  and  $G''$  contains  $C''$ . By Lemma B.1.4 we have that they can not contain the other clause, respectively.  $\square$

### Corollary B.1.7.

*The Algorithm B.1.1 produces two distinct graphs.*

**Proof:** Follows immediately by Theorem B.1.6 since  $G'$  contains  $C'$  but not  $C''$  whereas  $G''$  contains  $C''$  but not  $C'$ .  $\square$

Note that if we say 'copy a clause' we mean copying the ground clauses of a refutation graph. If a ground clause is represented by a clause with variables and an additional ground substitution for these variables, we have to copy both the clause and the ground substitution, respectively.

## B.2 Breaking Several Clauses

If we have to break a set of clauses  $C_1, \dots, C_m$  into sets of clauses  $C'_1, \dots, C'_m$  and  $C''_1, \dots, C''_m$ , we need as result – how described at the beginning of this chapter – still only two refutation graphs  $G'$  and  $G''$ , where  $G'$  does not contain any clause of  $C'_1, \dots, C'_m$  and  $G''$  does not contain any clause of  $C''_1, \dots, C''_m$ . We obtain these refutation graphs by breaking the clauses stepwise.

Breaking clause  $C_1$  in graph  $G$  into  $C'_1$  and  $C''_1$  using the Algorithm B.1.1 results in two refutation graphs  $G_1$  and  $G_2$ . By Theorem B.1.6 we know that  $G_1$  does not contain  $C''_1$  and  $G_2$  does not contain  $C'_1$ . If we break in  $G_1$  and  $G_2$  the clause  $C_2$  we obtain the graphs  $G_{11}, G_{12}$  (from  $G_1$ ) and  $G_{21}, G_{22}$  (from  $G_2$ ), where  $G_{11}$  does not contain  $C'_1$  and  $C''_2$  and  $G_{22}$  does not contain  $C'_1$  and  $C'_2$ . We can continue with breaking clause  $C_3$  in the graphs  $G_{11}$  and  $G_{22}$  and so on.

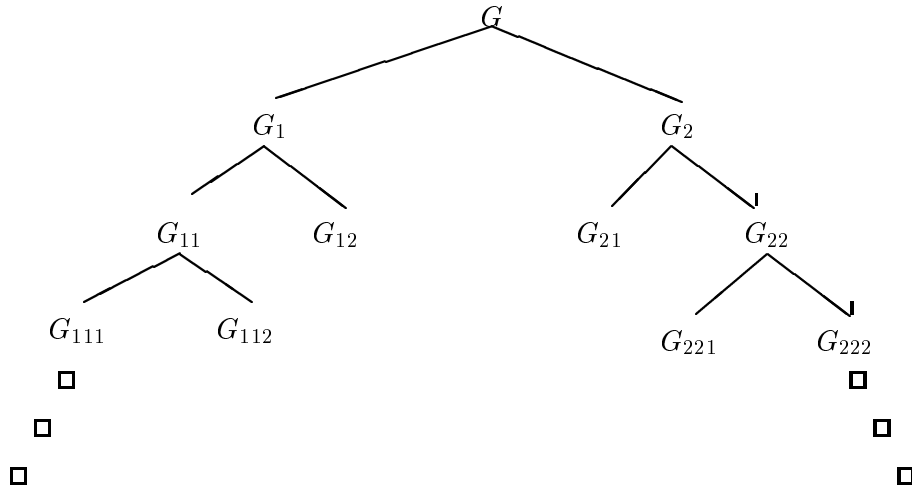


Figure B.1: Tree of Graphs

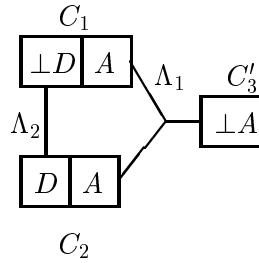
**Draft**  
9/2/2000

Of the resulting tree (see Figure B.1 we need only the two out-most branches. If a graph  $G_{1j}$  or  $G_{2j}$  does not contain the clause  $C_{j+1}$  that has to be broken next, we can assign  $G_{1j+1} = G_{1j}$  or  $G_{2j+1} = G_{2j}$  since then it still holds that  $G_{1j+1}$  does not contain  $C_{j+1}''$  or  $G_{2j+1}$  does not contain  $C_{j+1}'$ .

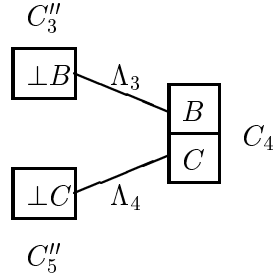
*Example B.2.1.*

Assume we have an open ND-line with formula  $A \wedge (B \vee C)$ . We obtain by clause normalization of the negation of this formula (open lines are negated for clause normalization) the clauses  $[\neg A, \neg B]$  and  $[\neg A, \neg C]$ . Assume we have the refutation  $G$  the graph of Example B.1.2. Both clauses are contained in  $G$ :  $C_3 = [\neg A, \neg B]$  and  $C_5 = [\neg A, \neg C]$ .

If we apply the rule  $TE \wedge$  on the open ND-line we have to break the clauses into:  $C_3' = [\neg A]$ ,  $C_3'' = [\neg B]$ ,  $C_5' = [\neg A]$ , and  $C_5'' = [\neg C]$ . We constructed the graphs  $G_1$  and  $G_2$  already in Example B.1.2. To obtain  $G_{11}$  we have to break  $C_5$  in  $G_1$ . But since  $G_1$  does not contain  $C_5$  we obtain directly  $G_{11} = G_1$ .



To obtain  $G_{22}$  we have to break the clause  $C_5$  in graph  $G_2$ . The following is the graph  $G_{22}$  computed from clause  $C_5''$ :



■

## Appendix C

# The Proof of the UCS-Theorem

In this section we prove the UCS-theorem 4.3.3 presented first in Section 4.3.

**Theorem C.0.2 (UCS-Theorem).**

Let be  $(AC, \{UC_1, \dots, UC_n\}, R_{lit})$  an UCS in a refutation graph  $G$ . Let be  $A$  a formula in whose CNF  $AC$  is and let be  $\sigma$  the ground substitution of  $AC$  in  $G$ . Let be  $L_1, \dots, L_n$  the (respectively unique) literals of  $UC_1, \dots, UC_n$ . Then holds:

$$\frac{L_1, \dots, L_n}{(R_{lit})\sigma}$$

can be justified by an assertion application of assertion  $A$  on preconditions  $L_1, \dots, L_n$ , if

$C_1$ : the clauses  $UC_1, \dots, UC_n$  themselves are already ground in  $G$ , or in other words, their ground substitutions are empty,

$C_2$ : during the clause normalization from  $A$  to  $AC$  the  $\delta$ -rule is never applied.

Thus, we have the following situation for the proof:  $G$  is a refutation graph with a UCS  $(AC, \{UC_1, \dots, UC_n\}, R_{lit})$ .  $A$  is a formula such that  $AC$  is in the CNF of  $A$  and  $\sigma$  is the ground substitution of  $AC$  in  $G$ . Furthermore, the conditions  $C_1$  and  $C_2$  are satisfied.

Since  $AC$  is in the CNF of  $A$  we can create a derivation tree  $T_d$  with  $A$  as root and the literals of  $AC$  as leaves. We call the literals of  $AC$  different to  $R_{lit}$  the *rest literals* and the branch in  $T_d$  from  $AC$  to  $R_{lit}$  the *main branch*. The other branches from  $AC$  to the rest literals we call the *side branches*.

We give now an algorithm that transforms the derivation tree  $T_d$  into a ND-tree that:

- satisfies the compositions and decompositions constraints,
- has a main branch from  $A$  to  $(R_{lit})\sigma$ ,
- has side branches that start with  $L_1, \dots, L_n$  and end at the main branch.

Therefore, this ND-tree represents an assertion application of assertion  $A$  on the premises  $L_1, \dots, L_n$  with result  $(R_{lit})\sigma$ .

The transformation process starts with the formula  $A$  as root of the derivation tree and as start point of the main branch of the assertion application. We work then along the main branch of the derivation tree and transfer stepwise the applications of the clause normalization rules in the derivation tree into corresponding applications of ND-rules in the ND-tree. To transfer the steps we need the following cases:

Draft  
9/2/2020

Main branch rules:

**Case 1:** ( $\alpha$ -rule)

$$\frac{F \wedge G}{F} \quad (\alpha\text{-rule}) \quad \text{becomes} \quad \frac{F \wedge G}{F} \wedge E_L$$

$$\frac{F \wedge G}{G} \quad (\alpha\text{-rule}) \quad \text{becomes} \quad \frac{F \wedge G}{G} \wedge E_R$$

**Case 2:** ( $\alpha$ -rule)

$$\frac{\neg(F \vee G)}{\neg F} \quad (\alpha\text{-rule}) \quad \text{becomes} \quad \frac{\neg(F \vee G)}{\neg F \wedge \neg G} \text{Ta} \quad \frac{\neg F \wedge \neg G}{\neg F} \wedge E_L$$

$$\frac{\neg(F \vee G)}{\neg G} \quad (\alpha\text{-rule}) \quad \text{becomes} \quad \frac{\neg(F \vee G)}{\neg F \wedge \neg G} \text{Ta} \quad \frac{\neg F \wedge \neg G}{\neg G} \wedge E_R$$

**Case 3:** ( $\alpha$ -rule)

$$\frac{\neg(F \Rightarrow G)}{F} \quad (\alpha\text{-rule}) \quad \text{becomes} \quad \frac{\neg(F \Rightarrow G)}{F \wedge \neg G} \text{Ta} \quad \frac{F \wedge \neg G}{F} \wedge E_L$$

$$\frac{\neg(F \Rightarrow G)}{\neg G} \quad (\alpha\text{-rule}) \quad \text{becomes} \quad \frac{\neg(F \Rightarrow G)}{F \wedge \neg G} \text{Ta} \quad \frac{F \wedge \neg G}{G} \wedge E_R$$

**Case 4:** ( $\beta$ -rule)

$$\frac{F \vee G}{F} \quad (\beta\text{-rule}) \quad \text{becomes} \quad \frac{F \vee G \quad \neg F}{G} \vee E_L \quad \text{or} \quad \frac{F \vee G \quad \neg G}{F} \vee E_R$$

if  $G$  is at the main branch of  
the derivation tree (Case 4.1)

if  $F$  is at the main branch of  
the derivation tree (Case 4.2)

**Case 5:** ( $\beta$ -rule)

$$\frac{F \Rightarrow G}{\neg F} \quad (\beta\text{-rule}) \quad \text{becomes} \quad \frac{F \Rightarrow G}{G} \Rightarrow E$$



$$\frac{F \Rightarrow G \quad F}{G} \Rightarrow E \quad \text{or} \quad \frac{F \Rightarrow G \quad \neg G}{\neg F} \text{Contradiction}$$

if  $G$  is at the main branch of  
the derivation tree (Case 5.1)

if  $\neg F$  is at the main branch of  
the derivation tree (Case 5.2)

**Case 6:** ( $\beta$ -rule)

$$\begin{array}{c} \neg(F \wedge G) \\ \swarrow \quad \searrow \\ \neg F \quad \neg G \end{array} \quad (\beta\text{-rule}) \quad \text{becomes}$$

$$\frac{\frac{\neg(F \wedge G)}{\neg F \vee \neg G} \text{Ta} \quad \neg \neg G}{\neg F} \vee E_L \quad \text{or} \quad \frac{\frac{\neg(F \wedge G)}{\neg F \vee \neg G} \text{Ta} \quad \neg \neg F}{\neg G} \vee E_R$$

if  $\neg F$  is at the main branch of  
the derivation tree (Case 6.1)

if  $\neg G$  is at the main branch of  
the derivation tree (Case 6.2)

**Case 7:** ( $\gamma$ -rule)

$$\begin{array}{c} \forall x.F \\ | \\ F[x'] \end{array} \quad (\gamma\text{-rule}) \quad \text{becomes} \quad \frac{\forall x.F}{F[t]} \forall E$$

If the ground substitution  $\sigma$  contains for  $x'$  the pair  $x' \mapsto t$ .

**Case 8:** ( $\gamma$ -rule)

$$\begin{array}{c} \neg \exists x.F \\ | \\ \neg F[x'] \end{array} \quad (\gamma\text{-rule}) \quad \text{becomes} \quad \frac{\frac{\neg \exists x.F}{\forall x. \neg F} \text{Ta}}{\neg F[t]} \forall E$$

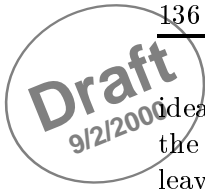
If the ground substitution  $\sigma$  contains for  $x'$  the pair  $x' \mapsto t$ .

**Case 9:** ( $\neg\neg$ -rule)

$$\begin{array}{c} \neg\neg F \\ | \\ F \end{array} \quad (\neg\neg\text{-rule}) \quad \text{becomes} \quad \frac{\neg\neg F}{F} \text{Ta}$$

Condition  $C_2$  of the UCS-theorem forbids the use of the  $\delta$ -rule (see Section 4.3). When we apply these transformation cases along the main branch of the derivation tree we create successively the main branch of the assertion application. It is an invariant of this process (that we shall prove in Lemma C.0.5) that the formula at the end node of the ND-branch (as derived so far) and the formula at the current regarded node in the derivation tree are equal modulo the ground substitution  $\sigma$  (the formulas are equal if  $\sigma$  is applied on the formulas in the derivation tree). This guarantees to reach at the end  $(R_{lit})\sigma$  as formula of the last node of the main branch of the ND-tree.

The side branches of the ND-tree are created backwardly. We perform not explicitly the inversion of the side branches of the derivation tree how we do in Section 4.3 to illustrate the



ideas of the proof. We start working on the side branches at the node on that they start at the main branch of the derivation tree; then we work along the branches until we reach the leaves of the derivation tree. But we have the problem that in the cases of the main branch rules that handle the connection of a side branch there can be two different situations:

1. In Cases 4.1, 4.2, 5.2, 6.1 and 6.2 we need  $\neg G$  at the side branch of the ND-tree where  $\neg G$  is the formula of the corresponding node in the side branch of the derivation tree. Thus, we need here that the formula in the side branch of the ND-tree is the negation of the formula in the side branch of the derivation tree.
2. In Case 5.1 we need  $F$  in the side branch of the ND-tree where  $\neg F$  is the formula of the corresponding node in the side branch of the derivation tree. Thus, we need here that the formula in the side branch of the derivation tree is the negation of the formula in the side branch of the ND-tree.

Since we construct the side branches of the ND-tree backwardly, we have not only to distinguish the cases of the different connectives and quantifiers but also whether we need that the formula in the ND-tree is the negation of the formula in the derivation tree or vice versa. Altogether, we need the following cases:

**Side branch rules:**

**Case 1:** ( $\alpha$ -rule)

Needed formula in ND-tree is negation of the formula in the derivation tree.

$$\begin{array}{ccc}
 \begin{array}{c} F \wedge G \\ | \\ F \end{array} & (\alpha\text{-rule}) & \text{becomes} \quad \frac{\frac{\neg F}{\neg F \vee \neg G} \vee I_L}{\neg(F \wedge G)} T_{aut} \\
 \\
 \begin{array}{c} F \wedge G \\ | \\ G \end{array} & (\alpha\text{-rule}) & \text{becomes} \quad \frac{\frac{\neg G}{\neg F \vee \neg G} \vee I_R}{\neg(F \wedge G)} T_{aut}
 \end{array}$$

Here and in the cases 4, 5, 7 the formulas  $F \wedge G$ ,  $F \vee G$ ,  $F \Rightarrow G$  and  $\forall x.F$  in the derivation tree are explicitly not negations. Therefore, the case that the formula in the derivation tree is the negation of the needed formula in the ND-tree is not possible.

**Case 2:** ( $\alpha$ -rule)

$$\begin{array}{ccc}
 \begin{array}{c} \neg(F \vee G) \\ | \\ \neg F \end{array} & (\alpha\text{-rule}) & \text{becomes} \\
 \\
 \frac{F}{F \vee G} \vee I_L & & \frac{\frac{F}{F \vee G} \vee I_L}{\neg\neg(F \vee G)} T_{aut}
 \end{array}$$

if  $F \vee G$  is needed in the ND-tree      if  $\neg\neg(F \vee G)$  is needed in the ND-tree

$$\frac{\neg(F \vee G)}{\neg G} \quad (\alpha\text{-rule}) \quad \text{becomes}$$

$$\frac{G}{F \vee G} \vee I_R \quad \frac{\frac{G}{F \vee G} \vee I_R}{\neg\neg(F \vee G)} Taut$$

if  $F \vee G$  is needed in the ND-tree      if  $\neg\neg(F \vee G)$  is needed in the ND-tree

**Case 3:** ( $\alpha$ -rule)

$$\frac{\neg(F \Rightarrow G)}{F} \quad (\alpha\text{-rule}) \quad \text{becomes}$$

$$\frac{\frac{\neg F}{\neg F \vee G} \vee I_L}{F \Rightarrow G} Taut \quad \frac{\frac{\frac{\neg F}{\neg F \vee G} \vee I_L}{F \Rightarrow G} Taut}{\neg\neg(F \Rightarrow G)} Taut$$

if  $F \Rightarrow G$  is needed in the ND-tree      if  $\neg\neg(F \Rightarrow G)$  is needed in the ND-tree

$$\frac{\neg(F \Rightarrow G)}{\neg G} \quad (\alpha\text{-rule}) \quad \text{becomes}$$

$$\frac{\frac{G}{\neg F \vee G} \vee I_R}{F \Rightarrow G} Taut \quad \frac{\frac{\frac{G}{\neg F \vee G} \vee I_R}{F \Rightarrow G} Taut}{\neg\neg(F \Rightarrow G)} Taut$$

if  $F \Rightarrow G$  is needed in the ND-tree      if  $\neg\neg(F \Rightarrow G)$  is needed in the ND-tree

**Case 4:** ( $\beta$ -rule)

Needed formula in ND-tree is negation of the formula in the derivation tree.

$$\frac{F \vee G}{F} \quad (\beta\text{-rule}) \quad \text{becomes} \quad \frac{\frac{\neg F \quad \neg G}{\neg F \wedge \neg G} \wedge I}{\neg(F \vee G)} Taut$$

**Case 5:** ( $\beta$ -rule)

Needed formula in ND-tree is negation of the formula in the derivation tree.

$$\frac{F \Rightarrow G}{\neg F} \quad (\beta\text{-rule}) \quad \text{becomes} \quad \frac{\frac{F \quad \neg G}{F \wedge \neg G} \wedge I}{\neg(F \Rightarrow G)} Taut$$

**Case 6:** ( $\beta$ -rule)

$$\begin{array}{c}
 \neg(F \wedge G) \\
 \swarrow \quad \searrow \\
 \neg F \quad \neg G
 \end{array}
 \quad (\beta\text{-rule}) \quad \text{becomes}$$

$$\frac{F \quad G}{F \wedge G} \wedge I \qquad \frac{\frac{F \quad G}{F \wedge G} \wedge I}{\neg\neg(F \wedge G)} Taut$$

if  $(F \wedge G)$  is needed in the ND-tree      if  $\neg\neg(F \wedge G)$  is needed in the ND-tree

**Case 7:** ( $\gamma$ -rule)

Needed formula in ND-tree is negation of the formula in the derivation tree.

$$\begin{array}{c}
 \forall x.F \\
 | \\
 F[x']
 \end{array}
 \quad (\gamma\text{-rule}) \quad \text{becomes} \quad \frac{\frac{\neg F[t]}{\exists x \neg F} \exists I}{\neg \forall x F} Taut$$

If the ground substitution  $\sigma$  contains for  $x'$  the pair  $x' \mapsto t$ .

**Case 8:** ( $\gamma$ -rule)

$$\begin{array}{c}
 \neg \exists x.F \\
 | \\
 \neg F[x']
 \end{array}
 \quad (\gamma\text{-rule}) \quad \text{becomes}$$

$$\frac{F[t]}{\exists x F} \exists I \qquad \frac{\frac{F[t]}{\exists x F} \exists I}{\neg\neg \exists x F} Taut$$

if  $\exists x.F$  is needed in the ND-tree      if  $\neg\neg \exists x.F$  is needed in the ND-tree

If the ground substitution  $\sigma$  contains for  $x'$  the pair  $x' \mapsto t$ .

**Case 9:** ( $\neg\neg$ -rule)

$$\begin{array}{c}
 \neg\neg F \\
 | \\
 F
 \end{array}
 \quad (\neg\neg\text{-rule}) \quad \text{becomes}$$

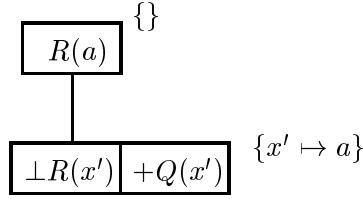
$$\varepsilon \qquad \frac{\neg F}{\neg\neg\neg F} Taut$$

if  $\neg F$  is needed in the ND-tree      if  $\neg\neg\neg F$  is needed in the ND-tree

Thereby  $\varepsilon$  describes the empty operation.

$$\begin{array}{c}
\forall x.((R(x) \vee P(x)) \Rightarrow Q(x)) \\
\mid \quad (\gamma\text{-rule}) \\
(R(x') \vee P(x')) \Rightarrow Q(x') \\
\swarrow \quad (\beta\text{-rule}) \quad \searrow \\
\neg(R(x') \vee P(x')) \quad Q(x') \\
\mid \quad (\alpha\text{-rule}) \\
\neg R(x')
\end{array}$$

which produces clause  $AC = [\neg R(x'), Q(x')]$  which we use in the following UCS:



The branch in the derivation tree from the root  $\forall x.((R(x) \vee P(x)) \Rightarrow Q(x))$  to  $Q(x')$  is the main branch of the derivation tree, the other branch is a side branch. The first step in the algorithm is to initialize the ND-tree with a node with formula  $A$  as the leaf of the main branch.

$$\forall x.((R(x) \vee P(x)) \Rightarrow Q(x))$$

Then we apply stepwise the main branch rules and the side branch rules. The first step in the derivation tree is an application of the  $\gamma$ -rule. The corresponding transfer rule is Case 7 of the main branch rules. We obtain in the ND-tree:

$$\frac{\forall x.((R(x) \vee P(x)) \Rightarrow Q(x))}{(R(a) \vee P(a)) \Rightarrow Q(a)} \forall E$$

The next step in the main branch of the derivation tree is an application of  $\beta$ -rule. We apply the corresponding main branch transfer rule Case 5 and obtain in the ND-tree:

$$\frac{\frac{\forall x.((R(x) \vee P(x)) \Rightarrow Q(x))}{(R(a) \vee P(a)) \Rightarrow Q(a)} \forall E \quad R(a) \vee P(a)}{Q(a)} \Rightarrow E$$

Thus, the transformation of the main branch of the ND-tree is finished. Next, we transform backwardly (starting at the main branch) the side branch of the derivation tree into a side branch in the ND-tree. The first step on the side branch in the derivation tree is an application of  $\alpha$ -rule which is applied to receive  $\neg R(x')$  from  $\neg(R(x') \vee P(x'))$ . Applying the corresponding side branch transfer rule Case 2 we obtain the following ND-tree:

$$\frac{\frac{\forall x.((R(x) \vee P(x)) \Rightarrow Q(x))}{(R(a) \vee P(a)) \Rightarrow Q(a)} \forall E \quad \frac{R(a)}{R(a) \vee P(a)} \vee I_L}{Q(a)} \Rightarrow E$$

An application of the  $\neg\neg$ -correction rule is not necessary and the transformation process is finished. ■

Before we prove the correctness of the derivation tree transformation algorithm we prove some useful lemmas.

**Lemma C.0.5 (Main Branch Lemma).**

*During the transformation-step the formulas at the end node of the main branch (regarding the root as the end) of the ND-tree and the formulas in the corresponding nodes of the derivation tree are equal modulo the ground substitution  $\sigma$  (they are equal if we apply the ground substitution on the derivation tree).*

**Proof:** Induction over the number  $n$  of applied main branch rules.

**Induction Base Case:**  $n = 0$  So far, we applied no rule. Thus, we are in the situation directly after the initialization step.

Therefore, the formula in the end node (the unique node) of the main branch of the ND-tree is  $A$ ;  $A$  is also the formula of the corresponding node (the root node) of the derivation tree.

**Induction Step:**  $n \rightarrow n+1$  By induction hypothesis the formula after  $n$  rule applications in the ND-tree (we call it  $FND_n$ ) is equal to the corresponding formula in the derivation tree (we call it  $FDT_n$ ) modulo the ground substitution  $\sigma$ .

**Case 1:**  $FND_n = FDT_n = G \wedge H$

In the derivation tree either  $G$  or  $H$  is derived using  $\alpha$ -rule. Hence, in the ND-tree we derive also  $G$  or  $H$  with main branch rule Case 1, respectively, such that  $FND_{n+1} = FDT_{n+1}$ .

**Case 2-9:** Analogous to case 1.

□

**Lemma C.0.6 (Side Branch Lemma).**

*During the transformation-step the formulas of the nodes at the end of the side branches (regarding the leaves as the ends) of the ND-tree and the formulas of the corresponding nodes in the derivation tree are negations of each other (modulo the ground substitution  $\sigma$ ).*

**Proof:** Analogous to the proof of the main branch lemma.

□

**Theorem C.0.7 (Transformation Theorem).**

*Let  $(AC, \{UC_1, \dots, UC_n\}, R_{lit})$  be a UCS satisfying the conditions of Theorem 4.3.3. Let  $A$  be a formula in whose CNF  $AC$  is and let  $T_d$  be the derivation tree of the corresponding clause normalization. Let  $\sigma$  be the ground substitution to  $AC$  and let be  $L_1, \dots, L_n$  the only literals of  $UC_1, \dots, UC_n$ , respectively.*

*Then the derivation tree transformation Algorithm C.0.3 computes from  $T_d$  and the ground substitution  $\sigma$  a ND-tree whose:*

- *main branch starts with  $A$  and ends with  $(R_{lit})\sigma$ ,*
- *side branches start with  $L_1, \dots, L_n$ ,*
- *main branch and side branches satisfy the decompositions and compositions constraints.*

**Proof:** The theorem follows by combining the main branch Lemma C.0.5, the side branch Lemma C.0.6 and the use of the  $\neg\neg$ -correction rule described in the derivation tree transformation algorithm.

□

With this theorem we can give a proof of the the UCS-theorem 4.3.3.

**Proof:**

By applying the derivation tree transformation Algorithm C.0.3 we obtain a ND-tree satisfying by construction the compositions and decompositions constraints since each single main branch rule (side branch rule) adds to the main branch (side branches) only composition (decomposition) ND-rules.

Furthermore, by using the transformation Theorem C.0.7 we have that the main branch starts with the assertion  $A$  and ends with the result  $(R_{lit})\sigma$  and the leaves of the side branches are  $L_1, \dots, L_n$ .

Therefore, we have that the whole ND-tree represents an assertion application of the assertion  $A$  on the premises  $L_1, \dots, L_n$  with result  $(R_{lit})\sigma$ . Hence, we can justify

$$\frac{L_1, \dots, L_n}{(R_{lit})\sigma}$$

by this assertion application of assertion  $A$  on premises  $L_1, \dots, L_n$

□



# Index

- $\Delta$ -relation, 25
- $\delta$ -quantified, 13
- $\gamma$ -quantified, 13
- $\gamma$ -restriction, 38
- acyclic clause graphs, 19
- alternative rules, 41, 45, 119
- application of a literal base case, 28
- assertion clause of UCS, 56
- assertion level, 49
- assertion level ND-proofs, 50
- assertion level steps, 50
- atomic formulas, 11
- backward directed algorithm, 53
- base algorithm for UCS-decomposable graphs, 92
- basic ND-proofs, 50
- basic ND-rules, 50
- binary-linked graph, 67
- clause, 12
- clause graph, 17
- clause nodes, 17
- clause normal form, 13
- closed line, 24
- closed ND-proof, 24
- CNF, 13
- composition rule, 49
- compositions and decompositions constraints, 49
- conclusion of a ND-line, 22
- connected clause graphs, 19
- constant symbol, 11
- contra-directed links, 77, 84
- contra-links, 77
- contra-links free graph, 77
- contradictory literals, 12
- correctly justified, 22
- cycle, 19
- cyclic clause graphs, 19
- decomposition rule, 49
- deduction graph, 19
- derivation trees, 15
- direct-decomposition-method, 105
- double multi-link, 67
- Eigenvariable condition, 34
- empty transformation problem, 26
- end step, 64
- expansion of assertion level steps, 52
- external rule, 119
- first-order signature, 11
- formulas, 11
- forward directed abstraction algorithm, 52
- function symbols, 11
- G-clauses, 74
- generalized ND-proof, 24
- graph conditions, 95
- ground clause, 12
- ground literal, 12
- ground substitutions, 20
- ground terms, 12
- hypotheses of a ND-line, 22
- initial ND-proof, 25
- initial transformation problem, 26
- internal rule, 119
- invariant 1, invariant 2, invariant 3, 26
- L-clause, 74
- link, 17
- link conditions, 17
- liquidation-strategy with the direct-decomposition-method, 105
- liquidation-strategy with the symmetrical-simplification-method, 109

literal, 12  
literal base cases, 28  
literal nodes, 17  
  
main branch of assertion application, 50  
mixed rule, 119  
  
ND-line, 22  
ND-proof, 22  
  
open lines, 24  
  
predicate symbols, 11  
pure literal nodes, 17  
  
refutation graph, 19  
result of an UCS, result literal of an UCS,  
    56  
  
separating links, 19  
separation-strategy with the direct-decomposition-  
    method, 106  
separation-strategy with the symmetrical-  
    simplification-method, 109  
shores of a link, 17  
side branch of an assertion application, 49  
simple multi-link, 67  
single-clause-breaking-algorithm, 128  
single-link, 67  
skolem function symbol, 13  
standard example, 14  
standard rules, 41, 45, 119  
subgraph, 18  
subgroup-criterion, 14  
  
tautology rules, 23  
terms, 11  
trail, 19  
trail to a link, 19  
transformation invariants, 26  
transformation of a literal base case, 28  
transformation problem, 25  
transformation rules, 29  
two steps concept, 54  
  
UCS base case, 96  
UCS-decomposable, 63  
UCS-decomposition, 63  
UCS-decomposition criterion, 90  
  
UCS-replacement, 62  
undirected links, 77  
uniform notation, 12  
unit clause step, 55  
unit clauses of UCS, 56  
unit offering, 84  
  
walk, 18

# Symbolverzeichnis

$T(\Sigma)$ .....	Set of all terms over signature $\Sigma$ .....	11
$APL(\Sigma)$ .....	Set of all atomic first-order formulas over signature $\Sigma$ .....	11
$PL(\Sigma)$ .....	Set of all first-order formulas over signature $\Sigma$ .....	11
$\alpha$ -formulas .....	certain kind of formulas .....	12
$\beta$ -formulas .....	certain kind of formulas .....	13
$\gamma$ -formulas .....	certain kind of formulas .....	13
$\delta$ -formulas .....	certain kind of formulas .....	13
$Hyp$ .....	ND rule .....	23
$TND$ .....	ND rule .....	23
$\Rightarrow I$ .....	ND rule .....	23
$Cases$ .....	ND rule .....	23
$IP_1$ .....	ND rule .....	23
$IP_2$ .....	ND rule .....	23
$\wedge I$ .....	ND rule .....	23
$\wedge E_L$ .....	ND rule .....	23
$\wedge E_R$ .....	ND rule .....	23
$\vee I_L$ .....	ND rule .....	23
$\vee I_R$ .....	ND rule .....	23
$\vee E_L$ .....	ND rule .....	23
$\vee E_R$ .....	ND rule .....	23
$Contradiction$ .....	ND rule .....	23
$Indirect$ .....	ND rule .....	23
$\forall I$ .....	ND rule .....	23
$\forall E$ .....	ND rule .....	23
$Choice$ .....	ND rule .....	23
$\exists I$ .....	ND rule .....	23
$\Rightarrow E$ .....	ND rule .....	23
$MT$ .....	ND rule .....	23

$\neg\neg E$ .....	ND rule .....	23
$Taut$ .....	ND rule .....	23
$Weaken$ .....	ND rule .....	23
$<^*_L$ .....	lesser relation on links .....	67
$\leq_g$ .....	Relation on multi-links .....	76
$TI\wedge$ .....	Transformation Rule .....	120
$TMCases$ .....	Transformation Rule .....	120
$TI\Rightarrow$ .....	Transformation Rule .....	120
$TI\Leftrightarrow$ .....	Transformation Rule .....	120
$TI\neg$ .....	Transformation Rule .....	121
$TE\wedge$ .....	Transformation Rule .....	122
$TE\vee$ .....	Transformation Rule .....	122
$TE\Rightarrow$ .....	Transformation Rule .....	122
$TE\Leftrightarrow$ .....	Transformation Rule .....	122
$TE\forall$ .....	Transformation Rule .....	122
$TE\exists$ .....	Transformation Rule .....	123
$TE\neg$ .....	Transformation Rule .....	123
$TMInf$ .....	Transformation Rule .....	123
$TMContra$ .....	Transformation Rule .....	124
$TM\vee left$ .....	Transformation Rule .....	124
$TM\vee right$ .....	Transformation Rule .....	124

# Bibliography

- [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.
- [Da97] B.I. Dahn and al. Integration of automated and interactive theorem proving in ILF. In *Proceedings of CADE-14*, pages 57–60, 1997.
- [Eis88] Norbert Eisinger. *Completeness, Confluence, and Related Properties of Clause Graph Resolution*. PhD thesis, Universität Kaiserslautern, Kaiserslautern, Germany, 1988.
- [Fit90] Melvin Fitting. *First-Order Logic and Automated Theorem Proving*. Springer, 1990.
- [FK99] A. Franke and M. Kohlhase. MATHWEB, an agent-based communication layer for distributed automated theorem proving. In *Proceedings of CADE-16*, pages 217–221, 1999.
- [Gal86] Jean. H. Gallier. *Logic for Computer Science, Foundations of Automated Theorem Proving*. Harper & Row, 1986.
- [Gen35] Gerhard Gentzen. Untersuchungen über das logische Schließen I. *Math. Zeitschrift*, 39:176–210, 1935.
- [HF96] Xiaorong Huang and Armin Fiedler. Presenting machine-found proofs. In M.A. McRobbie and J.K. Slaney, editors, *Proceedings of the 13th Conference on Automated Deduction*, number 1104 in LNAI, pages 221–225, New Brunswick, NJ, USA, 1996. Springer Verlag.
- [Hor99] H. Horacek. Presenting proofs in a human-oriented way. In *Proceedings of CADE-16*, pages 142–156, 1999.
- [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. *Human Oriented Proof Presentation: A Reconstructive Approach*. PhD thesis, Fachbereich Informatik, Universität des Saarlandes, Saarbrücken, Germany, 1994.
- [Hua94b] Xiaorong Huang. Planning reference choices for argumentative texts. In *Proc. of 7th International Workshop on Natural Language Generation*, pages 145–152, Kennebunkport, Maine, USA, 1994.

Draft  
9/2/2000

- [Hua94c] 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.
- [Hua96a] Xiaorong Huang. *Human Oriented Proof Presentation: A Reconstructive Approach*. Infix, Sankt Augustin, 1996.
- [Hua96b] Xiaorong Huang. Translating machine-generated resolution proofs into nd-proofs at the assertion level. In Norman Foo and Randy Coebel, editors, *Proc. of PRICAI-96*, LNAI 1114, pages 399–410. Springer, 1996.
- [Koh99] Michael Kohlhase. OMDoc: Towards an OPENMATH representation of mathematical documents. Draft available at <http://www.ags.uni-sb.de/~omega/www/projects/openmath/omdoc/omdoc.ps>, 1999.
- [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, Carnegie Mellon University, Pittsburgh, Pennsylvania, USA, 1983.
- [Pfe87] Frank Pfenning. *Proof Transformation in Higher-Order Logic*. PhD thesis, Carnegie Mellon University, 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.
- [Pos99] *POST* syntax. Information available at <http://www.ags.uni-sb.de/~omega/primer/post.html>, 1999.
- [Rob65] J. A. Robinson. A machine-oriented logic based on the resolution principle. *J. of ACM*, 12, 1965.
- [Sho76] Robert E. Shostak. Refutation graphs. *Artificial Intelligence*, 7:51–64, 1976.
- [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.
- [Smu68] Raymund. M. Smullyan. *First-Order Logic*. Springer, New York, 1968.
- [SS97] Geoff Sutcliffe and Christian Suttner. The results of the cade-13 atp system competition. *Journal of Automated Reasoning*, 18(2):259–264, 1997. Special Issue on the CADE-13 Automated Theorem Proving System Competition.