Fachbereich Informatik
Universität Kaiserslautern
Postfach 3049
D-6750 Kaiserslautern

SEKI - REPORT

Presentation of Proofs in an Equational Calculus

Christoph Lingenfelder, Axel Präcklein

SEKI Report SR-90-15 (SFB)

Presentation of Proofs in an Equational Calculus

Christoph Lingenfelder
Fachbereich Informatik
Universität Kaiserslautern
Postfach 3049
D-6750 Kaiserslautern
phone: 49 631 205 3335
email: lingenf@informatik.uni-kl.de

Axel Präcklein
Fachbereich Informatik
Universität Kaiserslautern
Postfach 3049
D-6750 Kaiserslautern
phone: 49 631 205 3344
email: prckln@informatik.uni-kl.de

Abstract

One of the main reasons why computer generated proofs are not widely accepted is often their complexity and incomprehensibility. Especially proofs of mathematical theorems with equations are normally presented in an inadequate and not intuitive way. Often completion and rewrite proofs are only given in the form of a program trace. This is even more of a problem for the presentation of inferences drawn by automated reasoning components in other AI systems.

For first order logic, proof transformation procedures have been designed in order to structure proofs and state them in a formalism that is more familiar to human mathematicians. In this report we present a method to handle equational proofs in such systems. To this end equation solution graphs are introduced to represent paramodulation or rewrite proofs. In the process of transforming these proofs into proofs with equation chains, the inherent structure can also be extracted by exploiting topological properties similar to those of refutation graphs in the pure first order case.

This research was supported by the Deutsche Forschungsgemeinschaft (DFG), SFB 314

Contents

Abstract i

Contents ii

1 Introduction 3

2 Representation of Equality Proofs 5

2.1 General Definitions 5
2.2 Basic Equality Reasoning Procedures 8
2.3 Decomposition Graphs 11
2.4 Equality Chains 14

3 Transformation and Structuring 16

3.1 General Definition 16
3.2 Transformation Rules 18
3.3 A Proof Transformation System for Equality 19
3.4 Example 21

4 Conclusion 30

5 Literature 31

1 Introduction

With the increasing strength of Automated Deduction systems the length and complexity of computer generated proofs has reached a degree where they become almost impossible to understand even for the expert. To add to their incomprehensibility, almost every research group uses its own format and style of stating a proof. This has led to a state where only specialists, and sometimes only specialists in the very method of automated reasoning, are capable to understand and check a proof found by an automated deduction system.

But whenever human beings are addressed, the need of easily understandable and clearly structured arguments becomes apparent. Therefore it is necessary to be able to represent proofs in a more abstract and better structured way. Ideally one would like the proof to be given in natural language, with a large variety of inference rules. As a preliminary step in this direction it seems to be useful to transform the computer generated proof into a proof in a natural deduction system which, although still a system of formal logic, has been devised to approximate as much as possible an intuitive form of reasoning.

The transformation of proofs into a natural deduction formulation has solved some of the problems, see [An80], [Mi83], or [Li86], but by and large the increasing length and complexity of the transformed proofs adds to their incomprehensibility rather than to reduce it. It is therefore paramount to be able to state the proofs in a hierarchically structured way, as mathematicians do, formulating subgoals and lemmata. There has been some success in structuring computer generated proofs, cf. [Li90], [PN90], or [Hu91], but all of these approaches are restricted to logics without equality.

We feel, however, that this is a severe restriction, as equality is essential for any natural coding of mathematical problems and of AI problems in general. Therefore, in this report, we adopt Lingenfelder's approach, so that proofs involving equational reasoning can also be structured automatically.

In chapter 2 the formal basis for this work is defined, especially the different calculi and proof representations: equation solution graphs constructed by equality theorem provers and equality chains as a better understandable form of equality proofs. These equality solution graphs can be the result of paramodulation-based reasoning, but can also easily be constructed from a proof using the Knuth-Bendix completion algorithm [KB70]. The representation of equality proofs as equality chains has been chosen for its naturalness and similarity to the reasoning of

mathematicians. Its role resembles the usage of a system of natural deduction in first order logic, as has been invented by Gerhard Gentzen for its simplicity and systematic use of the connectives.

Chapter 3 develops a basic system of proof transformation covering equality reasoning. In this report only pure equality proofs are handled, but the method fits well into the transformation approach, so that it can also be used in conjunction with proof transformation procedures for first order logic. Furthermore we tackle the task of finding the underlying proof structure. As for the case of first order logic this can be accomplished by exploiting topological properties of the graphs representing the computer generated proofs. Structure can be imposed upon the proofs by introducing lemmata, both to avoid duplication of parts of the proof and to arrange a larger proof in a sequence of subgoals easier to understand.

2 Representation of Equality Proofs

In this chapter we will define the logic and introduce all the necessary definitions for the equality calculi used in this report. Everything is standard first order predicate logic with a built-in equality predicate, however the equality predicate is the only predicate needed. Then the paramodulation rule and the Knuth-Bendix completion procedure are introduced. For the actual proof transformation we define equation solution graphs as a starting point and equality chains as the final form of the proofs. These equation solution graphs are well-suited for our purpose because they reflect the inherent proof structure as do refutation graphs in the case of first order logic. They are also independent of the method by which the proofs were found.

2.1 General Definitions

This section contains the basic definitions of the underlying logic. There are no important differences from the usual way of defining these concepts; similar definitions can for instance be found in [Ga86] or in [Lo78].

2.1-1 Definition: (Signature, Terms)

We define a *signature* Σ=($\mathbb{C}$, $\mathbb{V}$, $\mathbb{F}$) as a triple of a finite set of *constant symbols* $\mathbb{C}$, a countable set of *variable symbols* $\mathbb{V}$, and a finite set $\mathbb{F}$ of *function symbols*. $\mathbb{F}$ is the union of sets $\mathbb{F}_n$ of n-ary function symbols (n = 1, 2, ...); all the $\mathbb{F}_n$ are finite and only finitely many of them are non-empty. Then the set $\mathbb{T}\Sigma$ of *terms* is the smallest set with

i. $\mathbb{V}, \mathbb{C} \subseteq \mathbb{T}\Sigma$

ii. if $f \in \mathbb{F}_n$ and $t_1, t_2, \ldots, t_n \in \mathbb{T}\Sigma$, then $ft_1t_2\ldots t_n \in \mathbb{T}\Sigma$.[1]

If the signature is clear from the context the abbreviation $\mathbb{T}$ is often used instead of $\mathbb{T}\Sigma$. With $\Sigma_{gr} = (\mathbb{C}, \varnothing, \mathbb{F})$ the set of all *ground terms* is $\mathbb{T}\Sigma_{gr}$ or simply $\mathbb{T}_{gr}$.

V(t) is the set of variables occurring in a term t. V(o) is an abbreviation for the set of variables occurring in an arbitrary object o, and the same convention is similarly used for $\mathbb{F}_n$, $\mathbb{F}$, $\mathbb{T}$, and $\mathbb{T}_{gr}$.

[1] In some examples we will also use an infix notation for special two-place function symbols like +.

2.1-2 Definition: (Substitutions)

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

A substitution σ is *idempotent* if $\sigma \circ \sigma = \sigma$. This is equivalent to the requirement that none of the variables of its domain occurs in any of the terms of its codomain, cf. [He87]. In this report all substitutions will be idempotent. If a substitution maps into $\mathbb{T}_{gr}$ it is called a *ground substitution,* if it is a bijection and maps into $\mathbb{V}$ it is called a *renaming*.

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

2.1-3 Definition: (Formulae)

We introduce the set $\mathbb{P} = \bigcup_{0 \leq n} \mathbb{P}_n$ consisting of finite sets of n-ary *predicate symbols* $(n = 0, 1, \ldots)$. There are two special zero-place predicate symbols, *TRUE* (written $\top$) and *FALSE* (written $\bot$), $\top, \bot \in \mathbb{P}_0$, and a binary predicate symbol EQUAL (written $=$), $= \in \mathbb{P}_2$. The objects of the form $Pt_1t_2 \ldots t_n$ with $P \in \mathbb{P}_n$ and $t_1, t_2, \ldots, t_n \in \mathbb{T}$ constitute the set $\mathbb{A}$ of *atoms.*

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

(a) Unary connective	$\neg$	*negation sign*
(b) Binary connectives	$\wedge$	*conjunction sign*
	$\vee$	*disjunction sign*
	$\Rightarrow$	*implication sign*
(c) Quantifiers	$\forall$	*universal quantifier*
	$\exists$	*existential quantifier*
(d) Structuring Signs	(	*opening parenthesis*
	)	*closing parenthesis*

The set Φ of *formulae* of First Order Predicate Logic is now defined as the smallest set with:

(i) $\mathbb{A} \subseteq \Phi$

(ii) If $F, G \in \Phi$, then $(F \wedge G)$, $(F \vee G)$, and $(F \Rightarrow G)$ are all in Φ.

(iii) If $F \in \Phi$ and $x \in \mathbb{V}$, then $\neg F$, $(\forall x\ F)$, and $(\exists x\ F)$ are all in Φ. ♦

In this report we consider purely equational proofs, that is, all formulae are of the type $e_1 \wedge ... \wedge e_n \Rightarrow e_{n+1}$, where all e_i are equations. An atom $= s\, t$ is normally written in infix form, $s = t$.

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

In order to establish a well-defined connection between the original formula to be proved and the term nodes and equations in the proof (when it is represented as an equation solution graph), we need a relation between the elements of the graph (term nodes and equations) and the terms occurring in the original formula. The following definitions are made in order to formalize this correspondence. Similarly, to describe equational transformations, we must be able to refer to subterms within terms or atomic formulae.

2.1-4 Definition: (Subformulae, Subterms)

For any formula F, atom A, or term t, we define the sets S(F) of *subformulae* of F and T(A) and T(t) of *subterms* of A and t as follows:

(σ_1) If $F \in \mathbb{A}$, then $S(F)=\{F\}$.

(σ_2) If F is of the form $G \wedge H$, $G \vee H$, or $G \Rightarrow H$, then $S(F)=\{F\} \cup S(G) \cup S(H)$.
G and H are called *immediate subformulae* of F.

(σ_3) If F is of the form $\neg G$, $\forall x\, G$, or $\exists x\, G$, then $S(F)=\{F\} \cup S(G)$.
In these cases G is the only *immediate subformula* of F.

(τ_1) If $t \in \mathbb{C} \cup \mathbb{V}$, then $T(t)=\{t\}$.

(τ_2) If t is of the form $f t_1 t_2 \ldots t_n$ with $f \in \mathbb{F}_n$ and $t_1, t_2, \ldots, t_n \in \mathbb{T}$, then $T(t)=\bigcup_{i \leq n} T(t_i) \cup \{t\}$.

(τ_3) If A is of the form $P t_1 t_2 \ldots t_n$ with $P \in \mathbb{P}_n$ and $t_1, t_2, \ldots, t_n \in \mathbb{T}$, then $T(A)=\bigcup_{i \leq n} T(t_i)$.

(τ_4) If $F \in \Phi$, then $T(F)=\bigcup_{A \in S(F) \cap \mathbb{A}} T(A)$.

If s is a subterm of a formula F, $s \in T(F)$, we sometimes write F(s) to denote this fact. An occurrence of F(t) in the same context will then represent the formula where s has been replaced by t.

2.1-5 Definition: (Subterm and Subformula Occurrences)

In order to specify subterm occurrences of a given term $t \in \mathbb{T}$ (or formula occurrences of a formula $F \in \Phi$) we use finite sequences $<k_1.k_2.\ldots.k_n>$ of integers. We define the set of *subterm occurrences* $\Omega(t)$ as follows:

(α) $t<> \in \Omega(t)$
(β) $t<i.k_2.\ldots.k_n> \in \Omega(t)$ if $t_i<k_2.\ldots.k_n> \in \Omega(t_i)$,
where $t = ft_1t_2\ldots t_n$, $f \in \mathbb{F}_n$, and $t_1,t_2,\ldots,t_n \in \mathbb{T}$

Subformula occurrences and subterm occurrences within formulae are defined analogously. $\Omega(F)$ can be viewed as the set of partial paths through the formula tree of F. By this definition a canonical mapping ω from the set of subformula and subterm occurrences into the set of formulae or terms is induced:

$$\omega: \Omega(F) \to S(F) \cup T(F), \qquad \Omega(t) \to T(t)$$

If no confusion is possible we speak of the subterm occurrence o with $\omega(o)=s$ only as an occurrence of s. The subset $\Omega_a(F)$ of $\Omega(F)$ whose elements are mapped to atoms is called the set of *atom occurrences* within the formula F.

2.1-6 Example: (Formula Occurrences)

With $F = (\forall u\, fiuu = e) \wedge (\forall v\, fve = v) \wedge (\forall xy\, Sx \wedge Sy \Rightarrow Sfiyx) \Rightarrow (\forall z\, Sz \Rightarrow Siz)$ $F<1.3.1.1.2>$ denotes the occurrence of Sy within F, and $F<1.1.1.1.1>$ represents the subterm occurrence of iu within F. The set of atom occurrences within F is $\Omega_a(F) = \{F<1.1.1>, F<1.2.1>, F<1.3.1.1.1>, F<1.3.1.1.2>, F<1.3.1.2>, F<2.1.1>, F<2.1.2>\}$.

Note, that in general identical formulae or terms may appear as different occurrences within a given term or formula.

2.2 Basic Equality Reasoning Procedures

2.2-1 Paramodulation Method

Let F(t) be a formula with a subterm t, and e be an equation $t' = s$. e can be *paramodulated into* F(t) if, for a renaming ρ such that F(t) and ρe have no variables in common, there exists a unifier σ of ρt' and t. Then the formula $\sigma F(\rho s)$ is called a *paramodulant* of F(t) and e.

Originally this rule was proved to be sound and complete in combination with resolution and functional reflexive axioms [RW69]; the functional reflexive axioms were later shown to be superfluous by D. Brand [Br75]. In addition, for the subterms t in the above definition no variables must be considered.

Paramodulation is a deduction rule that is applicable "almost everywhere" making search graphs very bushy [Bu83] and so it should only be used if the result is of utmost importance for other arguments in the proof.

2.2-2 Knuth-Bendix Completion

The observation that equations can be "applied" to terms led to a term replacement approach for the treatment of the equality relation, constraining the application of the paramodulation rule drastically. The main idea for an algorithm is to consider the equations as rules that can only be applied in one direction, which is determined by a partial ordering on the set of terms.

A method to decide the equality of two terms under special equality theories can then be obtained by "reducing" the terms to a unique normal form using the directed equations. The set of theory axioms must obey certain conditions, it must be confluent and Noetherian to ensure completeness and termination of the decision procedure. The equations defining the theory must be directable and must have the above properties or it must be possible to add other equations such that the new system is equivalent to the old one and has the desired properties.

The procedure for this purpose developed by D. Knuth and P. Bendix [KB70] is called *completion*. The resulting system of directed equations constitutes a set of rewrite rules.

When computing a normal form of a term all situations where two rules can be applied to derive different successors are "dangerous" because it must be ensured that both cases lead to the same normal form later on. D. Knuth and P. Bendix showed that it is sufficient to consider critical pairs between rules and to add the corresponding equations to ensure this behaviour. Critical pairs can be constructed from two rules or two instances of the same rule if the left hand sides of the rules overlap. This means that some subterm of the left hand side can be unified with the other left hand side. One term of the critical pair is the right hand side of the first rule with the unifier applied to it. For the other term the unifiable subterm in the one left hand side is replaced by the other right hand side and again the unifier is applied to the result.

In principle the Knuth-Bendix completion algorithm works as follows [KB70]: Beginning with a set of undirected equations, an empty set of directed rules, and a reduction ordering, it tries to derive a convergent set of rules from the equations. It applies the following steps until no equations remain: take an equation, apply all rules to the equation, direct the equation according to the given reduction ordering, and put it into the set of rules. Generate all critical pairs, that is, terms for which rule applications overlap, between the new rule and the set of rules and put them into the set of

equations. If this algorithm terminates, it produces a set of rules that can be used to decide the equality of arbitrary terms of the given theory.

A rule is applicable to a term if the left hand side of the rule matches the term or any of its subterms. If a rule is applied to an object with subterms to which it is applicable, then these are replaced by the right hand side of the rule with the matcher applied to it. In the field of Automated Deduction the application of the rules is often called demodulation [Wo67,Wo84], where an ordering for the actual orientation of the equations is selected heuristically in most cases.

G. Peterson [Pe83] was the first to develop a resolution and paramodulation calculus which reduces to the Knuth-Bendix algorithm when only given unit equality axioms and theorems. M. Rusinowitch [HR86, Ru87] extended this work such that only maximal[1] literals of the clauses must be considered for paramodulation.

Both G. Peterson's and M. Rusinowitch' approaches allow only very restricted reduction orderings and only demodulation steps by unit reduction rules. In Hsiang's and Rusinowitch' definition superpositions with the left and right hand side of an equation in a maximal literal are allowed. The problem with ordered paramodulation is that it does not reduce to Knuth-Bendix completion in the case when only unit equations are present. In this case ordered paramodulation is weaker, in the sense that is has a larger search space than completion. H. Zhang and D. Kapur [ZK88] extend it to a larger number of orderings as well as contextual rewriting. But this calculus is not complete in combination with tautology removal. Bachmair and Ganzinger [BG90] propose how the system of inference rules can be repaired such that this incompleteness no longer occurs.

The extensions of completion to the different versions of superposition calculi are very powerful because of the constraints to the paramodulation rule application and the reduction facility. In addition every resolution and paramodulation theorem prover can be controlled in a way to simulate such a calculus so that all the features of the underlying prover remain available in the absence of equations [De88, Pr90]. The more equations occur in a proof the more a mechanism for proof structuring based on resolution and refutation graphs as described in [LP90] is led astray. So we need a special consideration of long sequences of equation applications.

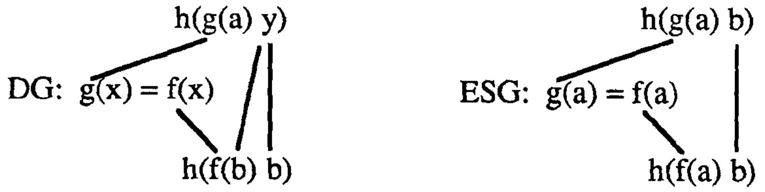
[1] It is straightforward to extend the term ordering needed for the orientation of equations to an ordering on the literals.

2.3 Decomposition Graphs

In order to represent equational proofs and to detect their inherent structure we want to use graphs similar to Bläsius' decomposition graphs. They should be independent of the underlying proof search method, whether it be paramodulation, demodulation as in [Bl86], or completion. If necessary they can be easily constructed from other proof formats whenever all the proof steps are recorded in some sort of a trace.

The main advantage of graphs for the purpose of proof representation is that they only keep the essential information. They abstract from the procedure of searching for a proof to its static skeleton, that is, no order of the proof steps is given, and they omit all unnecessary intermediate results.

The graphs consist basically of two terms s and t and a set of equations, which are connected by links. We will define two types of graphs, Decomposition Graphs (DGs) and Equation Solution Graphs (ESGs). For DGs the terms connected by the links must be unifiable. In ESGs we work in fact with ground instances of equations and terms, and we demand in addition that every node has exactly one link. If these links are constructed in a consistent way the graph constitutes a proof of s=t from the equations occurring in the graph.



Now we proceed to define the notion of decomposition and equation solution graphs rigorously. For once we distinguish between terms and term occurrences that are linked together in the graph. To this end we define term nodes as the basic building blocks of decomposition graphs.

2.3-1 Definition: (Term Nodes)

For an arbitrary, finite set M a set N={(x,t) | x∈M and t∈$\mathbb{T}\Sigma$} is called a set of *term nodes* if (x,s)∈ N and (x,t)∈ N imply s = t. Alternatively a set of term nodes can be described as a set M and a mapping £ from M into $\mathbb{T}\Sigma$.

Given a signature Σ=($\mathbb{C}$, $\mathbb{V}$, $\mathbb{F}$) which defines a set of terms $\mathbb{T}\Sigma$, we use a set $\mathbb{F}^* = \bigcup_{1 \le i \le \infty} \mathbb{F}^*_i$, where the sets $\mathbb{F}^*_i$ are isomorphic to $\mathbb{F}_i$, and a set N of term nodes in

order to define $\Sigma^*=\Sigma(N)=(N, \emptyset, \mathbb{F}^*)$ and the resulting set of terms $\mathbb{T}\Sigma^*$, or simply $\mathbb{T}^*$. The elements of $\mathbb{T}^*$ are called *abstract terms.*

A subset of $\mathbb{T}\Sigma^*$ with the property that every element of N occurs exactly once is called *monoform.* The mapping £ induced by N is extended to the abstract terms in $\mathbb{T}^*$ in the obvious way.

2.3-2 Definition: (Decomposition Graph)

A *decomposition graph (DG)* is a structure $\Gamma=(N, s^*, t^*, E^*, \Lambda)$. N is a set of term nodes, s^* and t^* are elements of $\mathbb{T}^*=\mathbb{T}\Sigma(N)$, $E^*\subseteq\{l^*=r^* \mid l^*, r^*\in\mathbb{T}^*\}$, such that the set $\mathbb{T}^*_m=\{s^*, t^*\}\cup\{w^* \mid w^*=r^* \text{ or } l^*=w^* \in E^*\}$ is a monoform subset of $\mathbb{T}^*$. $\Lambda\subseteq\{[A,B] \mid A, B\in N \text{ where } £A \text{ and } £B \text{ are unifiable}\}$ is a set of *links.*

A decomposition graph $\Gamma'=(N', s'^*, t'^*, E'^*, \Lambda')$ is called a ***subgraph*** of $\Gamma=(N, s^*, t^*, E^*, \Lambda)$ if $N'\subseteq N$, $E'^*\subseteq E^*$, and $\Lambda'\subseteq\Lambda$. s'^* and t'^* can be any subterms of the original $\mathbb{T}^*_m$.

2.3-3 Definition: (Equation Solution Graph)

For a given equational theory $E\subseteq\{l=r \mid l, r\in \mathbb{T}\}$ a decomposition graph $\Gamma=(N, s^*, t^*, E^*, \Lambda)$ is an *equation solution graph (ESG)*

if the function £ maps into $\mathbb{T}_{gr}$,

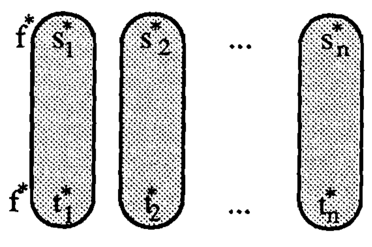
if for every abstract equation $l^*=r^*\in E^*$ there exists an equation $l=r\in E$ such that $£^*l^*=£^*r^*$ is an instance of $l=r$,

and if it has one of the following forms:

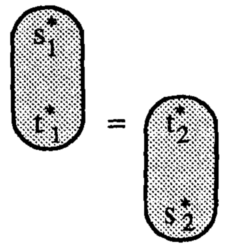
1. Reflexivity
$\Gamma=(\{A, B\}, A, B, \emptyset, [A, B])$ with
$A=(A',t_{gr})$ and $B=(B',t_{gr})$.

$$\begin{tikzcd}[t_{gr}] \arrow[d, dash] \\ [t_{gr}]\end{tikzcd}$$

2. Decomposition
$\Gamma=(\dot{\bigcup}_{1\le i\le n} N_i, f^*s_1^*\ldots s_n^*, f^*t_1^*\ldots t_n^*, \dot{\bigcup}_{1\le i\le n} E_i^*, \dot{\bigcup}_{1\le i\le n} \Lambda_i)$
such that for all i $(N_i, s_i^*, t_i^*, E_i^*, \Lambda_i)$ is a subgraph of Γ. $\dot{\bigcup}$ indicates the union of disjoint sets.



3. Extended Transitivity
$\Gamma = (N_1 \cup N_2, s_1^*, s_2^*,$
$E_1^* \cup E_2^* \cup \{t_1^* = t_2^*\}, \Lambda_1 \cup \Lambda_2)$
such that $\Gamma=(N_1, s_1^*, t_1^*, E_1^*, \Lambda_1)$ and
$\Gamma=(N_2, t_2^*, s_2^*, E_2^*, \Lambda_2)$ are subgraphs
of Γ.



Such an equation solution graph represents a proof of the equation $£^*s^* = £^*t^*$ with respect to the equational theory E. If E is given as a set $\{e_1, \ldots, e_n\}$ of equations this can be considered a proof of the formula $e_1 \wedge \ldots \wedge e_n \Rightarrow £^*s^* = £^*t^*$.

In order to represent the proof of a universally quantified equation the notion of ESGs is extended to allow variables, but these are considered constants, that is, any given ground substitution for the variables must transform the extended ESG into a ground ESG.

2.3-4 Example: (Equation Solution Graph)

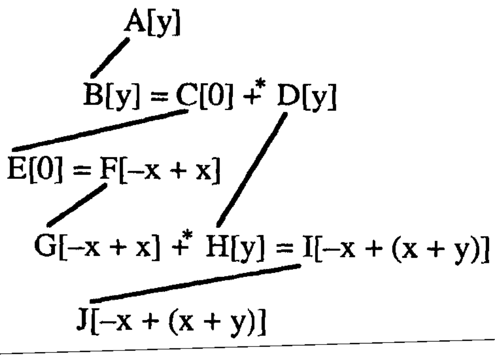
Let A = (A', y), B = (B', y), C = (C', 0), D = (D', y), E = (E', 0), F = (F', $-x+x$), G = (G', $-x+x$), H = (H', y), I = (I', $-x+(x+y)$), and J = (J', $-x+x$) be a set of term nodes. Note that F' and G' are mapped onto the same term $-x+x$.

Then ({A, B, C, D, E, F, G, H, I, J}, A, J,
{B=C+*D, E=F, G+*H=I},
{[A, B], [C, E], [D, H], [F, G], [I, J]})

is the ESG depicted below. In a more intuitive version the terms of the term nodes are integrated into the notation of the nodes:

({A[y], B[y], C[0], D[y], E[0], F[–x+x],
G[–x+x], H[y], I[–x+x], J[–x+x]}, A, J,
{B=C+*D, E=F, G+*H=I},
{[A, B], [C, E], [D, H], [F, G], [I, J]})

And now this can be represented by the following figure:



2.4 Equality Chains

2.4-1 Definition: (Equation Chains)

An *equation chain* relative to an equational theory E is a finite sequence of terms $(t_1, ..., t_n)$ such that for each i with $2 \leq i \leq n$ there exist subterm occurrences $t_{i-1}<k_{i-1}>$, substitutions σ_i, and equations $l_i = r_i$ in E with $\omega(t_{i-1}<k_{i-1}>) = \sigma_i(l_i)$ or $\omega(t_{i-1}<k_{i-1}>) = \sigma_i(r_i)$. t_i is constructed from its predecessor by replacing $\omega(t_{i-1}<k_{i-1}>)$ by $\sigma(r_i)$ or $\sigma(l_i)$, respectively. The sequence $(l_2 = r_2, \ldots, l_n = r_n)$ is called a *justification* of the equation chain.

Equation chains are written $t_1 = t_2 = \ldots = t_n$ to indicate that all the terms in an equation chain are equal with respect to the theory E.

2.4-2 Example: (Equation Chain)

$(y, 0+y, (-x+x)+y, -x+(x+y))$ is an equation chain for $E=\{x=0+x,\ 0=-x+x,\ x+(y+z)=(x+y)+z\}$.

2.4-3 Definition: (Equation Proof)

An equation *proof line* consists of

i. an equation chain γ: $t_1 = t_2 = \ldots = t_n$; its *conclusion* is $t_1 = t_n$, and

ii. a justification for the chain.

A finite sequence S of proof lines is an *equation proof (EP)* of an equation e from equational assumptions E, if

iii. e is the conclusion of the last line of S,

iv. in each line, γ is an equation chain relative to $E \cup \{s=t \mid s=t$ is the conclusion of a previous proof line in S$\}$. ♦

In the following example we use an intuitive way to write proof lines with equation chain and justification interleaving. A more formal notation is $t_1 = t_2 = \ldots = t_n\ (j_2, j_3, \ldots, j_n)$.

2.4-4 Example: (Equation Proof)

As an example let us prove that $A1: x=0+x$, $A2: 0=-x+x$, and $A3: x+(y+z)=(x+y)+z$ imply $x=x+0$. This is, that left and right inverses are the same in groups:

$$
\begin{array}{llll}
(1) & y & = 0+y & \text{A1} \\
 & & = (-x+x)+y & \text{A2} \\
 & & = -x+(x+y) & \text{A3} \\
(2) & x & = --x+(-x+x) & 1 \\
(3) & x & = --x+(-x+x) & 2 \\
 & & = --x+0 & \text{A2} \\
(4) & --x & = ----x+0 & 3 \\
(5) & 0 & = ---x+(--x+0) & 1 \\
(6) & \multicolumn{2}{l}{----x+(---x+x) = x} & 1 \\
(7) & x & = --x+0 & 3 \\
 & & = (----x+0)+0 & 4 \\
 & & = (----x+(---x+(--x+0)))+0 & 5 \\
 & & = (----x+(---x+x))+0 & 3 \\
 & & = x+0 & 6
\end{array}
$$

Chains of length two are special, because they do not really derive anything "new", but they are simply instances of previously generated equations. Often they can be omitted altogether, and we will only insert them if the instance in case is rather complicated.

3 Transformation and Structuring

The construction of equation proofs, by humans and computers alike, is conducted in single steps. To prove any valid equation e one always starts with a line s = t without correct justification. Such a line is obviously no proof, because it is not correctly justified; it is assumed, however, that a proof of the equation is known, for instance in form of an ESG. Now the proof is constructed by deriving subgoals until it is completed. In the intermediate states one may find completed subproofs, but also others that are not yet done. To formalize the procedure of the search for an equation chain proof, we use Generalized Equation Proofs in analogy to Lingenfelder's generalized natural deduction proofs [Li90] or Andrews' proof outlines for natural deduction proofs.

3.1 General Definition

3.1-1 Definition: (Generalized Equation Proof)

A finite sequence S of proof lines is called a *Generalized Equation Proof (GEP)* of an equation e, if

i. e is the conclusion of the last line of S,
ii. each line contains an equation chain relative to $E \cup E' \cup \{s=t \mid s=t$ is the conclusion of a previous proof line in $S\}$.

This allows lines not correctly justified within the calculus, but it is assumed that these lines are "sound", in the sense that the conclusions are valid if the justifying equations E' are sound. Lines not correctly justified within the calculus are called *external* lines, lines justified within the calculus are called *internal.* When no external lines are present in a GEP, it is a normal EP.

A GEP consisting of just one external line with equation chain e justified by $e \in E'$ is called a *trivial GEP* for e. ♦

When an equation in a justification is represented by an ESG we use the name of the graph in the justification. Similarly the name or number of a proof line in a justification replaces this line's conclusion.

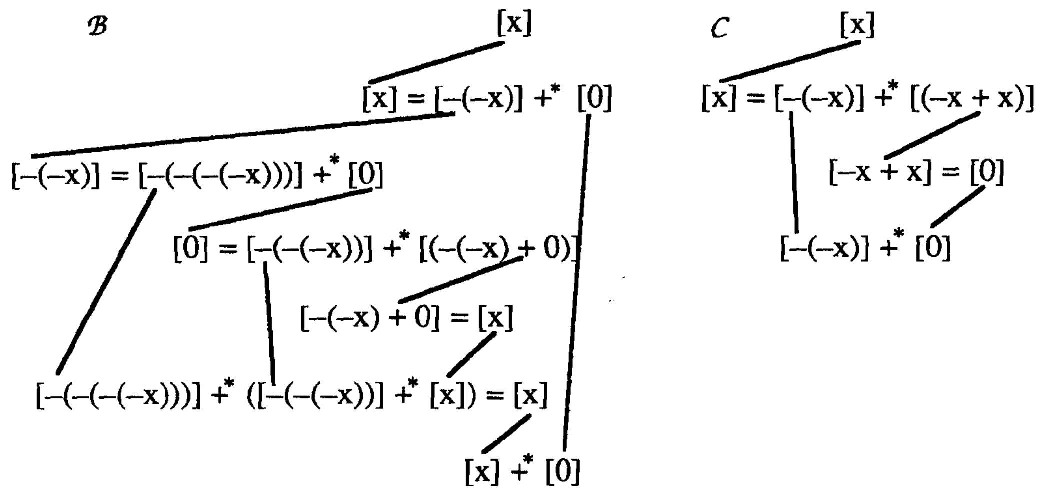
3.1-2 Example: (GEP)

In this example we give first the initial GEP of the formula $F: x = 0+x \wedge 0 = -x+x \wedge x+(y+z) = (x+y)+z \Rightarrow x=x+0$ corresponding to example 2.4-4, with a constant

symbol 0 and function symbols + and –. Infix notation is employed for + and also for the corresponding $+^*$.

(7) $x = x + 0$ $\mathcal{B}$

For a proof of $\mathcal{B}$ several equation solution graphs were derived by the system, the following two and the graph $\mathcal{A}$ of example 2.3-4.



B
[x]
[x] = [–(–x)] +* [0]
[–(–x)] = [–(–(–(–x)))] +* [0]
[0] = [–(–(–x))] +* [(–(–x) + 0)]
[–(–x) + 0] = [x]
[–(–(–(–x)))] +* ([–(–(–x))] +* [x]) = [x]
[x] +* [0]
C
[x]
[x] = [–(–x)] +* [(–x + x)]
[–x + x] = [0]
[–(–x)] +* [0]

In order to find an equation proof of a formula F, a finite sequence of generalized EPs can be constructed whose first element is a trivial GEP for e, and whose justification is an ESG. To be able to generate such a sequence of GEPs it is necessary to describe the rules by which a GEP is constructed from its predecessor in the sequence. In the following example one such transition between two consecutive GEPs is shown.

3.1-3 Example: (Proof Transformation)

During the proof transformation process the following GEP has been arrived at:

(3)	x	= ––x+0	$\mathcal{C}$
(4)	––x	= ––––x+0	$\mathcal{C}$
(5)	0	= –––x+(––x+0)	$\mathcal{A}$
(6)	––––x+(–––x+x) = x		$\mathcal{A}$
(7)	x	= ––x+0	3
		= (––––x+0)+0	4
		= (––––x+(–––x+(––x+0)))+0	5
		= (––––x+(–––x+x))+0	3
		= x+0	6

Now the proof of the graph $\mathcal{A}$ must be expanded only once. This leads to the next GEP.

$$\begin{array}{lllr}
(1) & y & = -x+(x+y) & \mathcal{A} \\
(3) & x & = --x+0 & \mathcal{C} \\
(4) & --x & = ----x+0 & \mathcal{C} \\
(5) & 0 & = ---x+(--x+0) & 1 \\
(6) & \multicolumn{2}{l}{----x+(---x+x) = x} & 1 \\
(7) & x & = --x+0 & 3 \\
 & & = (----x+0)+0 & 4 \\
 & & = (----x+(---x+(--x+0)))+0 & 5 \\
 & & = (----x+(---x+x))+0 & 3 \\
 & & = x+0 & 6
\end{array}$$

In the following section we will give a formal account of some of these transition rules. In their description α and β are used as labels for the lines and the justifications $\mathcal{A}$, $\mathcal{A}_1$, $\mathcal{A}_2$, ..., $\mathcal{A}_n$ represent proofs of the respective lines. For all these rules one must make sure that the proofs $\mathcal{A}_1$, $\mathcal{A}_2$, ..., $\mathcal{A}_n$ can be constructed from $\mathcal{A}$ or are otherwise known. How this can be done if the proof is given in form of an equation solution graph will be shown later, when the automatic proof transformation procedure is described.

3.2 Transformation Rules

In the description of the transformation rules the lines on the left hand side of the arrow ($\rightarrow$) are replaced by those on the right hand side in the next generalized EP of the sequence.

Instantiation:

$$(\alpha)\ ..t{=}s..\ \ (..e..) \quad \rightarrow \quad \begin{cases} (\alpha) & t{=}s\ (e) \\ (\beta) & ..t{=}s..\ \ (..\alpha..) \end{cases}$$

if t=s is an instance of the axiom $e \in E$ modulo symmetry of the equality predicate.

Elimination:

$$(\alpha)\ ..t{=}t..\ \ (..\mathcal{A}..) \quad \rightarrow \quad (\alpha)\ \ ..t..\ \ (....)$$

Insert:

$(\alpha)\ ..s_1=s_2..\ (..\mathcal{A}..) \quad \rightarrow \quad (\alpha)\ ..s_1=t_1=t_2=s_2..\ (..\mathcal{A}_1,t_1=t_2,\mathcal{A}_2..)$

if $\mathcal{A}$ is an ESG constructed corresponding to case 3 of definition 2.3-3 and the terms of the splitting equation $t_1=t_2$ represent instances of the top level terms of an ESG.

Assumption:

$(\alpha)\ ..s_1=s_2..\ (..\mathcal{A}..) \quad \rightarrow \quad (\alpha)\ ..s_1=t_1=t_2=s_2..\ (..\mathcal{A}_1,e,\mathcal{A}_2..)$

if $\mathcal{A}$ is an ESG constructed corresponding to case 3 of definition 2.3-3 and the splitting equation $t_1=t_2$ is an instance of an axiom e of the set of input equations E.

Decomposition:

$$(\alpha)\ ..ft_1...t_n=fs_1...s_n..\ (..\mathcal{A}..) \quad \rightarrow \quad \left\{ \begin{array}{l} (\alpha)\ \ ..ft_1...t_n=fs_1t_2...t_n=...=fs_1...s_n.. \\ \qquad\qquad\qquad\qquad (..,\mathcal{A}_1,...,\mathcal{A}_n,..) \end{array} \right.$$

if $\mathcal{A}$ is an ESG constructed corresponding to case 2 of definition 2.3-3.

Lemma:

$$(\beta)\ ..t'=s'..\ (..\mathcal{A}..) \quad \rightarrow \quad \left\{ \begin{array}{l} (\alpha)\ \ t=s\ (\mathcal{A}) \\ (\beta)\ \ ..t'=s'..\ (..\alpha..) \end{array} \right.$$

If t'=s' is an instance of t=s, which is the pair of top level terms of $\mathcal{A}$.

3.3 A Proof Transformation System for Equality

The set of transformation rules defined in the previous section constitutes a proof system for equational proofs. This means that for any valid equation e, there is a finite sequence of GEPs starting with e and ending with an EP for e. Every element in this sequence can be constructed from its predecessor by application of one of the transition rules.

This system could be used as a proof checker, the user choosing from a menu of applicable rules, and the system correctly applying them. Some of the rules (called "automatic") are always useful and can automatically be applied by the system. For the others (called "user guided", normally **Instantiation** and **Lemma**) the user must make a decision or provide more information. So the system can actually do much more by preselecting a subset of the transformation rules and giving the user a much smaller choice of rules.

Now the strategy for a semiautomatic proof system can be described by the following algorithm:

3.3-1 Algorithm: (Basic Proof Transformation)

i. Start with a GEP e ($\mathcal{A}$).

ii. Check whether the proof is already completed, in which case the GEP is returned as final proof.

iii. Decide whether to apply any of the applicable "user guided" rules. If so, do it .

iv. Apply "automatic" rules until some of the "user guided" rules are newly applicable, then go to ii.

The decision about the application of any "user guided" transformation rules can be resolved interactively by the user or according to appropriate heuristics, making use of the information in a given proof, for instance a previously computed equation solution graph. The selection between different rules that might be applicable is guided by the equation solution graph representing the proof of the external lines in a GEP.

In [Li90] it is shown how a proof represented as a refutation graph without equality can guide the "search" for a natural deduction proof. In this context, search means to transform the given, graph-represented proof into the natural deduction calculus, rather than to find an original proof.

We assume that a proof of an equation e has already been found by an automated deduction system. We will further assume that this proof is represented as an equation solution graph Γ. Actually the proof is often represented by more than one graph – resulting from its construction from paramodulation proofs. One of them represents the main line of the argument, the others prove equations that are used together with the original equational theory E in the main proof. These auxiliary graphs can be considered lemmata in the proof, especially if they are used more than once in different instantiations.

Up to now, the main incentive for the introduction of a lemma was to avoid an unnecessary duplication of parts of the proof. But this is not the only reason why mathematicians use lemmata. In many cases they are used purely to structure the proof, so that the idea behind a proof becomes better visible.

In an automatic proof transformation the difficulty is obviously to find meaningful lemmata. And it is here again that the topological structure of the equation solution

graph may successfully be exploited. The task is to find parts of the equation solution graph that are sufficiently complex in order to justify the introduction of a lemma.

Such lemma graphs can be isolated in two different settings, corresponding to the recursive cases of definition 2.3-3. For decomposition, any of the subgraphs proving the equality of two subterms may become a lemma graph. If the graph contains a separating equation, as in the case of extended transitivity, any of the two subgraphs can be considered a lemma.

In any of these cases a lemma is only introduced if the corresponding subgraph is not trivial. An exact and meaningful definition of triviality is very difficult in this context, so the choice must be made using a heuristic approach:

We consider a subgraph as non-trivial if it contains the instance of an equation that is used in a different instantiation elsewhere in the graph.

Another criterion is the distinction between completion and rewriting steps, which can be made if the underlying paramodulation rule discriminates these steps according to the Knuth-Bendix algorithm. Completion steps are more important and substantial while rewriting steps can usually be considered a calculation rather than a proof. So a subgraph containing only rewriting steps can be treated as trivial.

And of course the absolute and the relative size of the subgraph, the depth of the terms involved and other syntactical criteria should be taken into account.

3.4 Example

As an example we choose a standard text book example of linear algebra, the rule of cancellation in a group. In first order notation with equality this law is represented by the following formula:

$$\begin{aligned} &\forall x,y,z\colon x = x+0 \wedge x = 0+x \wedge 0 = -x+x \wedge 0 = x+-x \wedge x+(y+z) = (x+y)+z \\ &\qquad\Rightarrow \quad \forall x,y,a\ a+x = a+y \ \Rightarrow x = y \\ &\qquad\quad\ \wedge\ \forall x,y,b\ x+b = y+b \ \Rightarrow x = y \end{aligned}$$

An automatically generated paramodulation[1] proof of this formula is shown below. a and b as well as the different occurrences of x and y become Skolem constants in the normal form, and therefore also in the equation solution graph. They are named c1 through c6 in the sequel.

[1] The final resolution step is only needed for technical reasons.

```
Axioms:    All x,y,z + (+ (x y) z) = + (x + (y z))
           All x + (0 x) = x
           All x + (x 0) = x
           All x + (- (x) x) = 0
           All x + (x - (x)) = 0

Theorems:  All x1,x2,a + (a x1) = + (a x2) Impl x1 = x2
           All y1,y2,b + (y1 b) = + (y2 b) Impl y1 = y2

Set of Axiom Clauses Resulting from Normalization[1]
          A1:    All x:Any =(x x)
          A2:    All x,y,z:Any =(+(+(z y) x) +(z +(y x)))
          A3:    All x:Any =(+(0 x) x)
          A4:    All x:Any =(+(x 0) x)
          A5:    All x:Any =(+(-(x) x) 0)
          A6:    All x:Any =(+(x -(x)) 0)
Set of Theorem Clauses Resulting from Normalization and Splitting
Splitpart 1:[2]
          T7:    =(+(c1 c2) +(c1 c3))
          T8:    ¬ =(c2 c3)
Splitpart 2:
          T9:    =(+(c5 c4) +(c6 c4))
          T10:   ¬ =(c5 c6)

Refutation of Splitpart 1:
A5,1 & A2,1   →  P2:  All x,y:Any =(+(0 y) +(-(x) +(x y)))
P2,1 & A3     →  Rw3: All x,y:Any =(y +(-(x) +(x y)))
T7,1 & Rw3,1  →  P4:  =(c2 +(-(c1) +(c1 c3)))
P4,1 & Rw3    →  Rw5: =(c2 c3)
Rw5,1 & T8,1  →  R6:  □
```

[1] In the proof all the clauses have names consisting of an abbreviation for the inference rule used and a number as a running index. A means axiom, T theorem, R resolvent, P paramodulant, and Rw rewrite.

[2] The system automatically divides the proof into separate parts for the two conjuncts of the theorem.

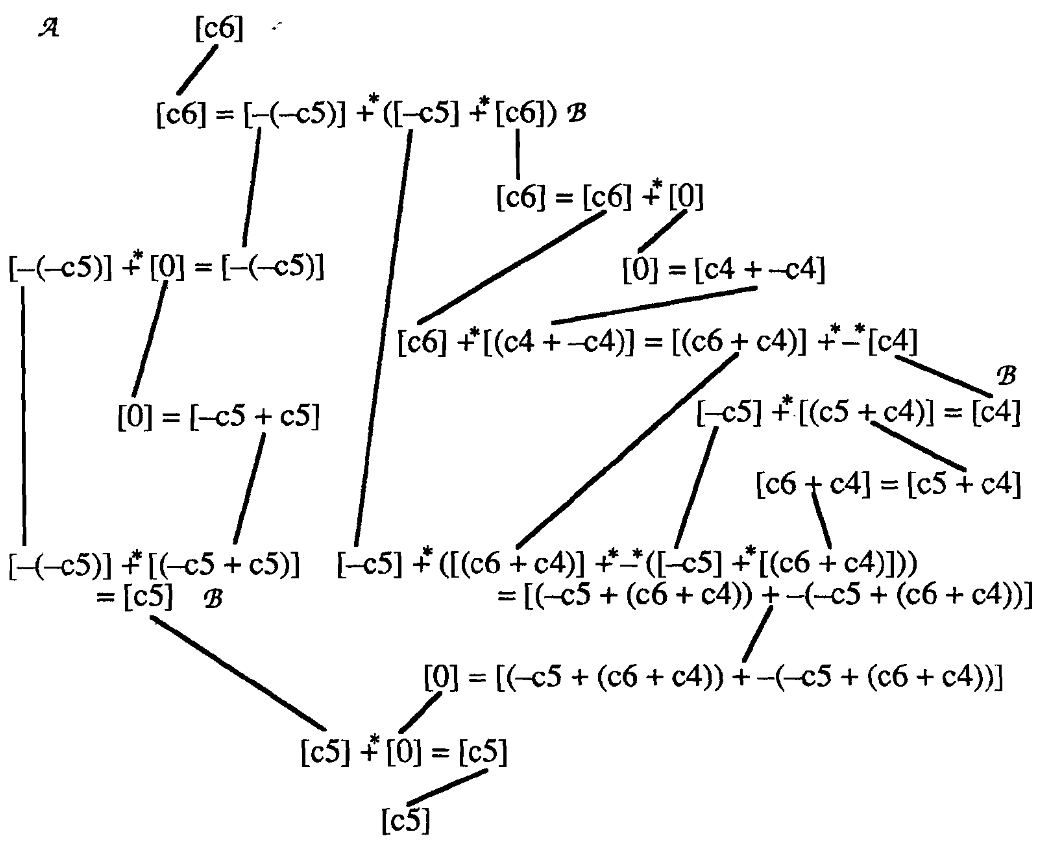
```
Refutation of Splitpart 2:[1]
A5,1 & A2,1    →  P8:   All x,y:Any =(+(0 y) +(-(x) +(x y)))
P8,1 & A3      →  Rw9:  All x,y:Any =(y +(-(x) +(x y)))
A5,1 & Rw9,1   →  P10:  All x:Any =(x +(-(-(x)) 0))
P10,1 & A4     →  Rw11: All x:Any =(x -(-(x)))
T9,1 & Rw9,1   →  P14:  =(c4 +(-(c5) +(c6 c4)))
A6,1 & A2,1    →  P15:  All x,y:Any =(0 +(y +(x -(+(y x)))))
P14,1 & P15,1  →  P16:  =(0 +(-(c5) +(+(c6 c4) -(c4))))
P16,1 & A2     →  Rw17: =(0 +(-(c5) +(c6 +(c4 -(c4)))))
Rw17,1 & A6    →  Rw18: =(0 +(-(c5) +(c6 0)))
Rw18,1 & A4    →  Rw19: =(0 +(-(c5) c6))
Rw19,1 & Rw9,1→  P20:  =(c6 +(-(-(c5)) 0))
P20,1 & Rw11   →  Rw21: =(c6 +(c5 0))
Rw21,1 & A4    →  Rw22: =(c6 c5)
Rw22,1 & T10,1→  R23:  □
```

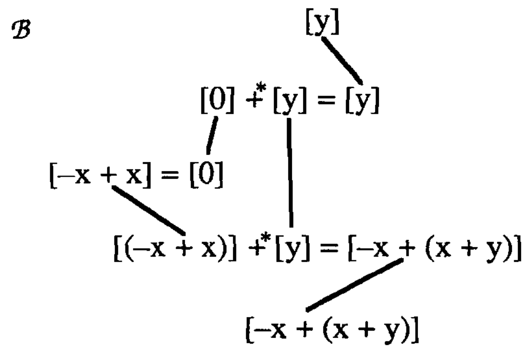
In the example we consider the proof of the second (more complicated) splitpart.[2] It is first translated into an equation solution graph. The nodes labeled with $\mathcal{B}$ in the upper graph are instances of the equation solution graph $\mathcal{B}$ below; a complete equation solution graph can be obtained by inserting three copies of the equation solution graph for `Rw9: All x,y:Any =(y +(-(x) +(x y)))` with corresponding instantiation.

We begin the transformation with the graphs $\mathcal{A}$ and $\mathcal{B}$, where $\mathcal{A}$ represents the main proof, and the trivial GEP (13) c6 = c5 $\mathcal{A}$.

[1] The proofs for the splitparts differ essentially although both parts of the theorem can be proved analogously. The system generates a more complicated proof for the second part, because it is based on the Knuth-Bendix algorithm. The axiom of associativity is directed, and hence can not be used in the direction opposite to that in the first step of splitpart 1.

[2] We only strive to transform the representation of the proof into a more readable format, but do not intend to change the structure from a complicated to a simpler proof. In the actual example, a mechanism to detect analogies between theorems may be adequate to select a shorter proof for the second splitpart.


𝒜
[c6]
[c6] = [–(–c5)] +* ([–c5] +* [c6]) ℬ
[c6] = [c6] +* [0]
[–(–c5)] +* [0] = [–(–c5)]
[0] = [c4 + –c4]
[c6] +*[(c4 + –c4)] = [(c6 + c4)] +*–*[c4]
ℬ
[0] = [–c5 + c5]
[–c5] +* [(c5 + c4)] = [c4]
[c6 + c4] = [c5 + c4]
[–(–c5)] +* [(–c5 + c5)] = [c5] ℬ
[–c5] +* ([(c6 + c4)] +*–*([–c5] +*[(c6 + c4)])) = [(–c5 + (c6 + c4)) + –(–c5 + (c6 + c4))]
[0] = [(–c5 + (c6 + c4)) + –(–c5 + (c6 + c4))]
[c5] +* [0] = [c5]
[c5]



One application of **Assumption** with a graph 𝒜″ and a trivial second graph 𝒞 yields:

(13) c6	=	c5+0	𝒜″
	=	c5	A4
	=	c5	𝒞

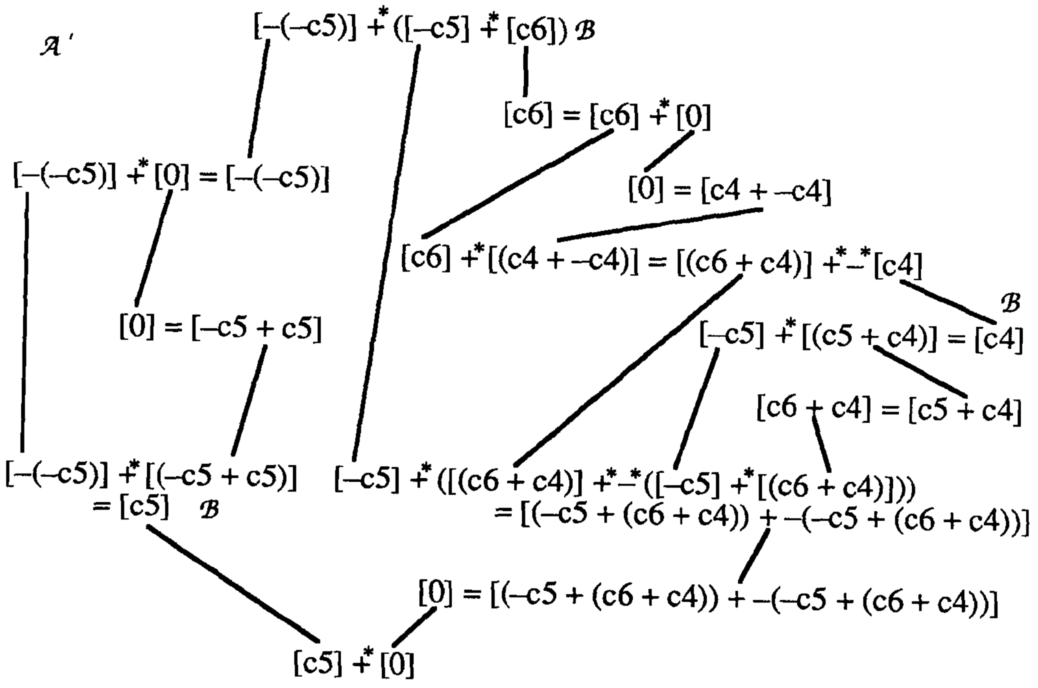
This is simplified by **Elimination**:

(13) c6	=	c5+0	𝒜″
	=	c5	A4

An application of **Insert** with subsequent **Elimination** of the trivial first graph leads to:

(13)	c6	= −−c5+(−c5+c6)	$\mathcal{B}$
		= c5+0	$\mathcal{A}'$
		= c5	A4

with $\mathcal{A}'$:

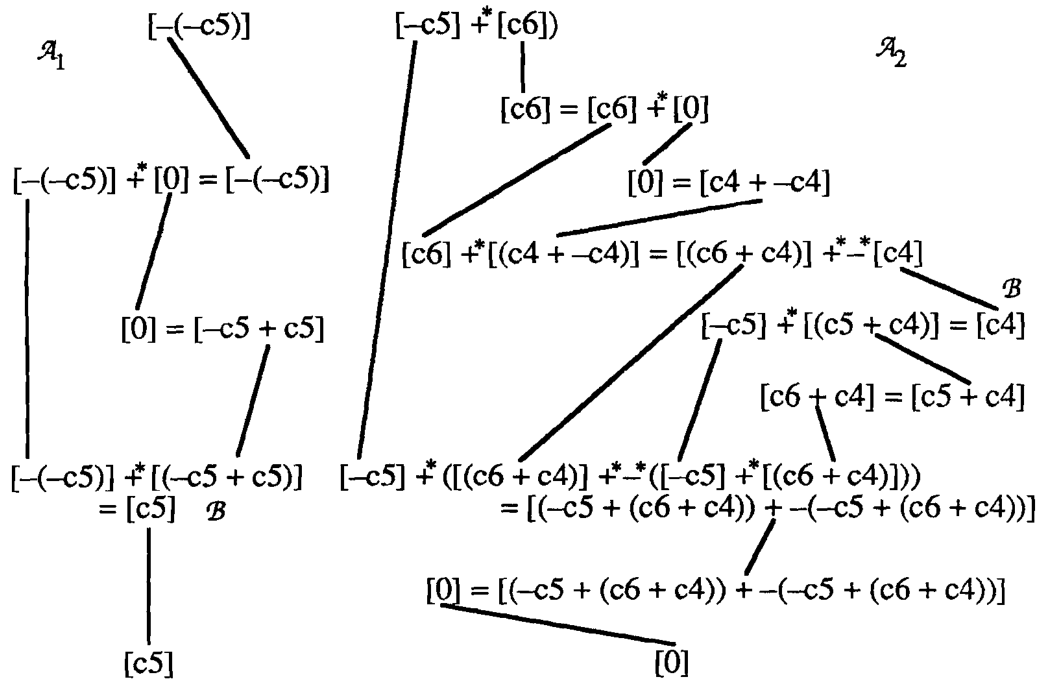


$\mathcal{A}'$
[−(−c5)] +* ([−c5] +* [c6]) $\mathcal{B}$
[c6] = [c6] +* [0]
[−(−c5)] +* [0] = [−(−c5)]
[0] = [c4 + −c4]
[c6] +* [(c4 + −c4)] = [(c6 + c4)] +*−* [c4]
$\mathcal{B}$
[0] = [−c5 + c5]
[−c5] +* [(c5 + c4)] = [c4]
[c6 + c4] = [c5 + c4]
[−(−c5)] +* [(−c5 + c5)] = [c5] $\mathcal{B}$
[−c5] +* ([(c6 + c4)] +*−* ([−c5] +* [(c6 + c4)])) = [(−c5 + (c6 + c4)) + −(−c5 + (c6 + c4))]
[0] = [(−c5 + (c6 + c4)) + −(−c5 + (c6 + c4))]
[c5] +* [0]

Now a **Decomposition** step is possible on the top level terms of $\mathcal{A}'$:

(13)	c6	= −−c5+(−c5+c6)	$\mathcal{B}$
		= c5+(−c5+c6)	$\mathcal{A}_1$
		= c5+0	$\mathcal{A}_2$
		= c5	A4

with $\mathcal{A}_1$ and $\mathcal{A}_2$:



$\mathcal{A}_1$
[–(–c5)]
[–(–c5)] +* [0] = [–(–c5)]
[0] = [–c5 + c5]
[–(–c5)] +* [(–c5 + c5)] = [c5] $\mathcal{B}$
[c5]
$\mathcal{A}_2$
[–c5] +* [c6])
[c6] = [c6] +* [0]
[0] = [c4 + –c4]
[c6] +*[(c4 + –c4)] = [(c6 + c4)] +*–*[c4]
$\mathcal{B}$
[–c5] +* [(c5 + c4)] = [c4]
[c6 + c4] = [c5 + c4]
[–c5] +* ([(c6 + c4)] +*–*([–c5] +*[(c6 + c4)])) = [(–c5 + (c6 + c4)) + –(–c5 + (c6 + c4))]
[0] = [(–c5 + (c6 + c4)) + –(–c5 + (c6 + c4))]
[0]

Both subgraphs of the decomposition are not trivial and so both are used to perform **Lemma** steps:

(4)	––c5	=	c5	$\mathcal{A}_1$
(12)	–c5+c6	=	0	$\mathcal{A}_2$
(13)	c6	=	––c5+(–c5+c6)	$\mathcal{B}$
		=	c5+(–c5+c6)	4
		=	c5+0	12
		=	c5	A4

The first equation of chain (13) is a proper instance of its justifying graph $\mathcal{B}$, which has been proven as a separate ESG. Therefore a **Lemma** step is applied:

(2)	y	=	–x+(x+y)	$\mathcal{B}$
(4)	––c5	=	c5	$\mathcal{A}_1$
(12)	–c5+c6	=	0	$\mathcal{A}_2$
(13)	c6	=	––c5+(–c5+c6)	2
		=	c5+(–c5+c6)	4
		=	c5+0	12
		=	c5	A4

After two **Assumptions** and the corresponding **Eliminations** for the first proof line we have:

(2)	y	= 0+y	A3
		= (–x+x)+y	$\mathcal{B}'$
		= –x+(x+y)	A2
(4)	––c5	= c5	$\mathcal{A}_1$
(12)	–c5+c6	= 0	$\mathcal{A}_2$
(13)	c6	= ––c5+(–c5+c6)	2
		= c5+(–c5+c6)	4
		= c5+0	12
		= c5	A4

with a graph $\mathcal{B}'$:

$$\begin{tikzcd}
\mathcal{B}' & & {[0] \overset{*}{+} [y]} \arrow[dl, dash] \arrow[dd, dash] \\
& {[-x + x] = [0]} \arrow[dr, dash] & \\
& & {[(-x + x)] \overset{*}{+} [y]}
\end{tikzcd}$$

Via one **Decomposition**, one **Assumption**, and the corresponding **Eliminations** we get:

(2)	y	= 0+y	A3
		= (–x+x)+y	A5
		= –x+(x+y)	A2
(4)	––c5	= c5	$\mathcal{A}_1$
(12)	–c5+c6	= 0	$\mathcal{A}_2$
(13)	c6	= ––c5+(–c5+c6)	2
		= c5+(–c5+c6)	4
		= c5+0	12
		= c5	A4

As the application of Axiom A5 is somewhat hidden we apply an **Instantiation** step:

(1)	0	= –x+x	A5
(2)	y	= 0+y	A3
		= (–x+x)+y	1
		= –x+(x+y)	A2
(4)	––c5	= c5	$\mathcal{A}_1$
(12)	–c5+c6	= 0	$\mathcal{A}_2$
(13)	c6	= ––c5+(–c5+c6)	2
		= c5+(–c5+c6)	4
		= c5+0	12
		= c5	A4

After further similar steps we come up with the final Equation Proof without graphs:

$$
\begin{array}{lllr}
(1) & 0 & = -x+x & A5 \\
(2) & y & = 0+y & A3 \\
 & & = (-x+x)+y & 1 \\
 & & = -x+(x+y) & A2 \\
(3) & \multicolumn{2}{l}{--c5+(-c5+c5) = c5} & 4 \\
(4) & --c5 & = --c5+0 & A4 \\
 & & = --c5+(-c5+c5) & A5 \\
 & & = c5 & 3 \\
(5) & c6 & = c6+0 & A4 \\
(6) & 0 & = c4+-c4 & A6 \\
(7) & \multicolumn{2}{l}{c6+(c4+-c4) = (c6+c4)+-c4} & A2 \\
(8) & c4 & = -c5+(c5+c4) & 2 \\
(9) & c5+c4 & = c6+c4 & T9 \\
(10) & \multicolumn{2}{l}{-c5+((c6+c4)+-(-c5+(c6+c4))) = (-c5+(c6+c4))+-(-c5+(c6+c4))} & A2 \\
(11) & \multicolumn{2}{l}{(-c5+(c6+c4))+-(-c5+(c6+c4)) = 0} & A6 \\
(12) & -c5+c6 & = -c5+(c6+0) & 5 \\
 & & = -c5+(c6+(c4+-c4)) & 6 \\
 & & = -c5+((c6+c4)+-c4) & 7 \\
 & & = -c5+((c6+c4)+-(-c5+(c5+c4))) & 8 \\
 & & = -c5+((c6+c4)+-(-c5+(c6+c4))) & 9 \\
 & & = (-c5+(c6+c4))+-(-c5+(c6+c4)) & 10 \\
 & & = 0 & 11 \\
(13) & c6 & = --c5+(-c5+c6) & 2 \\
 & & = c5+(-c5+c6) & 4 \\
 & & = c5+0 & 12 \\
 & & = c5 & A4
\end{array}
$$

Now the following global structure of the proof has become visible:

$$\begin{tikzcd}
 & & \mathcal{A}_1\ \ -(-c5) = c5 \arrow[dr, dash] & & -c5 + c6 = 0\ \ \mathcal{A}_2 \arrow[dl, dash] \\
\mathcal{B}\ \ y = -x + (x + y) \arrow[dr, dash] & & & -(-c5) + (-c5 + c6) = c5 + 0 \arrow[dll, dash] & \\
 & c6 = c5 & & &
\end{tikzcd}$$

A formulation in natural language might now be:

We prove left and right cancellation separately, for left cancellation ... (not shown explicitly here) ...

Right cancellation requires lemma $\mathcal{B}$: $y = -x+(x+y)$ which holds because $y = 0+y = (-x+x)+y = -x+(x+y)$ using left identity, left inverse, and associativity.

Next we prove $\mathcal{A}_1$: $--c5 = --c5+0 = --c5+(-c5+c5) = c5$ by right identity, left inverse, and lemma $\mathcal{B}$.

Additionally $-c5+c6 = -c5+(c6+0) = -c5+(c6+(c4+-c4))$ $= -c5+((c6+c4)+-c4) = -c5+((c6+c4)+-(-c5+(c5+c4)))$ $= -c5+((c6+c4)+-(-c5+(c6+c4))) = (-c5+(c6+c4))+-(-c5+(c6+c4)) = 0$ with right identity, right inverse, associativity, lemma $\mathcal{B}$, assumption of the theorem, associativity, and right inverse. This constitutes lemma $\mathcal{A}_2$: $-c5+c6 = 0$.

Finally we complete the proof: $c6 = --c5+(-c5+c6) = c5+(-c5+c6) = c5+0 = c5$ using lemmata $\mathcal{B}$, $\mathcal{A}_1$, $\mathcal{A}_2$ and right identity, q.e.d.

4 Conclusion

In this paper a method is described to transform equation solution graphs into structured equation chain proofs. The steps of a paramodulation or Knuth-Bendix-based proof are represented in the equation solution graphs by links for each application of an equation. Starting from an idea like the one published in [Li90], the necessary definitions and algorithms are given to meet the special needs of equality reasoning.

The main question with respect to the structuring of proofs is how these proofs could be structured in a different way than is given by the equation solution graph. As suggested by our final example, analogy can lead to restructured graphs. The representation of pure unconditional equality proofs in equality graphs, as in Karl-Hans Bläsius' dissertation, [Bl86], was a promising starting point to construct a procedure analogous to the algorithms known before.

The approach can also be added to the transformation of first order proofs into natural deduction, for instance to Lingenfelder's system [Li90]. Whenever several equations are successively applied to a formula, leading to chains of equality clause nodes, these chains are handled using ESGs. It seems to be straightforward to extend this method to the case where equations occur together with other theories.

5 Literature

[An80]	Peter B. Andrews	*Transforming Matings into Natural Deduction Proofs* Lecture Notes in Comp. Sci. **87**, Springer-Verlag, Proc of 5th CADE (1980), pp. 281-292
[BG90]	Leo Bachmair, Harald Ganzinger	*On Restrictions of Ordered Paramodulation with Simplification* Proceedings 10th CADE, LNCS (1990)
[Bl86]	Karl-Hans Bläsius	*Equality Reasoning Based on Graphs* PhD Thesis, Universität Kaiserslautern (1986) SEKI-Report SR-87-01
[Br75]	D. Brand	*Proving Theorems with the Modification Method* SIAM (Society for Industrial and Applied Mathematics) Journal of Computing (1975)
[Bu83]	A. Bundy	*The Computer Modelling of Mathematical Reasoning* Academic Press, London (1983)
[De88]	Jörg Denzinger	*EQTHEOPOGLES Ein Theorembeweiser für die Prädikatenlogik erster Stufe – basierend auf Rewrite Techniken* Diplomarbeit, Universität Kaiserslautern, Postfach 3049, D-6750 Kaiserslautern (1988)
[Ga86]	Jean H. Gallier	*Logic for Computer Science – Foundations of Automatic Theorem Proving* Harper & Row, Publishers, New York (1986)
[He87]	Alexander Herold	*Combination of Unification Algorithms in Equational Theories* PhD Thesis, SEKI-Report SR-87-05, Universität Kaiserslautern (1987)

[HR86] J. Hsiang, M. Rusinowitch
A New Method for Establishing Refutational Completeness in Theorem Proving
Proceedings 8th CADE, LNCS (1986)

[Hu91] Xiaorong Huang *On a Natural Calculus for Argument Presentation*
to appear as SEKI-Report,
Universität Kaiserslautern (1991)

[KB70] Donald E. Knuth, Peter B. Bendix
Simple Word Problems in Universal Algebras
Computational Problems in Abstract Algebra,
Pergamon Press (1970)

[Li86] Christoph Lingenfelder *Transformation of Refutation Graphs into Natural Deduction Proofs*
SEKI-Report SR-86-10, Universität Kaiserslautern (1986)

[Li90] Christoph Lingenfelder *Structuring Computer Generated Proofs*
PhD Thesis, Universität Kaiserslautern (1990)

[LP90] Christoph Lingenfelder, Axel Präcklein
Proof Transformation with Built-in Equality Predicate
SEKI REPORT SR-90-13, Universität Kaiserslautern (1990)

[Lo78] Donald W. Loveland *Automated Theorem Proving: A Logical Basis*
North Holland (1978)

[Mi83] Dale Miller *Proofs in Higher Order Logic*
Ph.D. Thesis, Carnegie Mellon University,
Tech Report MS-CIS-83-87, University of Pennsylvania, Philadelphia (1983)

[Pe83] Gerald E. Peterson *A Technique for Establishing Completeness Results in Theorem Proving with Equality*
SIAM (Society for Industrial and Applied Mathematics) Journal of Computing (1983),
pp. 82-100

[PN90] Frank Pfenning, Daniel Nesmith
Presenting Intuitive Deductions via Symmetric Simplification
Lecture Notes in AI **449**, Springer-Verlag, Proc of 10th CADE (1990), pp. 336-350

[Pr90] Axel Präcklein
Solving Equality Reasoning Problems with a Connection Graph Theorem Prover
Fachbereich Informatik, Universität Kaiserslautern, SEKI-Report SR-90-07 (1990)

[RW69] G. Robinson, Larry Wos
Paramodulation and Theorem Proving in First Order Theories with Equality
Machine Intelligence (1969)

[Ru87] M. Rusinowitch
Démonstration automatique par des téchniques de reécriture
Thèse de Doctorat d'État en Mathématique, Nancy (1987)

[Wo67] Larry Wos, G. Robinson, D. Carson, L. Shalla
The Concept of Demodulation in Theorem Prooving
JACM (1967), pp. 698-706

[Wo84,] Larry Wos, R. Overbeek, E. Lusk, J. Boyle
Automated Reasoning Introduction and Applications
Prentice Hall (1984)

[ZK88] Hantao Zhang and D. Kapur
First Order Theorem Proving Using Conditional Rewrite Rules
Proceedings 9th CADE (1988) pp. 1-20