Solution Lifting Method for Handling Meta-Variables in TH∃OREM∀

Boris Konev[*][†] **Tudor Jebelean**[‡]

konev@pdmi.ras.ru tudor@risc.uni-linz.ac.at

Abstract

We approach the problem of finding witnessing terms in proofs by the method of meta-variables. We describe an efficient method for handling meta-variables in natural style proofs and its implementation in the TH∃OREM∀ system. The method is based on a special technique for finding meta-substitutions when the proof search is performed in an AND-OR deduction tree. The implementation does not depend on the search strategy and allows easy integration with various special provers as well as with special unification/solving engines. We demonstrate the use of this method in the context of a special forward/backward inference strategy for producing proofs in elementary analysis.

1 Introduction

The problem of generating proofs that can be easily read by non-specialists has raised an increased interest in the recent years (see e.g. [7, 13, 5, 24]). One of the main advantages of "natural style" proving is that one can implement as proof procedures the techniques used by mathematicians, and one can also use advanced solvers for particular domains, in the form of special inference rules.

The need for readability of the result, but even more the flexibility needed for dynamic extension of the prover with special inference rules, as well as for dynamic interaction with special "black box" computing and solving engines makes it difficult to use established efficient techniques such as resolution, tableaux, or connection methods (see e.g. [7] for a survey). Therefore, the

[*]Supported in part by grant #1 of 6th RAS contest-expertise of RAS young scientists projects (1999)

[†]The research described in this publication was made possible in part by Award No. RM1-2409-ST-02 of the U.S. Civilian Research & Development Foundation for the Independent States of the Former Soviet Union (CRDF).

[‡]Supported by Austrian Science Foundation (FWF), SFB project F1302.

usual approach is to employ a sequent calculus (like Gentzen or similar)—whose main draw-back is the lack of determinism in selecting the right replacements in some of the quantified rules (elimination of $\forall$ in assumptions and of $\exists$ in goal). The corresponding problem in "natural style" deduction is the finding of "witnessing terms" (especially for existential goals).

In fact, human produced proofs often proceed by denoting such a desired witness by a variable at the meta-level, which is then instantiated later in the proof with a suitable term suggested by the inferred knowledge. In automated deduction this has been formalized as the method of *meta-variables* (also called *parameters*, *dummy variables*, *free variables*—see [26, 28, 21, 9, 8, 17]) and such an approach is used in almost all sequent-based automated provers since 1980's and in many tableaux provers (see e. g. [27]).

In any implementation of the meta-variable method, one constructs first a (partial) proof tree which contains meta-variables—this is usually called a *pre-proof* (in the literature, it is also named as *proof skeleton*). A substitution of concrete terms instead of meta-variables maps the pre-proof into a concrete proof. Roughly speaking, construction of a proof splits into a propositional part and a "lucky" choice of needed terms. Meta-variables are usually assigned with values by means of unification, but the problem of good choice still remains open. It can be seen that given a pre-proof, finding a correct substitution is NP-hard.

Our paper has two aims. The first aim of the paper, is to present a novel technique for handling meta-variables that we call *solution lifting.* Instead of commonly used enumeration of possible deduction trees, we incorporate `AND` and `OR` nodes into the deduction tree itself. We associate with each node of a pre-proof a substitution which makes the subtree a correct proof. Then, we "lift" the substitutions bottom-up along the pre-proof. If at certain node the substitutions assigned to its successors are incompatible, the pre-proof should be expanded somewhere under this node. If eventually we succeed to lift a substitution up to the root, the proof search is finished.

The resulting procedure uses restricted backtracking: we never roll-back proof tree construction steps, instead, different solutions are tried. Inspection of possible solutions is unavoidable, unless $P = NP$.

The second aim of the paper is to present implementation of the method within the TH∃OREM∀ system [10, 13, 11]. We build our implementation upon the TH∃OREM∀'s mechanism for combining various provers and solvers by adding the functionality which is needed for handling meta-variables.

The TH∃OREM∀ system is a result of work of the TH∃OREM∀ group (see `http://www.theorema.org`) based on earlier work of Buchberger [10]. The meta-variable method was incorporated into the system by the first author of this paper during his visit to RISC.

This paper is an extended version of [22].

2 Meta-variables in Natural Deduction

In this section we explain how meta-variables appear in the sequent-based calculus and how they help to find witnessing terms. Our major contribution is not the fact that our method uses meta-variables, but the special technique for handling them, presented in Section 3. The notion of meta-variables, the use of unification for finding values, the use of Skolem constants instead of Skolem functions, "liberalized" skolemizations, and various variants of ordering restrictions can be found in the literature, e.g. [15, 16, 17, 25, 23, 1]; however, we put the material here in order to make the paper self-contained and to make the notations precise.

We consider a Natural Deduction calculus operating on *proof situations*, which are represented as single-conclusion *sequents* of the form

$$A_1, A_2, \ldots, A_n \vdash G \ \text{ or } \ \Gamma, A \vdash G$$

where $A_1, A_2, \ldots, A_n, A$ denote assumptions, Γ denotes several assumptions, and G is the goal (to be proved).

In order to construct the proof of a sequent, we recursively transform it to one or several sequents until we reduce it to a set of obviously valid sequents (*final sequents*). Reductions are performed by means of *inference rules* of the form

$$\frac{S_1; \quad S_2; \quad \ldots ; \quad S_n}{S}$$

(sequent S reduces to sequents $S_1, \ldots, S_n$). Usually (and in this paper) one only considers rules where S is valid iff all S_i are valid: see Section 3.1. A *proof* is a tree of sequents whose root is the initial sequent and the leaves are final sequents.

The "difficult" rules that eliminate quantifiers (cf. [18]) are:

$$\frac{\Gamma \vdash F_{\{x \to t\}} \vee \exists x F}{\Gamma \vdash \exists x F}, \qquad \frac{\Gamma, F_{\{x \to t\}} \wedge \forall x F \vdash G}{\Gamma, \forall x F \vdash G}, \tag{1}$$

where t is a ground term, and $F_{\{x \to t\}}$ is the result of substituting the term t in place of all free occurrences of x in F.

For these rules a *technique* is needed to generate the appropriate terms t. The main idea of the meta-variable method is to postpone the construction of the concrete term t until the moment when the proof search suggests an appropriate one. This is done by replacing t in (1) by a new meta-variable ξ, whose value is to be found in the proof process:

$$\frac{\Gamma \vdash F_{\{x \to \xi\}} \vee \exists x F}{\Gamma \vdash \exists x F}, \qquad \frac{\Gamma, F_{\{x \to \xi\}} \wedge \forall x F \vdash G}{\Gamma, \forall x F \vdash G}. \tag{2}$$

The proof-tree obtained using rules (2) is called a *pre-proof*, which is called *correct* if there exist a *correct* substitution for all meta-variables which transforms the pre-proof into a real proof. (We may assume that such a substitution is ground, because we can place new constants instead of unresolved meta-variables.) Therefore, a substitution is correct iff **(a)** all intermediate rule applications are correct; and **(b)** final nodes are valid.

Concerning (a), the only rules that may become incorrect are:

$$\frac{\Gamma \vdash F_{\{x \to c\}}}{\Gamma \vdash \forall x F}, \qquad \frac{\Gamma, F_{\{x \to c\}} \vdash G}{\Gamma, \exists x F \vdash G}, \tag{3}$$

and the classical sequent calculus [18] requires that c be a new free variable (we can also view at c as to a new constant, that is more natural and does not change validity of the derivation) that is:

$$c \text{ does not occur in } \Gamma, F, \text{ and } G. \tag{4}$$

A straightforward way to ensure (4) is to keep track of a set of restrictions of the form $c \notin \xi$, for all ξ occurring in Γ, F, G, see, e.g., [25, 23]. However, this turns out to be excessively restrictive: it often leads to much backtracking and/or to longer proofs [1].

Therefore, we use a milder version of restriction which does not actually ensure (4), but still produces only valid proofs. For each meta-variable ξ and for each Skolem constant c that occur in F we keep track of the restrictions

$$c_1 \succ \xi, \ldots, c_n \succ \xi \text{ and } c_1 \succ c, \ldots, c_n \succ c.$$

In order to test whether a substitution σ satisfies a set of restrictions R, first, for each replacement in σ of the form $\xi \to t$ (ξ: meta-variable, t: term), we add to the set of restrictions: $\xi \succ c$ and $\xi \succ \zeta$ for each Skolem constant c and for each metavariable ζ occurring in t. Then, we check whether the relation $\succ$ does not have any cycle.

This mechanism ensures the soundness of the calculus, because it is essentially a simulation of the method presented in [1]: Skolem constants, as we use them, correspond to Skolem functions; restrictions correspond to function arguments. Note that the relation $\succ$ is of practical significance for the presentation of the proof, as we discuss in Section 4.

Concerning (b), final proof situations are of the form

$$\Gamma, F \vdash F, \quad \text{and} \quad \Gamma, A, \neg A \vdash G,$$

where F and A are atomic formulas. Therefore, a leaf of a pre-proof is transformable into a final proof situation if and only if one of the assumptions is unifiable with the goal or an assumption is unifiable with the negation of an assumption. The union of all such pairs of formulas forms a *unification problem* for the pre-proof.

3 The proof search procedure

In our system, the successive reductions (and finally the proof) are represented as an `AND-OR` *deduction tree.* `AND` nodes correspond to the usual sequent reductions (see Section 2), while `OR` nodes represent proof alternatives; `OR` nodes cannot be unary.

Speaking formally, a deduction tree generated by the system is not a proof: it contains extra branches. We call a deduction tree an *`AND/OR` proof* if for every `OR` node there exists a successor such that the tree made by recursive replacing the `OR` node with the corresponding successor is a proof as defined in Section 2.

TH∃OREM∀ can combine several proof engines called **basic provers**, some of them specialized for certain domains. Basic provers return subtrees representing one or several deduction steps. A **control procedure** activates the various basic provers, combines their output and thus constructs the proof-tree in a top-down inductive manner. The proof search procedure of TH∃OREM∀ is described in detail in [14].

A *proof value* is assigned to each node of a deduction tree. The proof value of a node can be *proved*, *failed* or *pending* (with the obvious meaning). After each step of inference, some of the proof values may change. The control procedure *propagates* the proof values in a bottom-up manner, changing *pending* into *proved* or *failed* when possible.

The mechanism that handles meta-variables splits on two levels—the level of a basic prover and the level of the control procedure. Indeed, the basic prover *assigns* some particular values to meta-variables, while the control procedure ensures that values assigned to a meta-variable on different branches of the deduction tree are *compatible.*

3.1 Basic prover for first-order logic

In the TH∃OREM∀ system, the basic prover that carries-out the inferences in first-order logic is called `PND` (Proof by Natural Deduction) [12]. In this section we describe the extension of this prover for handling meta-variables. The prover is implemented as a set of rules which correspond to sequent calculus inferences.

The first rule is the one which detects [possible] final proof situations: Given a sequent of the form

$$\Gamma, A \vdash B, \quad (\text{ or } \ \Gamma, A, \neg B \vdash G),$$

where A and B are quantifier free and this pair of formulas was not tried earlier, check whether they are unifiable. If the most general unifier of A, B satisfies the set of restrictions R, associated to the node of the proof tree as explained later, then the prover will create an `OR` node with two alternative branches:

- A *"proved"* branch where the new substitution and the augmented set of restrictions are retained as "proof-value".
- A *"pending"* branch with the old proof situation, but where this particular substitution is forbidden in future inferences.

(Information about the "forbidden" substitution is than recursively passed to descendants of the node).

Informally, the prover tries a substitution and keeps the possibility to go further if it is not correct.

If the given proof situation cannot be transformed into a final proof situation, (or the substitution is not compatible with the set of restrictions) then `PND` tries to apply other rules, which basically split either the goal or one of the assumptions in a natural way. Some of these rules will create `AND` nodes with several branches.

- Rules for simplification and splitting:

$$\frac{\Gamma, \neg G_2, \ldots, \neg G_k \vdash G_1}{\Gamma \vdash G_1 \vee \cdots \vee G_2}; \quad \frac{\Gamma, G_1 \vdash G_2}{\Gamma \vdash G_1 \implies G_2};$$

$$\frac{\Gamma, \neg A \vee B \vdash G}{\Gamma, A \implies B \vdash G}; \quad \frac{\Gamma, A_1, \ldots, A_k \vdash G}{\Gamma, A_1 \wedge \cdots \wedge A_k \vdash G};$$

$$\frac{\Gamma, G \vdash \mathit{false}}{\Gamma \vdash \neg G}; \quad \frac{\Gamma, A \vdash G}{\Gamma, \neg\neg A \vdash G}; \quad \frac{\Gamma, \neg A_1 \vee \cdots \vee \neg A_k \vdash G}{\Gamma, \neg(A_1 \wedge \cdots \wedge A_k) \vdash G};$$

$$\frac{\Gamma, \neg A_1 \wedge \cdots \wedge \neg A_k \vdash G}{\Gamma, \neg(A_1 \vee \cdots \vee A_k) \vdash G}; \quad \frac{\Gamma, A \wedge \neg B \vdash G}{\Gamma, \neg(A \implies B) \vdash G};$$

$$\frac{\Gamma, \forall x_1, \ldots, x_n \neg F \vdash G}{\Gamma, \neg(\exists x_1, \ldots, x_n F) \vdash G}; \quad \frac{\Gamma, \exists x_1, \ldots, x_n \neg F \vdash G}{\Gamma, \neg(\forall x_1, \ldots, x_n F) \vdash G}.$$

$$\frac{\Gamma, A_1 \vdash G; \quad \cdots \quad \Gamma, A_k \vdash G}{\Gamma, A_1 \vee \cdots \vee A_k \vdash G}; \quad \frac{\Gamma \vdash G_1; \quad \cdots \quad \Gamma \vdash G_k}{\Gamma \vdash G_1 \wedge \cdots \wedge G_k}.$$

Of particular interest to the meta-variables method are the following:

- Rules that introduce Skolem constants:

$$\frac{\Gamma \vdash F_{\{x_1 \to c_1, \ldots, x_n \to c_n\}}}{\Gamma \vdash \forall x_1, \ldots, x_n F}; \quad \frac{\Gamma, F_{\{x_1 \to c_1, \ldots, x_n \to c_n\}} \vdash G}{\Gamma, \exists x_1, \ldots, x_n F \vdash G},$$

  where $c_1, \ldots c_n$ are new Skolem constants[1]. For each meta-variable ξ and for each Skolem constant c that occur in F we associate with the node of the proof tree the set of restrictions

$$c_1 \succ \xi, \ldots, c_n \succ \xi \text{ and } c_1 \succ c, \ldots, c_n \succ c,$$

  as explained in Section 2 and pass them to the successors.

[1]In the classic Gentzen's calculus, [18], free variables are used instead of constants. Replacement of the variables with constants keeps the proof valid and is more natural for non-professionals.

- Rules that introduce meta-variables:

$$\frac{\Gamma \vdash G_{\{x_1 \to \xi_1, \ldots, x_n \to \xi_n\}} \vee \exists x_1, \ldots, x_n G}{\Gamma \vdash \exists x_1, \ldots, x_n G};$$

$$\frac{\Gamma, A_{\{x_1 \to \xi_1, \ldots, x_n \to \xi_n\}} \wedge \forall x_1, \ldots, x_n A \vdash G}{\Gamma, \forall x_1, \ldots, x_n A \vdash G},$$

where $\xi_1, \ldots, \xi_n$ are new meta-variables.

3.2 Control procedure

We consider first the situation of deterministic proofs—that is, we have only `AND` nodes. During the construction of the pre-proof, each node is augmented with a set R of restrictions of the form $a \succ b$, which are propagated top-down. Each final node (leaf) is assigned the proof value *proved* and the appropriate substitutions for the meta-variables. When all the subnodes of an `AND` node are *proved*, then we try to combine the corresponding substitutions and restrictions as shown below (compatibility of a substitution with the restrictions is checked as described in the previous section, at the final proof rules). If successful, the node is also *proved*. Inductively, this will construct bottom-up a correct substitution σ for the root. At the root, still unresolved meta-variables are replaced with new constants in order to make the substitutions ground.

The induction is provided by the function:

Function `Simple_Combine`

Input: A list of pairs $\{(\lambda_1, R_1), (\lambda_2, R_2), \ldots, (\lambda_n, R_n)\}$ [substitutions and restrictions of the subnodes].

Output: A pair (σ_N, R_N) or `NULL` [substitution and restrictions of the father node].

Method:

- Let $\lambda_i = \{\xi_{i,k} \to t_{i,k}\}_{k=1,\ldots}$.
- Find a most general unifier σ_N for $\langle \xi_{1,1} \ldots \xi_{n,1} \ldots \rangle$ and $\langle t_{1,1} \ldots t_{n,1} \ldots \rangle$.
- If σ_N exists and satisfies $\cup_i R_i$, then return $(\sigma_N, \cup_i R_i)$ else return `NULL`.

Obviously the resulting λ is a most general substitution mapping the corresponding subtree in a real proof. (Note that we do not have ordering restrictions between meta-variables, and different most general unifiers may differ only in variable renaming-way; therefore, any most general unifier can be selected in the functions.)

Note that the unification algorithm used above can be particularly tailored to the special domains involved in the proof problem with the following restriction: There must be a way to compute a solution to the specific unification problem incrementally, i.e., given solutions corresponding to subnodes $N_1, N_2, \ldots, N_n$, one should be able to construct a solutions that satisfies all these subnodes *simultaneously*, or to find out that there is no such a simultaneous solution.

Consider now the case, when the deduction tree may contain `OR` nodes. An `OR` node N may have some *pending* subnodes, as well as some *proved* subnodes $N_1, \ldots, N_n$ assigned with $(\lambda_1, R_1), \ldots, (\lambda_n, R_n)$. It is natural to assign to N the set $\{(\lambda_1, R_1), \ldots, (\lambda_n, R_n)\}$ of "possible" solutions.

The *pending* subnodes have no solutions assigned, however we keep a list of pointers to these nodes, in order to return there in the case of failure.

Now we are ready to describe the general case. To each node we add a *solution system*—a set whose elements are either pairs of the form (σ_N, R_N) or pointers to nodes of the constructed deduction tree. Let functions `Pairs`() and `Pointers`() select from a solution system the elements of the first and the second type respectively. A node is *proved* if the solution system assigned to it contains a pair. Thus, solution systems play the key role in manipulating proof values; in fact, we compute proof values and solution systems by means of one function `Compute_Value`.

To describe the function `Compute_Value` we need two auxiliary functions `CombineOR` and `CombineAND` that compute a solution system associated with `OR` and `AND` nodes of the deduction tree.

For an `OR` node the solution system is the union of solution systems of its immediate successors.

Function `CombineOR`

Input: A list of solution systems $\{S_1, \ldots, S_r\}$.

Output: A solution system S.

Method:

- Return the union of $S_1, \ldots, S_r$.

If all immediate successors of an `AND` node are proved, a solution system is assigned to each of them. Let us denote these solution systems by $S_1, \ldots, S_r$. Each substitution occurring in `Pairs`(S_i) is correct for the corresponding subtree. `CombineAND` chooses an element s_i of `Pairs`(S_i) for all $i : 1 \leq i \leq r$, and passes $s_1, \ldots, s_r$ to the function `Simple_Combine`. The output of `Simple_Combine` contains a correct substitution or is `NULL`.

If no solution system can be obtained in this way, the function returns the union of sets of pointers occurring in the input solution systems. The proof search procedure can expand further the deduction tree starting from these nodes, then new leaves with the *proved* proof value may appear.

If there are no pointers, there is no hope to prove the node, and the function returns `NULL`.

Function `CombineAND`

Input: A list of solution systems $\{S_1, \dots, S_r\}$.

Output: A solution system $\{(\sigma_1, R_1), \dots, (\sigma_M, R_M), p_1, \dots, p_N\}$ or a list of pointers $\{p_1, \dots, p_N\}$ or `NULL`.

Method:

- Let $S = \emptyset$.
- Let $P = \texttt{Pointers}(S_1) \cup \cdots \cup \texttt{Pointers}(S_r)$.
- For all sequences $(s_1, s_2, \dots, s_r) \in \texttt{Pairs}(S_1) \times \texttt{Pairs}(S_2) \times \cdots \times \texttt{Pairs}(S_r)$
  - if $\texttt{Simple\_Combine}(s_1, \dots, s_r) \neq \texttt{NULL}$, add the output of $\texttt{Simple\_Combine}(s_1, \dots, s_r)$ to S.
- If $S = \emptyset$ and $P = \emptyset$, return `NULL`,
- else if $S = \emptyset$, return P,
- else return $S \cup P$.

The function `Compute_Value` combines the computation of proof values with the computation of solution systems. In addition, it builds the global list `NList` of nodes.

Function `Compute_Value`

Input: A node N having immediate successors $N_1, \dots, N_t$.

Output: A proof value and a solution system assigned to N.

Side effect: Changes the global list `NList`.

Method:

- If N is an `AND` node
  - If the proof value of one of $N_1, \dots, N_t$ is *failed*, set the proof value of N to *failed* and exit.
  - If the proof value of one of $N_1, \dots, N_t$ is *pending*, set the proof value of N to *pending* and exit.

- Let $S_1, \ldots, S_t$ be the solution systems associated with $N_1, \ldots, N_t$ respectively. Let $S_{\texttt{AND}} = \texttt{CombineAND}(S_1, \ldots, S_t)$.
  * if $\texttt{Pairs}(S_{\texttt{AND}}) = \emptyset$ and $\texttt{Pointers}(S_{\texttt{AND}}) = \emptyset$, then set the proof value of N to *failed* and exit.
  * if $\texttt{Pairs}(S_{\texttt{AND}}) = \emptyset$ and $\texttt{Pointers}(S_{\texttt{AND}}) \neq \emptyset$, then set the proof value of N to *pending*, add $\texttt{Pointers}(S_{\texttt{AND}})$ to `NList` and exit.
  * if $\texttt{Pairs}(S_{\texttt{AND}}) \neq \emptyset$, then set the proof value of N to *proved*, associate with N the solution system $S_{\texttt{AND}}$ and exit.

- If N is an `OR` node
  - If the proof values of all of $N_1, \ldots, N_t$ are *failed*, set the proof value of N to *failed* and exit.
  - If the proof values of all of $N_1, \ldots, N_t$ are *pending* or *failed*, set the proof value of N to *pending* and exit.
  - If the proof value of one of $N_1, \ldots, N_t$ is *proved*,
    * Let $N'_{i_1}, \ldots, N'_{i_j}$ be those of the nodes $N_1, \ldots, N_t$ that have the *proved* proof value. Let $S_{i_1}, \ldots, S_{i_j}$ be the solution systems associated with $N'_{i_1}, \ldots, N'_{i_j}$ respectively.
    * Let $N''_{k_1}, \ldots, N''_{k_l}$ be those of the nodes $N_1, \ldots, N_t$ that have the *pending* proof value. Let $p_{k_1}, \ldots p_{k_l}$ be the pointers to the nodes $N''_{k_1}, \ldots, N''_{k_l}$ respectively. Let T be the solution system $\{p_{k_1}, \ldots p_{k_l}\}$.
    * Set the proof value of N to *proved*, associate with it the solution system $\texttt{CombineOR}(S_{i_1}, \ldots, S_{i_j}, T)$ and exit.

Note that some final proof steps may create the empty solution system $\mathcal{E} = \{\epsilon, \emptyset\}$, where ϵ in the empty substitution. The empty solution system can be combined with any solution system.

The proof search is performed by the function `Proof_Search` (the discussion of the proof strategy follows).

Function `Proof_Search`

Input: Initial proof situation $\Gamma \vdash G$.

Output: If $\Gamma \vdash G$ is valid, a deduction tree T presenting its proof.

Method:

- Construct the deduction tree T consisting of a single `AND` node corresponding to the initial proof situation.

- Let `NList` $= \emptyset$.
- While the proof value of the root of T is *pending* do
  - Choose (according to the proof search strategy) a leaf L of T having the *pending* proof value such that one of the following conditions is satisfied:
    1. no predecessor of L has the *proved* proof value;
    2. L is a successor of one of the nodes occurring in `NList`.
  - If no proof-rule is applicable to L, set the proof value of L to *failed*.
  - Apply (according to the proof search strategy) a proof-rule to L, generating one or more successors of L in an AND or an OR node.
  - Propagate proof values from leaves to root applying inductively the function `Compute_Value`. During the propagation, the list `NList` may change.
  - Exclude from the list `NList` all nodes that do not have successors with the *pending* proof value.

Note that in an `AND`/`OR` deduction produced by the described proof search procedure, each `OR` node has the following property: One of its two successor is a final node; thus, backtracking is not actually performed. Note also, that the procedure is not restricted to this case, and we can implement in it non-proof-confluent calculi, like versions of the connection method [6] where backtracking is unavoidable.

Proof search strategy. The function `Proof_Search` finds a proof of a given valid formula if supplied with a *fair* proof strategy. A strategy is fair if on an unsuccessful run with unlimited resources, for each branch of the deduction tree the following holds: To each formula, occurring in a proof situation of the branch, a rule is eventually applied, and if the formula is an existential goal or a universal assumption, metavariables are introduced in it infinitely many times. (Our notion of fair derivation coincides with the notion used for semantic tableaux, [7].)

The strategy used in the system is the following: We extend a deduction tree in a depth-first manner, until we found a candidate to be a final proof situation. When all successors of an `AND` node are proved, we try to combine these solutions. When it is not possible, we extend the deduction tree at the closest to the `AND` node *pending* leaf. Thus, we combine depth-first manner for exploring the tree with breadth-first manner of node selection to continue the search in the case of failure.

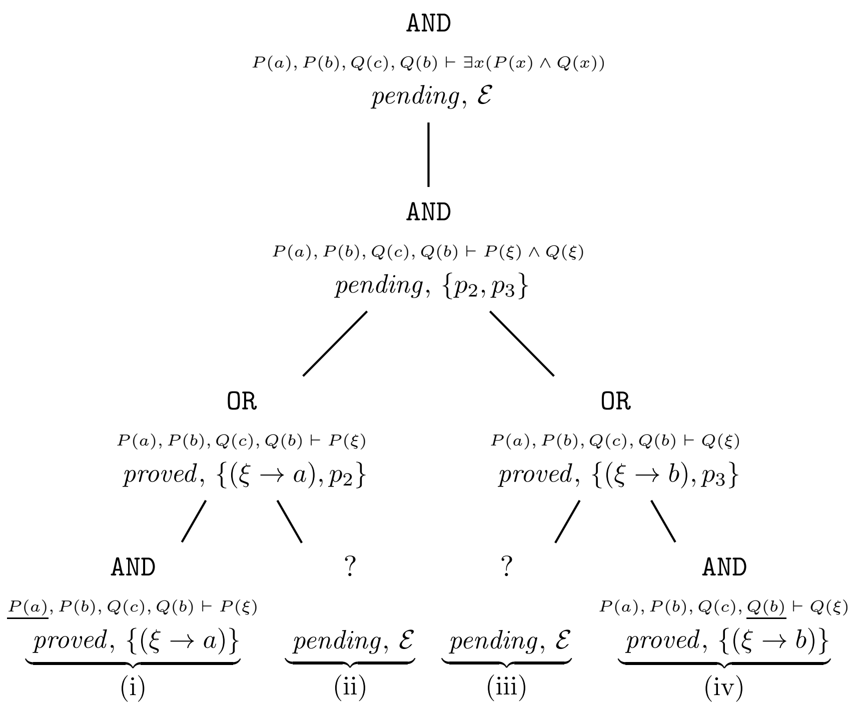
AND
$P(a), P(b), Q(c), Q(b) \vdash \exists x(P(x) \wedge Q(x))$
pending, $\mathcal{E}$
AND
$P(a), P(b), Q(c), Q(b) \vdash P(\xi) \wedge Q(\xi)$
pending, $\{p_2, p_3\}$
OR
$P(a), P(b), Q(c), Q(b) \vdash P(\xi)$
proved, $\{(\xi \to a), p_2\}$
OR
$P(a), P(b), Q(c), Q(b) \vdash Q(\xi)$
proved, $\{(\xi \to b), p_3\}$
AND
$\underline{P(a)}, P(b), Q(c), Q(b) \vdash P(\xi)$
proved, $\{(\xi \to a)\}$
(i)
?
pending, $\mathcal{E}$
(ii)
?
pending, $\mathcal{E}$
(iii)
AND
$P(a), P(b), Q(c), \underline{Q(b)} \vdash Q(\xi)$
proved, $\{(\xi \to b)\}$
(iv)

Figure 1: Deduction tree for $P(a), P(b), Q(c), Q(b) \vdash \exists x(P(x) \wedge Q(x))$ with proof values and solution systems.

In Figure 1 we present a deduction tree where we show proof values and associated solution systems.

In the figure, p_2 and p_3 are pointers to the leaves (ii) and (iii) respectively. At the presented moment, the list `NList` contains nodes (ii) and (iii). Hence, the procedure chooses as L one of the leaves (ii) and (iii), w.l.g., the leaf (ii). After the procedure expands the deduction tree, we get the deduction tree presented in Figure 2. The solution system associated with the root of the tree contains a correct substitution $\xi \to b$. The pointer p_3 shows the place where the deduction tree can be further expanded.

4 Example

This section describes our experiments with the system on a comprehensive example. It turns out that an unrestricted introduction of meta-variables (as described in Section 3.1) leads to large deduction trees, and, therefore, the system based on this method only is not able to find a proof in a reasonable time. Currently, we are working on strategy of combination of the method with some more special rules (e.g. instantiation of the goal if it matches a ground assumption), which are applied before the introduction of meta-

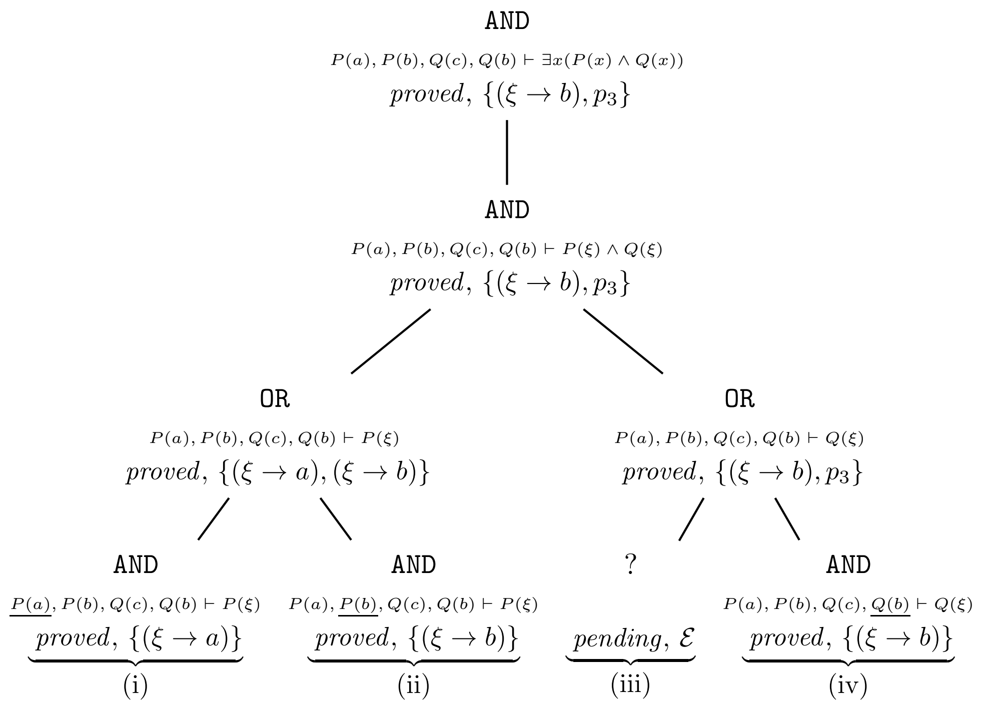
AND
$P(a), P(b), Q(c), Q(b) \vdash \exists x(P(x) \wedge Q(x))$
$\mathit{proved},\ \{(\xi \to b), p_3\}$
AND
$P(a), P(b), Q(c), Q(b) \vdash P(\xi) \wedge Q(\xi)$
$\mathit{proved},\ \{(\xi \to b), p_3\}$
OR
$P(a), P(b), Q(c), Q(b) \vdash P(\xi)$
$\mathit{proved},\ \{(\xi \to a), (\xi \to b)\}$
OR
$P(a), P(b), Q(c), Q(b) \vdash Q(\xi)$
$\mathit{proved},\ \{(\xi \to b), p_3\}$
AND
$\underline{P(a)}, P(b), Q(c), Q(b) \vdash P(\xi)$
$\underbrace{\mathit{proved},\ \{(\xi \to a)\}}_{\text{(i)}}$
AND
$P(a), \underline{P(b)}, Q(c), Q(b) \vdash P(\xi)$
$\underbrace{\mathit{proved},\ \{(\xi \to b)\}}_{\text{(ii)}}$
?
$\underbrace{\mathit{pending},\ \mathcal{E}}_{\text{(iii)}}$
AND
$P(a), P(b), Q(c), \underline{Q(b)} \vdash Q(\xi)$
$\underbrace{\mathit{proved},\ \{(\xi \to b)\}}_{\text{(iv)}}$

Figure 2: Finished deduction tree for $P(a), P(b), Q(c), Q(b) \vdash \exists x(P(x) \wedge Q(x))$.

variables. Such an approach is especially good for the case when a proof situation consists of formulas of a special kind, like Horn-formulas.

The presented proof was fully automatically generated by the system by use of the following strategy:

- Try to detect final proof situations.
- Apply rules for simplification and splitting.
- Match the goal vs. a ground assumption: backward chaining on the goal.
- Introduce meta-variables into the goal.
- Match assumptions: forward reasoning.

The difference between this strategy and the one described in Section 3.2 is that backward chaining is performed before introduction of meta-variables into the goal and meta-variables are not introduced into assumptions at all. (The latter choice is questionable so it is controlled by an option of the prover.)

In the proof shown in **Appendix 1**, Skolem constants are denoted by integer subscripted symbols (e. g. N_0), while meta-variables are denoted by symbols superscripted with "*" and subscripted with integers (e. g. N_0^*). Only one successful branch is shown (the simplification is also done automatically). The (few) alternative branches which have been removed include: further use of the transitivity (formula (>=>)), and alternative solutions for i_0^*, j_0^* (an alternative branch also finds the solution $\max(N_2, N_1)$).

We mentioned in Section 2 that ordering constraints are of great significance for the presentation of proofs because it can be interpreted as: if $\xi \succ c$, (for ξ metavariable and c Skolem constant), then the deduction step introducing ξ should occur *after* the deduction step introducing c. Therefore, $\succ$ indicates the order in which the deduction steps should be rearranged in order to be able to transform the pre-proof into a concrete proof in which (4) holds—which is more natural.

In this proof the relation $\succ$ over the Skolem functions and the metavariables is:

$$n_0 \succ N_0^* \succ i_0^*, j_0^* \succ N_0, N_1 \succ \delta_0 \succ \epsilon_0.$$

This suggests the rearrangement of the steps presented in **Appendix 2** (only the part after the line labeled (9) changes).

Note that the steps of the type "Because (14), we instantiate (eps) to ..." have been changed into "We instantiate (eps) to ...", in order to avoid forward references. In the former case the prover uses (14) as a hint in order to instantiate (eps) usefully, however the hint is not available in the latter proof. This shows an additional reason why searching for a proof in the "correct" order is more difficult than in the "incorrect" one.

This proof can be transformed into a correct proof (without metavariables) by replacing all the occurrences of the metavariables with their ground value and by changing some of the explanatory text. The part of the proof which changes is listed in **Appendix 3**.

5 Conclusions and related work

The presented approach has the following distinctive characteristics:

- The method can be used together with any set of inference rules, in particular with any special provers.
- The method can be used together with any special (constraint) solvers and rewrite engines based on implicit or explicit equality sets provided the simultaneous solution can be computed incrementally.
- The proof is obtained as soon as possible, and inconsistent meta-substitutions are eliminated as soon as they can be detected.

- Proof sub-trees are not destroyed, but they are reused as much as possible.
- The method can be used together with any strategy for constructing the proof tree.
- The method can be easily parallelized.

A similar approach has been independently discovered by M. Giese [20, 19] for semantic tableaux. His primer goal is proof efficiency; the readability of the resulted proof is hardly discussed.

We survey briefly other implementations of the metavariable method we are aware of.

In free variable tableau calculi (see e.g. [3]), the choice of substitution is done non-deterministically and the substitution is applied immediately to the whole pre-proof. The indeterminism needs to be resolved by *enumeration* of all possible substitutions and *backtracking*. Usually an AND-OR tree (of proofs) is used for this purpose (cf. [3]). The main disadvantage of such a method is that the information on inference steps may have been lost after backtracking—this is why this kind of methods are also called *destructive*.

A straightforward way to enforce non-destructiveness for tableau calculus is described in [16, 17]. The method consists of delaying any application of the substitution until all branches can be closed simultaneously. This requires global termination testing after each deduction step.

Recently, complete destructive procedures that avoid backtracking have appeared [4] (similar ideas can be found in [2]). The essence of these procedures is that whenever a substitution is applied, new subtrees are added to all affected branches of the deduction tree. These subtrees "reconstruct" the destroyed formulas. Thus, backtracking is avoided at the cost of increasing the size of the deduction tree.

References

[1] R. Hähnle and P. Schmitt. The liberalized δ-rule in free variable semantic tableaux. *Journal of Automated Reasoning*, 13(2):211–221, 1994.

[2] P. Baumgartner, N. Eisinger, and U. Furbach. A confluent connection calculus. In *CADE99*, pages 329–343, 1999.

[3] B. Beckert and R. Hähnle. Analytic tableaux. In W. Bibel and P. Schmitt, editors, *Automated Deduction—a Bases for Applications*, number 8 in Applied Logic Series, pages 11–41. Kluver Academic Publishers, 1998.

[4] Bernhard Beckert. Depth-first proof search without backtracking for free variable semantic tableaux. In Neil Murray, editor, *Position Papers, International Conference on Automated Reasoning with Analytic Tableaux and Related Methods, Saratoga Springs, NY, USA*, Technical Report 99-1, pages 33–54. Institute for Programming and Logics, Department of Computer Science, University at Albany – SUNY, 1999.

[5] M. Beeson. Automatic derivation of the irrationality of e. In *CALCULEMUS 99*, Electronic Notes in Theoretical Computer Science, 99. `www.elsevier.nl/locate/entcs`.

[6] W. Bibel. *Automated Theorem Proving.* Vieweg, Braunschweig, second edition, 1987.

[7] W. Bibel and P. Schmitt, editors. *Automated Deduction—a Bases for Applications*, volume 1–3 of *Applied Logic Series.* Kluver Academic Publishers, 1998.

[8] K. Broda. The relationship between semantic tableau and resolution theorem proving. In *Proceedings of Workshop on Logic*, Debrecen, Hungary, 1980. Also published as technical report, Imperial College, Department of Computing, London, UK.

[9] F. M. Brown. Toward the automation of set theory and its logic. *Artificial Intelligence*, 10:281–316, 1978.

[10] B. Buchberger. Using *Mathematica* for doing simple mathematical proofs. In *4-th International Mathematica users' conference, Tokyo.* Wolfram Media Publishing, 1996.

[11] B. Buchberger, C. Dupre, T. Jebelean, F. Kriftner, K. Nakagawa, D. Vasaru, and W. Windsteiger. Theorema: A progress report. Technical Report 99-42, Research Institute for Symbolic Computation, December 1999.

[12] B. Buchberger and T. Jebelean. The predicate logic prover. In *Proceedings of the First International Theorema Workshop*, number 97-20 in RISC Linz report series. Research Institute for Symbolic Computation, June 1997.

[13] B. Buchberger, T. Jebelean, F. Kriftner, M. Marin, E. Tomuta, and D. Vasaru. A survey of the *Theorema* project. In W. Kuechlin, editor, *Proceedings of ISSAC'97 (International Symposium on Symbolic and Algebraic Computation)*, Maui, Hawaii, July 1997.

[14] B. Buchberger and E. Tomuta. Combining provers in the *t*heorema system. Technical report 98-02, Research Institute for Symbolic Computation, 1998.

[15] G. V. Davydov, S. Yu. Maslov, G. E. Mints, V. P. Orevkov, and A. O. Slisenko. A computer algorithm for the determination of deducibility on the basis of the inverse method. In J. Siekmann and G. Wrightson, editors, *Automation of Reasoning (Classical papers on Computational Logic)*, volume 2, pages 531–541. Springer Verlag, 1983. Original paper (in Russian) appeared in 1969.

[16] A.I. Degtyarev, A.V. Lyaletski, and M.K. Morokhovets. Evidence algorithm and sequent logical inference search. In *LPAR 1999*, pages 44–61, 1999.

[17] M. Fitting. *First-Order Logic and Automated Theorem Proving.* Springer, second edition, 1996.

[18] G. Gentzen. Untersuchungen über das logische schließ. *Mathematische Zeitschrift*, 39:176–210, 405–433, 1934.

[19] Martin Giese. Incremental closure of free variable tableaux. to appear in LNCS series, 2001.

[20] Martin Giese. Proof search without backtracking using instance streams, position paper. In *Proc. Int. Workshop on First-Order Theorem Proving, St. Andrews, Scotland*, 2000.

[21] C. Kanger. *A simplified proof method for elementary logic*, pages 87–93. Computer Programming and Formal Systems / Studies in Logic and Foundations of Mathematics. North-Holland, 1963.

[22] B. Konev and T. Jebelean. Using meta-variables for natural deduction in theorema. In M. Kerber and M. Kohlhase, editors, *Calculemus 2000: integration of symbolic computation and mechanized reasoning*, A. K. Peters, Natik, Massatchussets, 2000.

[23] J. Krajíček and P. Pudlák. The number of proof lines and size of proofs in first order logic. *Arch. Math. Logic*, 27:69–84, 1988.

[24] E. Melis and V. Sorge. Employing external reasoners in proof planning. In *CALCULEMUS 99*, Electronic Notes in Theoretical Computer Science, 99. `www.elsevier.nl/locate/entcs`.

[25] V. P. Orevkov. *Complexity of Proofs and Their Transformations in Axiomatic Theories.* Translations of Mathematical Monographs. American Mathematical Society, 1991.

[26] D. Prawitz. An improved proof procedure, 1960. Theoria 26.

[27] A. Robinson and A. Voronkov. *Handbook of Automated Reasoning.* Elsevier, 2001.

[28] H. Wang. Toward mechanical mathematics. *IBM J. of Research and Develpment*, 4(1), 1960.

Appendix 1: Automatically generated proof

Prove:
(lim +) $\lim[f,a] \wedge \lim[g,b] \Rightarrow \lim[f \oplus g, a+b]$,
under the assumptions:
(lim:) $l\forall_{a,f}(\lim[f,a] :\Leftrightarrow \forall_\epsilon\ (\epsilon > 0 \Rightarrow \exists_N\ \forall_n\ (n \geq N \Rightarrow |f[n]-a| < \epsilon)))$,
($\oplus$:) $\forall_{f,g,x}((f \oplus g)[x] := f[x] + g[x])$,
(|++|) $l\forall_{a,b,x,y,\delta,\epsilon}(|x-a| < \delta \wedge |y-b| < \epsilon \Rightarrow |(x+y)-(a+b)| < \delta + \epsilon)$,
(max) $\forall_{i,j,k}(k \geq \max[i,j] \Rightarrow k \geq i \wedge k \geq j)$,
(eps) $\forall_\epsilon\ (\epsilon > 0 \Rightarrow \exists_\delta\ (\delta > 0 \wedge (\delta + \delta = \epsilon)))$,
(<=<) $\forall_{t,u,v}(u < v \wedge (v = t) \Rightarrow u < t)$.
We prove (lim +) by the deduction rule.
We assume
(1) $\lim[f,a] \wedge \lim[g,b]$
and show
(2) $\lim[f \oplus g, a+b]$.
For proving (2), by the definition (lim:), it suffices to prove:
(3) $\forall_\epsilon\ (\epsilon > 0 \Rightarrow \exists_N\ \forall_n\ (n \geq N \Rightarrow |(f \oplus g)[n] - (a+b)| < \epsilon))$.
Using ($\oplus$:), the goal (3) is transformed into:
(4) $\forall_\epsilon\ (\epsilon > 0 \Rightarrow \exists_N\ \forall_n\ (n \geq N \Rightarrow |(f[n]+g[n]) - (a+b)| < \epsilon))$.
The formula (1.1) is expanded by the definition (lim:) into:
(5) $\forall_\epsilon\ (\epsilon > 0 \Rightarrow \exists_N\ \forall_n\ (n \geq N \Rightarrow |f[n]-a| < \epsilon))$.
The formula (1.2) is expanded by the definition (lim:) into:
(6) $\forall_\epsilon\ (\epsilon > 0 \Rightarrow \exists_N\ \forall_n\ (n \geq N \Rightarrow |g[n]-b| < \epsilon))$.
For proving (4) we take all variables arbitrary but fixed and prove:
(7) $\epsilon_0 > 0 \Rightarrow \exists_N\ \forall_n\ (n \geq N \Rightarrow |(f[n]+g[n]) - (a+b)| < \epsilon_0)$.
We prove (7) by the deduction rule.
We assume
(8) $\epsilon_0 > 0$
and show
(9) $\exists_N\ \forall_n\ (n \geq N \Rightarrow |(f[n]+g[n]) - (a+b)| < \epsilon_0)$.
For proving (9) we have to find N_0^* such that:
(10) $\forall_n\ (n \geq N_0^* \Rightarrow |(f[n]+g[n]) - (a+b)| < \epsilon_0)$.
For proving (10) we take all variables arbitrary but fixed and prove:
(11) $n_0 \geq N_0^* \Rightarrow |(f[n_0]+g[n_0]) - (a+b)| < \epsilon_0$.
We prove (11) by the deduction rule.
We assume
(12) $n_0 \geq N_0^*$
and show
(13) $|(f[n_0]+g[n_0]) - (a+b)| < \epsilon_0$.
In order to prove (13), by (<=<), it is sufficient to prove:
(14) $\exists_v\ (|(f[n_0]+g[n_0]) - (a+b)| < v \wedge (v = \epsilon_0))$.
Because (14), we instantiate (eps) to:

(15) $\epsilon_0 > 0 \Rightarrow \exists_\delta\ (\delta > 0 \wedge (\delta + \delta = \epsilon_0))$.

From (8) and (15) we obtain by modus ponens

(16) $\exists_\delta\ (\delta > 0 \wedge (\delta + \delta = \epsilon_0))$.

By (16) we can take appropriate values such that:

(17) $\delta_0 > 0 \wedge (\delta_0 + \delta_0 = \epsilon_0)$.

For proving (14) (using (17.2)) it suffices to prove:

(18) $|(f[n_0] + g[n_0]) - (a + b)| < \delta_0 + \delta_0 \wedge (\delta_0 + \delta_0 = \epsilon_0)$.

Because a part of (18) coincides with (17.2), it is sufficient to prove:

(19) $|(f[n_0] + g[n_0]) - (a + b)| < \delta_0 + \delta_0$.

For proving (19), by (|++|) it suffices to prove

(20) $|f[n_0] - a| < \delta_0 \wedge |g[n_0] - b| < \delta_0$.

We prove the individual conjunctive parts of (20):

Proof of (20.1) $|f[n_0] - a| < \delta_0$:

Because (20.1), we instantiate (5) to:

(21) $\delta_0 > 0 \Rightarrow \exists_N\ \forall_n\ (n \geq N \Rightarrow |f[n] - a| < \delta_0)$.

From (17.1) and (21) we obtain by modus ponens

(22) $\exists_N\ \forall_n\ (n \geq N \Rightarrow |f[n] - a| < \delta_0)$.

By (22) we can take appropriate values such that:

(23) $\forall_n\ (n \geq N_0 \Rightarrow |f[n] - a| < \delta_0)$.

In order to prove (20.1), by (23), it is sufficient to prove:

(24) $n_0 \geq N_0$.

In order to prove (24), by (max), it is sufficient to prove:

(29) $\exists_j\ (n_0 \geq \max[N_0, j])$.

For proving (29) we have to find j_0^* such that:

(30) $n_0 \geq \max[N_0, j_0^*]$.

Formula (30) is proved because it is identical to (12) with the substitution $\{N_0^* \to \max[N_0, j_0^*]\}$.

Proof of (20.2) $|g[n_0] - b| < \delta_0$:

Because (20.2), we instantiate (6) to:

(25) $\delta_0 > 0 \Rightarrow \exists_N\ \forall_n\ (n \geq N \Rightarrow |g[n] - b| < \delta_0)$.

From (17.1) and (25) we obtain by modus ponens

(26) $\exists_N\ \forall_n\ (n \geq N \Rightarrow |g[n] - b| < \delta_0)$.

By (26) we can take appropriate values such that:

(27) $\forall_n\ (n \geq N_1 \Rightarrow |g[n] - b| < \delta_0)$.

In order to prove (20.2), by (27), it is sufficient to prove:

(28) $n_0 \geq N_1$.

In order to prove (28), by (max), it is sufficient to prove:

(31) $\exists_i\ (n_0 \geq \max[i, N_1])$.

For proving (31) we have to find i_0^* such that:

(32) $n_0 \geq \max[i_0^*, N_1]$.

Formula (32) is proved because it is identical to (12) with the substitution $\{N_0^* \to \max[i_0^*, N_1]\}$.

By combining the resulting substitutions we obtain:

$\{\{j_0^* \to N_1,\ i_0^* \to N_0,\ N_0^* \to \max[N_0, N_1]\}\}$

Appendix 2: Rearranged proof

[*identical part deleted*]
(9) $\exists_N \forall_n \, (n \geq N \Rightarrow |(f[n] + g[n]) - (a + b)| < \epsilon_0)$.
We instantiate (eps) to:
(15) $\epsilon_0 > 0 \Rightarrow \exists_\delta \, (\delta > 0 \wedge (\delta + \delta = \epsilon_0))$.
From (8) and (15) we obtain by modus ponens
(16) $\exists_\delta \, (\delta > 0 \wedge (\delta + \delta = \epsilon_0))$.
By (16) we can take appropriate values such that:
(17) $\delta_0 > 0 \wedge (\delta_0 + \delta_0 = \epsilon_0)$.
We instantiate (5) to:
(21) $\delta_0 > 0 \Rightarrow \exists_N \forall_n \, (n \geq N \Rightarrow |f[n] - a| < \delta_0)$.
From (17.1) and (21) we obtain by modus ponens
(22) $\exists_N \forall_n \, (n \geq N \Rightarrow |f[n] - a| < \delta_0)$.
By (22) we can take appropriate values such that:
(23) $\forall_n \, (n \geq N_0 \Rightarrow |f[n] - a| < \delta_0)$.
We instantiate (6) to:
(25) $\delta_0 > 0 \Rightarrow \exists_N \forall_n \, (n \geq N \Rightarrow |g[n] - b| < \delta_0)$.
From (17.1) and (25) we obtain by modus ponens
(26) $\exists_N \forall_n \, (n \geq N \Rightarrow |g[n] - b| < \delta_0)$.
By (26) we can take appropriate values such that:
(27) $\forall_n \, (n \geq N_1 \Rightarrow |g[n] - b| < \delta_0)$.
For proving (9) we have to find N_0^* such that:
(10) $\forall_n \, (n \geq N_0^* \Rightarrow |(f[n] + g[n]) - (a + b)| < \epsilon_0)$.
For proving (10) we take all variables arbitrary but fixed and prove:
(11) $n_0 \geq N_0^* \Rightarrow |(f[n_0] + g[n_0]) - (a + b)| < \epsilon_0$.
We prove (11) by the deduction rule.
We assume
(12) $n_0 \geq N_0^*$
and show
(13) $|(f[n_0] + g[n_0]) - (a + b)| < \epsilon_0$.
In order to prove (13), by (<=<), it is sufficient to prove:
(14) $\exists_v \, (|(f[n_0] + g[n_0]) - (a + b)| < v \wedge (v = \epsilon_0))$.
For proving (14) (using (17.2)) it suffices to prove:
(18) $|(f[n_0] + g[n_0]) - (a + b)| < \delta_0 + \delta_0 \wedge (\delta_0 + \delta_0 = \epsilon_0)$.
Because a part of (18) coincides with (17.2), it is sufficient to prove:
(19) $|(f[n_0] + g[n_0]) - (a + b)| < \delta_0 + \delta_0$.
For proving (19), by (|++|) it suffices to prove
(20) $|f[n_0] - a| < \delta_0 \wedge |g[n_0] - b| < \delta_0$.
We prove the individual conjunctive parts of (20):
Proof of (20.1) $|f[n_0] - a| < \delta_0$:
In order to prove (20.1), by (23), it is sufficient to prove:
(24) $n_0 \geq N_0$.

In order to prove (24), by (max), it is sufficient to prove:

(29) $\exists_j\ (n_0 \geq \max[N_0, j])$.

For proving (29) we have to find j_0^* such that:

(30) $n_0 \geq \max[N_0, j_0^*]$.

Formula (30) is proved because it is identical to (12) with the substitution $\{N_0^* \to \max[N_0, j_0^*]\}$.

Proof of (20.2) $|g[n_0] - b| < \delta_0$:

In order to prove (20.2), by (27), it is sufficient to prove:

(28) $n_0 \geq N_1$.

In order to prove (28), by (max), it is sufficient to prove:

(31) $\exists_i\ (n_0 \geq \max[i, N_1])$.

For proving (31) we have to find i_0^* such that:

(32) $n_0 \geq \max[i_0^*, N_1]$.

Formula (32) is proved because it is identical to (12) with the substitution $\{N_0^* \to \max[i_0^*, N_1]\}$.

By combining the resulting substitutions we obtain:

$\{\{j_0^* \to N_1, i_0^* \to N_0, N_0^* \to \max[N_0, N_1]\}\}$.

Appendix 3: Proof without meta-variables

[*identical part deleted*]
For proving (9) , by taking $\{N \to \max[N_0, N_1]\}$, it suffices to prove:
(10) $\forall_n(n \geq \max[N_0, N_1] \Rightarrow |(f[n] + g[n]) - (a + b)| < \epsilon_0)$.
For proving (10) we take all variables arbitrary but fixed and prove:
(11) $n_0 \geq \max[N_0, N_1] \Rightarrow |(f[n_0] + g[n_0]) - (a + b)| < \epsilon_0$.
We prove (11) by the deduction rule.
We assume
(12) $n_0 \geq \max[N_0, N_1]$
[*identical part deleted*]
For proving (29), by taking $\{j \to N_1\}$, it suffices to prove:
(30) $n_0 \geq \max[N_0, N_1]$.
Formula (30) is proved because it is identical to (12).
[*identical part deleted*]
For proving (31), by taking $\{i \to N_0\}$, it suffices to prove:
(32) $n_0 \geq \max[N_0, N_1]$.
Formula (32) is proved because it is identical to (12).