S©MR ISSN 1813-3304

СИБИРСКИЕ ЭЛЕКТРОННЫЕ
МАТЕМАТИЧЕСКИЕ ИЗВЕСТИЯ

Siberian Electronic Mathematical Reports
http://semr.math.nsc.ru

Том 17, стр. 988–998 (2020) УДК 510.662
DOI 10.33048/semi.2020.17.073 MSC 03B35

PROOF SEARCH ALGORITHM IN PURE LOGICAL FRAMEWORK

D.YU. VLASOV

Abstract. A pure logical framework is a logical framework which does not rely on any particular formal calculus. For example, Metamath (http://metamath.org) is an instance of a pure logical framework. Another example is the Russell system (https://github.com/dmitry-vlasov/russell-flow), which may be considered a high-level language based on Metamath. In this paper, we describe the proof search algorithm used in Russell. The algorithm is proved to be sound and complete, i.e. it gives only valid proofs and any valid proof can be found (up to a substitution) by the proposed algorithm.
Keywords: automated deduction, logical framework.

1. Introduction

Historically, there are several approaches to automated reasoning for logical frameworks. The most popular is LCF [5], where the proofs are generated by programs (tactics), and a user is responsible for programming these tactics and using them in the process of proving a statement [6] - such systems as HOL, Isabelle, etc. use this approach. The other approach was invented and was explored by S. Maslov [7] - so called inverse method. The first applications of the Maslov' method for program verification were attempted 50 years ago [2].

The algorithm, which will be discussed in this paper has nothing in common with LCF methodology, but it shares specific features with the inverse method. Although, strictly speaking, this algorithm is of bottom-up kind (see classification of algorithms in [6], p.172), it has a top-down component, which, in turn, shares some common features with the inverse method. Another method, which has something

Vlasov, D.Yu., Proof search algorithm in pure logical framework.

The work is supported by RFFI (grant 17-01-00531).
Received March, 13, 2019, published July, 20, 2019.

common with our algorithm is Prawitz method [10]. This joint idea is search for a special computable substitution that is able to make combinable several fragments of the proof, while going "downwards", from premises to goal. The resolution method [11] also uses analogical idea (namely unification), but it tries to unify positive and negative entries of a proposition, therefore searching for an inconsistency, instead of compatibility.

The main difference between our algorithm and a top-down approach in [6] consists in locality (i.e. it affects only a currently observed inference transition) of the method that makes it efficient (but still non-trivial)

2. Inference in a Pure Logical Framework

2.1. **Language and Deductive System.** Let's consider a definition of a deductive system $\mathcal{D}$. First of all, let's fix a context free unambiguous grammar G for a language of expressions $L(G)$ with set of non-terminals N. For each $n \in N$ let's designate as G_n a grammar, obtained from G by changing the start non-terminal to n. We need these grammars because the formalized calculus may exploit expressions from different G_n languages. To show the fact, that an expression e belongs to language G_n we will use notation $e : n$. The reason to use this type-style notation (instead of $e \in L(G_n)$) is that it is used in the implementation language. We suppose, that for each non-terminal n there is an infinite set of terminal symbols v, such that the rule $n \to v$ is in G, so that $v : n$. We will call such symbol v a *variable* of type n. Let's denote $\mathrm{var}(e)$ all variables, which occur in an expression e.

The *assertion* has a form $a = \frac{p_1,\ldots,p_n}{p_0}$ where $p_1, \ldots, p_n$ are *premises* and p_0 is a *proposition* of a. Assertion with $n = 0$ is called an *axiom*, otherwise it is called an *inference rule*. The *deductive system* $\mathcal{D}$ is a pair: $\mathcal{D} = (G, A)$ where G is a grammar of expressions and A is a set of axiomatic assertions of $\mathcal{D}$ (axioms and rules of inference).

2.1.1. *Remark on terminology.* In the further text the terminology is inspired by a Metamath formal system, where a notion of an "assertion" or "statement" corresponds to the standard notion of an "inference rule" (axiomatic or admissible), an "axiomatic assertions" corresponds to the standard notion of an "axiom", and a "provable assertion" to a "derivable rule" or "theorem".

2.2. **Matching.** Given a finite set of variables $V = \{v_i : n_i | i < k\}$ and a set of expressions $E = \{e_i : m_i | i < k\}$, a mapping $\theta : V \to E$ is called a *substitution*, iff for any $i < k$ the rule $n_i \to m_i$ is derivable in grammar G. This means, that in grammar G you may substitute non-terminal n with non-terminal m, and, therefore, with the expression e. For example, in a formal set theory that admits *classes* and *sets*, a substitution of a set instead of class should be legal and valid but not vice versa.

Application of a substitution θ to an expression e is straightforward and is designated as $\theta(e)$. Note, that application of substitution demands a conformity of variable types. Further we will assume, that all application of substitution are type-correct. By definition of substitution, if e is an expression from $L(G)$, and θ is applicable to e then $\theta(e)$ also stays in $L(G)$ - this follows from the context freedom of G. The application of substitution to a complex object (like assertion or proof) is understood component-wise. There is a natural notion of *composition* of substitutions, which we will denote as $(\theta \circ \eta)(e) = \theta(\eta(e))$.

An expression e_1 *matches* with an expression e_2, iff there is a substitution θ such that $e_2 = \theta(e_1)$. Such substitution is called a *matching* of e_1 with e_2. The consequence of grammar unambiguity is that if a matching exists, it is unique.

2.3. **Inference.** A *pseudo-inference tree* in a deduction system $\mathcal{D} = (G; A)$ is a finite tree, which nodes are labeled: of odd depth with deduction rules from A(a-nodes) and of even depth with expressions from $L(G)$ (e-nodes) and any e-node has at most one descendant a-node. Also we demand that leafs must be only e-nodes and a-nodes are labeled with substitutions, so actually a label of an a-node is a pair: (a, θ) - an assertion and a substitution. Note, that here we don't demand validity: pseudo-inference may be invalid.

Two pseudo-inference trees T_1 and T_2 are *congruent*, iff their graph structures are isomorphic, corresponding a-nodes have the same assertions as labels, and no restrictions on e-nodes. Let's designate this relation as $T_1 \sim T_2$.

Given two pseudo-inference trees T_1 and T_2, we say that T_1 is *more general* then T_2 (respectively, T_2 is *less general* then T_1), iff there exist such substitution δ, that $T_2 = \delta(T_1)$ (recall, that application of substitution to complex structures is component-wise).

An *inference tree* is a pseudo-inference tree, if for any transition from e-nodes $e_1, \dots e_n$ via a-node a to e-node e_0 there exists a substitution θ such that $\frac{e_1,\dots e_n}{e_0}$ is equal to $\theta(a)$ (where θ is called a witness of this transition).

An assertion $s = \frac{p_1,\dots p_n}{p_0}$ is said to be derivable (in the system (G, A)), if there exist an inference tree with the root p_0 where the set of leaves is $\{p_1, \dots p_n\}$.

Note, that since the main goal of this system is to show the derivability of assertions, which may have premises, we don't separate the class of derivable assertions, which leafs are axioms.

Lemma 1 (Monotonicity)**.** *Let πbe a proof of a derivable assertion s and θ — any substitution of variables, which occur in s. Then $\theta(\pi)$ is a the proof of the derivable assertion $\theta(s)$.*

Proof. It is sufficient to notice, that application of a substitution to each proof transition keeps matching: if e_0 is deduced from $e_1, \dots, e_n$ in the proof π with assertion b and substitution η is its witness, then $\theta(e_0)$ is deduced from

$$\theta(e_1), \dots, \theta(e_n)$$

with the same a and substitution $\theta \circ \eta$ is its witness. □

The notion of a deductive system, presented here is very close to the notion of canonical deductive system given by Post [9]. The difference is in the treatment of expressions: Post's canonical system doesn't constrain the set of legal substitutions. Here we restrict the language to context-free unambiguous class, so that efficient unification/matching algorithms are possible. Also, monotonicity property, which we need to prove correctness, is proved only for the context-free grammars. Unambiguity of grammar is necessary for the uniqueness of a matching and unifier (up to renaming).

3. Proof Search Algorithm

Suppose that we have a deductive system $\mathcal{D}$ and we want to answer the question: is some particular statement s provable in $\mathcal{D}$? Let's note, that there is no symmetry

in asking for provability and non-provability of s in general case, because it is possible to show provability by giving actual proof and checking it, but it is not possible to assert that something is not provable - we might not have a negation in the deductive system (once more let's stress here that we are speaking about the general case, for a particular decidable calculus it may be wrong).

From the general considerations, while searching for a proof for the statement $s = \frac{p_1,\ldots,p_n}{p_0}$ we may follow different strategies:

(1) start with premises $p_1, \ldots, p_n$ make various inferences and try to obtain the goal p_0 (downwards approach)
(2) start with the goal p_0, look for all possible ways how it can be obtained in $\mathcal{D}$, get the sub-goals $q_1, \ldots, q_m$ and do the same for them, until we come up to premises (upwards approach)

The outer loop of the algorithm uses the second variant - upwards search, from conclusion to premises, but inside of it there is a top-down loop. So, in a very general sense, the proposed algorithm uses both proof-search strategies: bottom-up and top-down, but they are not equal and play different roles. Namely, upwards pass is a traversal of possible variants to derive a goal, while downwards pass is a quest for valid consequences of premises which uses the structure of a tree, which is built during the upwards pass. When we reach the root on the downwards pass, then the target statement is proved. Summarizing the above, the proposed algorithm uses a mixture of top-down and bottom-up strategies.

3.1. **Proof Variant Tree.** The proof search algorithm essentially is construction of a tree of proof variants - so called *proof variant tree* (PV-tree). The completeness of the algorithm is guaranteed by the completeness of the tree of variants. The PV-tree nodes are marked with expressions (nodes of even depth) and assertions (nodes with odd depth) - just like proof trees. The variables of e-nodes are marked up with replaceable/non-replaceable flags. It is necessary because some variables are passed from the statement, which is being proved, so they cannot be replaced or modified, therefore they are marked as non-replaceable; while others come from internal expressions of a proof and may be substituted with arbitrary expressions. The starting point of the tree construction algorithm is a goal expression p_0, all of its variables are marked as non-replaceable and these variables will stay non-replaceable while tracing further into the PV-tree.

Given a node of even depth, which is marked up with an expression e, we fork it out with nodes, marked up with all assertions $\{a_1, \ldots, a_n\}$ where the conclusion is unifiable with e with some substitution θ. In turn, for each of odd-depth node $a = \frac{q_1,\ldots,q_n}{q_0}$ and appropriate substitution θ, its premises $\theta(q_1), \ldots, \theta(q_n)$ form a set of direct descendants of a. Almost for sure there will be a collision of variable names at this step (except for a very rare occasions), so to avoid it we will accept an agreement that while matching a with e, we will replace all variables of a, which are not substituted with θ, with new fresh ones. Precedence is a binary relation on PV-trees which is designated as $n \succ m$: here n is a direct descendant of m.

For any subtree of an PV-tree we say that it is a *proof variant*, iff any e-node in it has at most one descendant. Any proof variant v immediately generates a pseudo-inference tree $\pi(v)$, when we remove all unrelated data from it.

Lemma 2. *Any proof variant π is more general, then any other pseudo-inference tree π', which is congruent to π and has the same root node.*

Proof. By induction on the depth of π. The base is trivial: when the depth is 1, then both trees are root nodes, which coincide. Step of induction. Let's consider some highest inference step in π and π', both due to some assertion $a = \frac{p_1,\ldots,p_n}{p_0}$. Corresponding e-nodes are $\frac{e_1,\ldots,e_n}{e_0}$ and $\frac{e'_1,\ldots,e'_n}{e'_0}$ for π and π' correspondingly (here e_i and e'_i are leafs of corresponding trees). By induction there is a substitution δ such that $\delta(e_0) = e'_0$. By construction of π and π' there are substitutions θ and θ' such that $e_i = \theta(p_i)$ and $e'_i = \theta'(p_i)$ for all $0 \le i \le n$. To prove the step of induction we need to extend δ to such substitution $\varepsilon \supseteq \delta$ that $\varepsilon(e_i) = e'_i$ for all i. For further considerations let's introduce following sets of variables: $V_0 = \mathrm{var}(e_0)$ and $V_1 = (\bigcup_{1\le i\le n} \mathrm{var}(e_i)) \setminus V_0$; $W_0 = \mathrm{var}(p_0)$ and $W_1 = (\bigcup_{1\le i\le n} \mathrm{var}(p_i)) \setminus W_0$. First of all let's notice, that any variable $x \in V_1$ is an image of some variable $y \in W_1$ by θ, because when we construct PV-tree, we introduce fresh variables for all variables from a, which are not touched by matching with assertion proposition. So, for those variables $x \in V_1$ the inverse mapping $\theta^{-1}(x) = y$ is defined. Let's define the substitution ε point-wise (variable-wise):

$$\varepsilon(x) = \begin{cases} \delta(x), & \text{if } x \in V_0 \\ \theta'(\theta^{-1}(x)), & \text{otherwise, i.e. } x \in V_1 \end{cases}$$

Now let's check, that $\varepsilon(e_i) = e'_i$ for all $i > 0$ (the case $i = 0$ follows from $\delta \subseteq \varepsilon$). If $\varepsilon(e_i) \ne e'_i$ for some i, then $(\varepsilon \circ \theta)(p_i) \ne \theta'(p_i)$. Then there is some $x \in W_0 \cup W_1$ such that $(\varepsilon \circ \theta)(x) \ne \theta'(x)$. If $x \in W_0$, then $(\varepsilon \circ \theta)(x)$ is a subexpression of e_0, and on variables of e_0 subsitutions ε and δ act the same, so $(\varepsilon\circ\theta)(x) = \delta(\theta(x)) \ne \theta'(x)$ and $\delta(e_0) \ne e'_0$ - contradiction. In case when $x \in W_1$ we know, that $\theta(x) = y$ for some $y \in V_1$, so by definition of ε we have $(\varepsilon \circ \theta)(x) = \varepsilon(y) = \theta'(\theta^{-1}(y)) = \theta'(x)$. □

3.2. **Substitution Pseudo-Inference Tree.** The nodes in PV-tree are marked not only by the expressions and assertions. Each node n in PV-tree has a set of its *substitution pseudo-inference trees*, which is designated as $s(n)$. *Substitution pseudo-inference tree* (SPI-tree) T is a pseudo-inference tree, which nodes are labeled with the nodes of PV-tree and substitutions. The substitution of a root node will be addressed as $\theta(T)$.

Initially, when created, the set of SPI-tree for any PV-tree node is empty. Let's consider some just created PV-tree e-node e. We look at the premises $p_1, \ldots, p_n$ of a statement, which is proved. If some p_i of these premises matches with e (note, that here variables in e are non-replaceable!), then e is trivially provable from p_i. So we place the one-node SPI-tree, constructed from the matching substitution and current PV-tree node, into the set of SPI-tree for this node.

If we find a new SPI-tree node for some e-node, then we try to shift it a step down to the root. For this purpose we test all of its siblings (they correspond to the premises of some assertion) for being also proved (i.e. the set of SPI-tree is non-empty). If we find, that all siblings of the node are proven, we can try to go a step further to the root and find an SPI-tree node for its ancestor. To do it we need a concept of unification of substitutions.

3.2.1. *Unification of Substitutions.* Two substitutions θ_1 and θ_2 are called *compatible*, iff $\forall x \in \mathrm{dom}(\theta_1) \cap \mathrm{dom}(\theta_2)\ \theta_1(x) = \theta_2(x)$. This relation is designated as $\theta_1 \sim_c \theta_2$. It is obvious, that relation $\sim_c$ is an equivalence and $\theta_1 \sim_c \theta_2 \Leftrightarrow \theta_1 \cup \theta_2$ is a substitution

Given a set of substitutions $\Xi = \{\theta_1, \ldots, \theta_n\}$, we say that a substitution δ is *a unifier* for Ξ, iff for all $i, j \le n$ we have $\delta\circ\theta_i \sim_c \delta\circ\theta_j$. Unifier δ is called *most general*,

iff for any other unifier η for the set Ξ, there is such η' that $\eta = \eta' \circ \delta$. For each set Ξ, if a unifier for Ξ exists, there is a unique up to the variable renaming most general unifier, which we will designate as $\mathrm{mgu}(\Xi)$. And the common substitution $\bigcup_i \mathrm{mgu}(\Xi) \circ \theta_i$ will be designated as $\mathrm{com}(\Xi)$

3.2.2. *Building SPI-tree for Assertion Nodes.* So, imagine that we have some a-node a in the proof variant tree, and all of its direct descendants $e_1, \ldots, e_2$ have non-empty sets of SPI-tree $s(e_1), \ldots, s(e_n)$. Then for any tuple of SPI-tree $T_1 \in s(e_1), \ldots, T_n \in s(e_n)$, if the set of substitutions $\{\theta(T_1), \ldots, \theta(T_n)\}$ is unifiable, and δ is the most general unifier for it (i.e

$$\delta = \mathrm{mgu}(\theta(T_1), \ldots, \theta(T_n))$$

) then a new SPI-tree T_0 with substitution $\theta = \mathrm{com}(\theta(T_1), \ldots, \theta(T_n))$ is added to $s(a)$. The tree of T_0 is obtained as:

$$T_0(T_1, \ldots, T_n) = \frac{\delta(T_1) \ \ldots \ \delta(T_n)}{(\theta, a)}$$

Note, that unifier δ, obtained at this step, propagates through the whole trees T_i (applies to all of its components: expressions and substitutions). Also, non-replaceable variables cannot be substituted with at this step. So, the set of all SPI-tree for the node a will be:

$$s(a) = \{T_0(T_1, \ldots, T_n) | T_1 \in s(e_1), \ldots, T_n \in s(e_n), \exists \mathrm{mgu}(\theta(T_1), \ldots, \theta(T_n))\}$$

3.2.3. *Building SPI-tree for Expression Nodes.* The set of SPI-tree for the e-node e is updated at each update of SPI-tree set of any of its descendants. For an e-node e, if one of its descendants is updated with the SPI-tree s, then the set $s(e)$ is also updated with a new node, which only descendant is s and substitution coincides with the substitution of a descendant:

$$T_0(T_1) = \frac{T_1}{(\theta, e)}$$

and a set of all SPI-tree nodes for the node e will be:

$$s(e) = \bigcup_{a \succ e} \{T_0(T_1) | T_1 \in s(a)\}$$

Lemma 3. *Each substitution pseudo-inference tree T defines a unique proof variant $\pi(T)$.*

Proof. Induction on the depth of T. The base is trivial: when we match some expression with a premise of a proven assertion, it clearly generates a proof variant. Step of induction comes from the definition of SPI-tree for e-nodes: each SPI-tree e-node has at most one direct descendant. □

Theorem 1 (Soundness)**.** *For any substitution pseudo-inference tree T with root (θ, e) the tree $\pi(T)$ is an inference tree of $\theta(e)$.*

Proof. Let's prove it by induction on the depth of T. The base of induction is trivial: we have no obligations on leafs except for them to be e-nodes.

Let's assume that for some assertion node $a = \frac{q_1, \ldots, q_m}{q_0}$ it has an SPI-tree node T_0 with substitution θ_0 and $T_1, \ldots, T_m$ are direct descendants of T_0. Let e_0 be a unique ancestor of a in the PV-tree. Then, by definition, for substitutions $\theta_1, \ldots, \theta_m$, corresponding to $T_1, \ldots, T_m$ we have that the set $\{\theta_1, \ldots, \theta_m\}$ has a

unifier δ. By induction, all T_i induce proofs $\pi_1(T_1), \ldots, \pi_m(T_m)$ for expressions $\theta_1(e_1), \ldots, \theta_m(e_m)$. By the definition of unifier of substitutions, for all $1 \leq i, j \leq m$ we have that $\delta \circ \theta_i \sim \delta \circ_c \theta_j$ and by construction of T_0:

$$\theta_0 = \bigcup_{1 \leq i \leq m} \delta \circ \theta_i$$

Also there is a substitution η such that $\eta(q_0) = e_0$ and $\eta(q_j) = e_j$ for all $j \leq m$. Let's consider a substitution $\eta' = \theta_0 \circ \eta$:

$$\eta'(q_0) = (\theta_0 \circ \eta)(q_0) = \theta_0(\eta(q_0)) = \theta_0(e_0)$$

$$\eta'(q_i) = (\theta_0 \circ \eta)(q_i) = \theta_0(\eta(q_i)) = (\delta \circ \theta_i)(\eta(q_i)) = (\delta \circ \theta_i)(e_i) = \delta(\theta_i(e_i))$$

By monotonicity lemma $\delta(\pi(s_i))$ will be a proof of $\delta(\theta_i(e_i))$, so η' is a witness for the observed transition in the proof. □

Lemma 4. *Let's consider disjoint SPI-tree trees T_1 and T_2, with roots e_1 and e_2 correspondingly. Then* $\mathrm{var}(T_1) \cap \mathrm{var}(T_2) = \mathrm{var}(e_1) \cap \mathrm{var}(e_2)$.

Proof. First of all, let's notice, that this property holds for PV-tree trees. Indeed, if T_1 and T_2 are two PV-tree with roots e_1 and e_2, then when we build T_1 and T_2 up from e_1 and e_2 correspondingly, then at each expansion step variables are kept or new fresh variables are introduced. When we construct SPI-tree, then substitutions, which affect nodes of SPI-tree, are constructed only from variables from the upper part of the tree, so, this property is hereditarily kept. □

Lemma 5. *Let's consider two pseudo-inference trees T and T', with roots e and e' correspondingly, three substitution θ, α and β such that $\theta(T) = T'$, $(\alpha \circ \beta)(e) = e'$ and require, that*

(1) $\mathrm{dom}(\beta) \cap \mathrm{dom}(\theta) \subseteq \mathrm{var}(e)$
(2) $\mathrm{var}(\mathrm{im}(\beta)) \cap \mathrm{dom}(\theta) \subseteq \mathrm{dom}(\beta)$
(3) $\mathrm{dom}(\alpha) \cap \mathrm{dom}(\theta) \subseteq \mathrm{var}(\mathrm{im}(\beta))$

Then there is a substitution σ such that $\theta = \sigma \circ (\beta \restriction_{\mathrm{dom}(\theta)})$.

Proof. Let's denote $X_0 = \mathrm{dom}(\beta) \cap \mathrm{dom}(\theta)$ and $X_1 = \mathrm{var}(\mathrm{im}(\beta)) \cap \mathrm{dom}(\theta)$. Let's notice, that $\theta(e) = e' = \alpha(\beta(e))$. Then by 1 for all $x \in \mathrm{var}(e)$ we have $\theta(x) = \alpha(\beta(x))$, so by 2 and 3 we obtain that $(\alpha \circ \beta) \restriction_{X_0} \subseteq \theta$. Then we can decompose θ as $\theta = (\alpha \circ \beta) \restriction_{X_0} \sqcup \theta_1$, where $\mathrm{dom}(\theta_1) \cap X_0 = \emptyset$. Then it is sufficient to take $\sigma = \theta_1 \cup \alpha \restriction_{X_1}$. In fact, we have $(\theta_1 \cup \alpha \restriction_{X_1}) \circ (\beta \restriction_{X_0}) = \theta_1 \cup (\alpha \circ \beta) \restriction_{X_0} = \theta$. □

Theorem 2 (Dominance)**.** *If the algorithm finds a substitution pseudo-inference tree T for the root of PV-tree for some assertion $a = \frac{p_1, \ldots, p_n}{p_0}$, then $\pi(T)$ is more general then any proof π' of a, congruent to $\pi(T)$.*

Proof. By the correctness lemma we have that $\pi(T)$ is a proof of a. Now let's check that $\pi(T)$ is the most general proof of a. Let π' be another proof of a, congruent to $\pi = \pi(T)$. By lemma 2 we have, that underlying PV-tree P of T is more general, then π', so there is a substitution ε_1 such that $\varepsilon_1(P) = \pi'$.

We prove, that $\pi(T)$ is more general then π' by induction on the depth of T. The base of induction is obvious: leaf nodes of T are premises of a, therefore they are the same for π and π'. The step of induction. Let's consider the lowest (root) inferences $\frac{e_1, \ldots, e_m}{e_0}$ and $\frac{e'_1, \ldots, e'_m}{e'_0}$ from π and π' accordingly. Inference step $\frac{e_1, \ldots, e_m}{e_0}$ in T is labeled with substitution $\theta_0 = \theta(T)$. By construction of SPI-tree, we can

consider all direct SPI-tree descendants $T_1, \ldots, T_m$ of the root of T, and their root expressions e_i''. Then expressions e_i'' are obtained from corresponding nodes of PV-tree tree P with expressions e_i''' by applying substitutions $\theta_i = \theta(T_i)$: $e_i'' = \theta_i(e_i''')$. By construction of SPI-tree there is a most general unifier $\delta = \mathrm{mgu}(\theta_1, \ldots, \theta_m)$ and $\theta_0 = \mathrm{com}(\theta_1, \ldots, \theta_m)$.

By induction, for each $1 \le i \le 0$ there is a substitution ε_2^i such that $\varepsilon_2^i(T_i)$ is a subtree of π' with root e_i' and $\varepsilon_2^i(e_i) = e_i'$. Let's check, that $\varepsilon_2 = \bigcup_{1 \le i \le m} \varepsilon_2^i$ is a mapping, i.e. it is a substitution. It's sufficient to check, that $\varepsilon_2^i \sim_c \varepsilon_2^j$, i.e. for any $x \in \mathrm{dom}(\varepsilon_2^i) \cap \mathrm{dom}(\varepsilon_2^j)$ we have $\varepsilon_2^i(x) = \varepsilon_2^j(x)$. Let's analyze the origin of the variable x. It may come from expressions e_α''', $\alpha \in \{i, j\}$. Then $\varepsilon_1(x)$ is defined and then $\varepsilon_2^i(x) = \varepsilon_1(x) = \varepsilon_2^j(x)$, because $\varepsilon_1(e_\alpha''') = e_\alpha' = (\varepsilon_2^\alpha \circ \theta_\alpha)(e_\alpha''')$ for indexes $\alpha \in \{i, j\}$. The other possible origin of x is $\mathrm{var}(\mathrm{im}(\theta_\alpha))$, and by construction of SPI-tree we have that $\mathrm{var}(\mathrm{im}(\theta_\alpha)) \subseteq \mathrm{var}(T_\alpha)$. By lemma 4, in this case $x \in \mathrm{var}(T_i) \cap \mathrm{var}(T_j)$, so it comes from expressions e_α''', $\alpha \in \{i, j\}$ and we return to the previous case.

Let's check, that ε_2 is a unifier for substitutions θ_i, i.e. that:

$$\varepsilon_2 \circ \theta_i \sim_c \varepsilon_2 \circ \theta_j, \quad \text{for all } 1 \le i, j \le m.$$

Indeed, if $x \in \mathrm{var}(e_i'') \cap \mathrm{var}(e_j'')$ (i.e. $x \in \mathrm{dom}(\theta_i) \cap \mathrm{dom}(\theta_j)$), then $\varepsilon_2 \circ \theta_i(x) = \varepsilon_2 \circ \theta_j(x)$, because

$$(\varepsilon_2 \circ \theta_\alpha)(e_\alpha'') = e_\alpha' = \varepsilon_1(e_\alpha''), \quad \alpha \in \{i, j\}$$

so for x we have $\varepsilon_2 \circ \theta_i(x) = \varepsilon_1(x) = \varepsilon_2 \circ \theta_j(x)$.

Now let's recall, that δ is a most general unifier for $\theta_1, \ldots, \theta_m$, so there exists such ε_3 that $\varepsilon_2 = \varepsilon_3 \circ \delta$.

Then for each $1 \le i \le m$ we have

$$\varepsilon_3(e_i) = \varepsilon_3(\delta(e_i'')) = \varepsilon_3(\delta(\theta_i(e_i'''))) = e_i' = \varepsilon_1(e_i''')$$

Let's designate two sets of variables: $V_1 = \bigcup_{1 \le i \le n} \mathrm{var}(e_i''')$ and $V_0 = \mathrm{var}(e_0''') \setminus V_1$. Recall that $\theta_0 = \bigcup_{1 \le i \le m} \delta \circ \theta_i$. Then $\varepsilon_3(\theta_0(e_i''')) = \varepsilon_1(e_i''')$, so for any $x \in V_1$ we have that $\varepsilon_3(\theta_0(x)) = \varepsilon_1(x)$.

Now let's define a final substitution $\varepsilon_4 = \varepsilon_3 \cup \varepsilon_1 \restriction_{V_0}$. It is obvious, that for any $x \in V_0$ we have $\varepsilon_4(\theta_0(x)) = \varepsilon_1(x)$ because in this case $x \notin \mathrm{dom}(\theta_0)$, and for any $x \in V_1$ previously we got that $\varepsilon_4(\theta_0(x)) = \varepsilon_3(\theta_0(x)) = \varepsilon_1(x)$. So, $\varepsilon_4 \circ \theta_0 = \varepsilon_1$, and finally we get:

$$\varepsilon_4(e_0) = \varepsilon_4(\theta_0(e_0''')) = \varepsilon_1(e_0''') = e_0'$$

For all other expressions $e_i, 1 \le i \le m$ we have, that $\mathrm{var}(e_i) \cap V_0 = \emptyset$, so

$$\varepsilon_4(e_i) = \varepsilon_3(e_i) = e_i'$$

The last thing which is left to complete the proof is to show, that there is a substitution ε_2', which maps trees $\delta(T_i)$ onto corresponding subtrees of π'. To prove this, we use lemma 5, where we take $\varepsilon_4 \circ \delta$ as $\alpha \circ \beta$, and for that we need to check conditions 1-3 from lemma 5:

(1) $\mathrm{dom}(\delta) \cap \mathrm{dom}(\varepsilon_2^i) \subseteq \mathrm{var}(e_i'')$
(2) $\mathrm{var}(\mathrm{im}(\delta)) \cap \mathrm{dom}(\varepsilon_2^i) \subseteq \mathrm{dom}(\delta)$
(3) $\mathrm{dom}(\varepsilon_4) \cap \mathrm{dom}(\varepsilon_2^i) \subseteq \mathrm{var}(\mathrm{im}(\delta))$

The first two conditions hold because δ operates on variables $V = \bigcup_{1 \le i \le m} \mathrm{var}(e_i'')$ and lemma 4 is true, the third one holds because the only variables from $\mathrm{dom}(\varepsilon_4)$, which are not present in V, may come from variables of e_0, which doesn't occur in $\bigcup_{1 \le i \le m} \mathrm{var}(e_i)$, but these variables can't fall into any set $\mathrm{dom}(\varepsilon_2^i)$. □

Theorem 3 (Completeness)**.** *If a statement* $\frac{p_1,\ldots,p_n}{p_0}$ *has a proof* π*, then the set* $s(p_0)$ *of SPI-trees for the root* p_0 *at some moment of construction PV-tree will contain some SPI-tree* T*, such that* $\pi(T)$ *is more general then* π*.*

Proof. By previous theorem it is sufficient to show, that at some moment, the set $s(p_0)$ will contain an SPI-tree congruent to π. But it is clear from the character of the algorithm: at each step of expansion of PV-tree, we use *all possible* variants of expansion (limited by demand of matching), so at some moment we will obtain all nodes, corresponding to the proof π. □

Corollary 1. *The set of all provable assertions in any pure deductive system is computably enumerable.*

As it already was mentioned, the algorithm has two different aspects: bottom-up and top-down. The bottom-up procedure (construction of a PV-tree) is quite straightforward. The other one, top-down (construction SPI-trees), is more sophisticated, and unification of substitutions is a crucial part of it. Maslov' inverse method (mentioned in the Introduction), also uses analogical procedure, but, surprisingly, the substitution unification (that is called substitution combination in [1] a *combination* of substitutions) is not considered as the main operation but just one in a row of other complex formula transformations.

4. Conclusion and Future Work

The algorithm, presented in this paper, is extremely general, yet it allows practical implementation. The generality of the underlying formal system in almost maximal, because in comparison with the general notion of Post canonical system, it is restricted only by the language: language has to be context free and unambiguous. There are no other constraints like subformula property, which is vital for the inverse method. And the restriction of grammar to context-free and unambiguous class is natural: if you don't impose it, then there immediately arises a question about unification/matching algorithms for the language which is used.

Another good feature/property of the method, presented in this paper, is that it is completely ready for use out-of-the-box, and you don't need to "cook" [3] a considered logic in order to use it - just write down expression language, axioms and inference rules and you may feed the assertion of interest to a prover engine, which, in theory, will find a proof (if it exists).

What is left out of scope of this article is unification/matching problem. From the algorithm description it is clear, that efficient unification and matching algorithms are vital for the implementation of this method. And efficient matching of an expression with a (potentially huge) set of assertions is usually done with indexing [4] and is not trivial. The algorithm of unification for substitutions is even more complex and challenging. Experiments on proving a rather simple statement in classical propositional Hilbert-style logic (with cut rule) showed, that the number of SPI-trees may grow extremely fast. Just to feel the scale of this problem imagine, that we have an assertion with 5 premises (common case in Metamath theorem

base), each of which has a non-empty set of SPI-trees, having, for example 1, 10, 100, 10 and 100 elements respectively. Then we need to check 10^6 substitution tuples for unification. In experiment, fortunately, majority of these tuples have been found to be non-unifiable, so we will end up with something like 500 (or even 0) of solutions, but still, checking all of these 10^6 variants consequently is not affordable in practice. Efficient algorithm for such massive substitution unification was developed, but it needs a thorough analysis and separate research.

The other thing, which is intentionally missed in this paper, is treatment of proper substitutions for disjointed variables. Classical predicate calculus has special restrictions on substitutions, which may be applied to specific rules of inference (like introduction of $\forall$-quantifier). In Metamath such restrictions are simplified, but still are essentially a restriction on application of particular substitutions. Addition of such restrictions doesn't change the general scheme of algorithm, the only thing, which is necessary to track during traversing of PV-tree are these restrictions, which are not difficult to check. So, in order to keep text more simple and clear we decided to skip this details.

The problematic part of practical implementation of such method is computational complexity. The strong side of this method is its universality and ability to apply to the wide variety of calculi. And, as always, this universality causes problems. For example, we cannot rely on good properties of a considered deductive system: it may have a cut-like rule(s), no subformula property, etc. In practice this leads to the enormous growth of a search space while searching for a proof. The only way to cope with such combinatorial explosion is to use smart heuristics, which will lead search in the right direction. The author's strong belief is that the advanced methods of machine learning, based on the analysis of an already formalized proofs, may help to develop such methods.

References

[1] C.L. Chang, R.C. Lee, *Symbolic Logic and Mechanical Theorem Proving*, Academic Press, New York-London, 1973. Zbl 0263.68046

[2] V. Davydov, S. Maslov, G. Mints, V. Orevkov, A. Slissenko, *A computer algorithm of establisihing deducibility based on the inverse method*, Zap. Nauchn. Semin. LOMI, **16** (1969), 1–16. Zbl 0198.03602

[3] A. Degtyarev, A. Voronkov, *The Inverse Method*, in: *Handbook of Automated Reasoning, vol. 1, ch. 4*, Elsevier, Amsterdam, 2001, 179–272. Zbl 0992.03016

[4] I.V. Ramakrishnan, R.Sekar, A. Voronkov, *Term indexing*, in: *Handbook of Automated Reasoning, vol. 2, ch. 26*, Elsevier, Amsterdam (2001), 1853–1964. Zbl 0992.68189

[5] M.J.C. Gordon, *Representing a logic in the LCF metalanguage*, in: Tools and Notions for Program Construction: an Advanced Course, Cambridge University Press, (1982), 263–185.

[6] J. Harrison, *Handbook of Practical Logic and Automated Reasoning*, Cambridge University Press, Cambridge, (2009). Zbl 1178.03001

[7] S. Maslov, *An inverse method of establishing deducibility in classical predicate calculus*, Sov. Math. Dokl., **5** (1965), 1420–1424. Zbl 0146.24603

[8] N. Megill, *Metamath: A Computer Language for Pure Mathematics*, Lulu press, Morrisville, North Carolina, 2007.

[9] E. Post, *Formal Reductions of the General Combinatorial Decision Problem*, Am. J. Math., **65**:2 (1943), 197–215. Zbl 0063.06327

[10] D. Prawitz, H. Prawitz, N. Voghera, *A mechanical proof procedure and its realization in an electronic computer*, J. ACM, **7**:2 (1960), 102–128. Zbl 0213.02401

[11] J.A. Robinson, *A machine-oriented logic based on the resolution principle*, J. ACM, **12**:1 (1965), 23–41. Zbl 0139.12303

Dmitry Yuriecvich Vlasov
Sobolev Institute of Mathematics,
4, Koptyuga ave.,
Novosibirsk, 630090, Russia
Email address: vlasov@math.nsc.ru