Evidence Algorithm Approach to Automated Theorem Proving and SAD Systems
(In Honor of 50 Years of Evidence Algorithm Announcement)

Alexander Lyaletski, Alexandre Lyaletsky

National University of Life and Environmental Sciences, Heroiv Oborony str. 15, Kyiv, 03041, Ukraine

Abstract
In 1970, Academician V.M. Glushkov published a paper, in which he, along with a discussion of some of the problems of artificial intelligence, formulated a research programme called **Evidence Algorithm** (**EA)**, containing his vision of the problem of a computer-aided support of a human in theorem proving in mathematics. He suggested to make a research on the creation of an automated theorem-proving system with a formal language of a natural type as its input language, with a proof search procedure constructed on the notion of a computer evidence of a machine-made proof step developing in accordance with a knowledge accumulated by the system during its operation, and with the possibility of a human to make intervention in the proof search process. Two attempts have been made in order to implement this programme. The first led to the appearance in 1978 of the Russian-language automated theorem-proving system, denoted by **RuSAD**, and the second – to the appearance in 2002 of the English-language system for automate deduction, denoted by **EnSAD**. And if the development and trial operation of **RuSAD** were stopped in 1992 after the output from service of the ES-line computers, **EnSAD**, after its placement on the website "nevidal.org", is still online available now. This work is mainly devoted to the description of the approaches to the creation and realization of the evidential "engines" of both **RuSAD** and **EnSAD** – a topic insufficiently covered in publications on **EA**, **RuSAD**, and **EnSAD**.

Keywords
V.M. Glushkov, Evidence Algorithm, automated theorem proving, deduction, classical first-order logic, sequent calculus, resolution method.

Continuous improvement of the evidence algorithm will lead, sooner or later, to the fact that all the theorems that we know today will become evident from the point of view of the machine. During this period, the role of a mathematician will consist mainly in defining new concepts and in formulating a fundamentally new proposition and the art of proving a new machine-nonevident theorem will consist primarily in the ability to formulate a number of intermediate theorems and lemmas, each of which will be evident to the machine.

V.M. Glushkov, [1], 1970

1. Introduction

The first publications devoted to the possibility of using computers for making logical constructions, relate to the late 1950's - early 1960's (see, for example, [2] and [3]), when there were appeared computers of such a performance, memory capacity, and informational flexibility that the programming of complex intelligent processes has become possible. An early history of the

IT&I-2020 Information Technology and Interactions, December 02–03, 2020, KNU Taras Shevchenko, Kyiv, Ukraine
EMAIL: a.lyaletski@nubip.edu.ua (A. Lyaletski); alyaletsky@yahoo.com (A. Lyaletsky)
ORCID: 0000-0003-0370-5041 (A. Lyaletski); 0000-0002-0389-2759 (A. Lyaletsky)

development of this branch of computer science in Western countries can be found in [4]. As for the USSR and the countries of Eastern Europe, two schools of automated theorem proving appeared in the USSR almost simultaneously in the 1960s: one created by Academician V.M. Glushkov and the other in Leningrad, for the description of which we refer to [5] (see also [6]).

In 1962, V.M. Glushkov first spoke about the possibility of a computer to make intelligent processing of information in mathematics. Soon there was formed the first team of researchers on automated theorem proving. It was active from 1962 to 1969 and its precise subject was an analysis of proofs in Group Theory. During its research, the first steps were made in the framework of the **Evidence Algorithm** programme: there were constructed both a language for writing mathematical texts and a procedure for logical inference search. The last was later transformed into the so-called (sequent) calculus of auxiliary goals (**AG** calculus) [7], partially satisfying the following **EA** requirements: (i) an **EA**-style calculus should preserve the structure of an original problem, (ii) deduction in it should be separated from equality handling, (iii) it should goal-oriented, (iv) a proof search in it was should carried out in the signature of an original theory (for the achievement of this, in **AG** instead of skolemization, the Kanger notion [8] of the admissibility of a substitution of terms for variables was used and this made it possible to increase the efficiency of a logical search in **AG** in comparison with calculi using the ordinary Gentzen notion of an admissible substitution [9])[2].

After joining a new, second team of investigators to research on **EA** in 1971, a new stage began in the implementation of **EA**. As a result, the Russian-language system for automated deduction (here called Russian **SAD** and denoted by **RuSAD**) was constructed in accordance with the **EA** theses and it was put into trial operation in 1978. At that time span, the formal natural language **TL** [10] being in full compliance with the **EA** requirements for languages for writing mathematical texts was created and implemented in **RuSAD**. As for the **RuSAD** evidence maintaining engine, it was realized on the basis of both the resolution technique [11] and the sequent one, going back to the **AG** calculus, but using instead of the Kanger notion of admissibility, a new notion proposed by one of the authors of this paper in 1975 for improving quantifiers handling technique in the case, when the preliminary kolemization becomes a undesirable or impossible operation.

From 1983 until the collapse of the USSR and the decommissioning of the ES-lines computers in 1992, on which **RuSAD** was implemented, **RuSAD** was improved only in the direction of embedding human heuristic methods in its resolution part. The sequent approach to logical reasoning in the **EA** style did not develop in any way and was “preserved”, as it were, until 1998, when the paper's authors along with other researchers were joined to fulfilling the Intas project “Rewriting technique and efficient theorem proving”, in the framework of which there was decided to construct an English-language version of the **RuSAD** system, but already at a new level of the understanding of the **EA** programme and advances in information technologies.

This version, here called English **SAD** and denoted by **EnSAD**, was announced in 2002 at the IIS 2002 conference [12]. Now, the current **EnSAD** system is located at the site “http://nevidal.org/ sad.en.html” and there exists an online access to it. As its input language, **EnSAD** uses the **ForTheL** language [13] being an English-language modification of **TL** expanded later by a set of linguistic units for writing theorem proofs, which has turned **EnSAD** into a system capable of not only proving theorems, but also verifying mathematical texts. The own logical engine of **EnSAD** is built only on sequent formalism. But in order for the **EnSAD** system to use the already well-known (external) provers such as Vampire, SPASS, Otter, and so on, **EnSAD** contains a mechanism for connecting to these provers if a user so wishes (see, for example, [14]). In addition, a specific ontological method [15] for the processing of mathematical **ForTheL**-texts is implemented in **EnSAD**. For constructing the **EnSAD** logical engine, the sequential approach to conducting logical reasoning in the **EA**-style was revised and improved, which subsequently led to the emergence of a whole family of computer-oriented sequent calculi for both classical first-order logic and non-classical ones [16].

In 2008, work on the development of the **EnSAD** system was suspended due to lack of a financial support, but a theoretical studies on the construction of computer-oriented sequent calculi for a proof

[2] Here, we note that in what follows, we focus on the development and implementation of EA relating to the construction of a technique for a proof search in the EA-style. This is due to the fact that as a part of the research on EA, an original approach to the construction of machine-oriented calculi in the EA-style has been developed, to which the authors would like to draw attention of researchers on automated reasoning from the point of view of constructing efficient enough sequent calculi not only for classical first-order logic, but also for non-classical first-order ones, in particular, for r intuitionistic logic. All the others connected with EA found its description in [17].

search in classical and non-classical first-order logics have been continued. But in the course of such studies, an exact description of ideas that were used for constructing the evidential engines of the **RuSAD** and **EnSAD** systems are not fully reflected in publications on **EA**. This paper is intended to fill this gap and outline some possible ways of the further development of the **EnSAD** system.

2. EA and information processing in RuSAD and EnSAD

Both the systems, **RuSAD** and **EnSAD**, were constructed as applications suitable for the processing of mathematical texts written in a formal mathematical language. They are intended for solving the following task:

Let a (self-contained) text be written in a formal mathematical language and a certain proposition be pointed in it. It is necessary to find a proof of the proposition or to verify its proof given by a human.

The following scheme of a text transformation was proposed for the **RuSAD** and **EnSAD** systems (more details can be found in [17]):

Formalized text prepared by a user to prove/verify a theorem $\Rightarrow$
$\Rightarrow$ (using a parser) A self-contained first-order text $\Rightarrow$
$\Rightarrow$ (using a prover) A computer-made proof/verification $\Rightarrow$
$\Rightarrow$ (using an editor) Text in a form comprehensible for a human.

For preparing formalized (mathematical) texts, the Russian- and English-language versions of certain fragments of Russian and English natural languages were constructed, namely, **TL** (Theory Language) and **ForTheL** (Formal Theory Language), relatively. Their grammars and syntactical analyzers were designed and implemented in such a way that their outputs are computer internal presentations of formulas of the first-order language.

Note that **TL** and **ForTheL** reflect Glushkov's desire to have practical formal languages suitable for writing mathematical sentences and their proofs. (He wrote that they “should relate to the existing formal languages of mathematical logic as, for example, the ALGOL-60 language relates to the language of recursive functions or normal algorithms”[1].)

Here we complete the consideration of **TL** and **ForTheL** since there are many enough publications on these languages (the most part of such references can be found in [17] and, in particular, **ForTheL** has its complete description in the “ForTheL Reference” located on the site “nevidal.org”. At that, original “units” of the logical background of the “engines” of **RuSAD** and **EnSAD** do not have a detailed enough description in publications on the **EA** programme and **SAD** systems. Namely this will attract our attention in the rest of the paper.

Additionally, note that we restrict us by only the exact descriptions of sequent calculi providing this logical background, completely leaving aside relevant proofs and focusing attention only on their features.

3. Basic notions

A standard terminology of first-order logic without equality is used. The first order language is constructed over a signature, containing a countable set *Vr* of (object) variables, a finite (possibly, empty) set of functional symbols, and a finite (nonempty) set of predicate symbols.

Formulas are constructed with the help of the following logical connectives: the universal quantifier symbol $\forall$, the existential quantifier symbol $\exists$, and the propositional connectives for the implication ($\supset$), disjunction ($\vee$), conjunction ($\wedge$), and negation ($\neg$).

The notions of terms, atomic formulas, formulas, literals, complement literals, free and bound variables, and scopes of quantifiers are defined in the usual way [18] and assumed to be known to a reader.

We assume that no two quantifiers in any formula have, which can be achieved by renaming bound variables. Let a formula Φ be of the form $\neg\Psi$ or $\Psi \odot \mathrm{H}$ (where $\odot$ is one of $\wedge$,a common variable $\vee$, $\supset$), then $\neg$ and $\odot$ are called a principal propositional connective of Φ.

An equation is an unordered pair of terms s and t written as $s \approx t$. Assume L is a literal of the form $R(t_1, ..., t_n)$ (or $\neg R(t_1, ..., t_n)$) and M is a literal of the form $R(s_1, ..., s_n)$ (or $\neg R(s_1, ..., s_n)$), where R is a predicate symbol and $t_1, ..., t_n, s_1, ..., s_n$ are terms. Then $\Sigma(L, M)$ denotes the set of equations $\{t_1 \approx s_1, ..., t_n \approx s_n\}$. In this case, L and M are said to be equal modulo $\Sigma(L, M)$ ($L \approx M$ modulo $\Sigma(L, M)$).

A function σ from a set of variables to the set of terms is called a substitution if, and only if, it can be presented in the form $\sigma = \{x_1 \rightarrow t_1, ..., x_n \rightarrow t_n\}$, where $t_1, ..., t_n$ are terms, $x_1, ..., x_n$ are pairwise different variables, and $x_i \neq t_i$ for all $i = 1...n$.

Let Ex be an expression (i.e., a term or a formula) and σ a substitution. The result of the application of σ to Ex is denoted by $Ex \cdot \sigma$. For any set Ξ of expressions, $\Xi \cdot \sigma$ denotes the set obtained by the application of σ to every expression in Ξ. If Ξ is a set of (at least two) expressions and $\Xi \cdot \sigma$ is a singleton, then σ is called a unifier of Ξ.

The notions of a most general unifier and most general simultaneous unifier (mgsu) of sets of expressions are understood in the usual sense (see, for example, [11]).

If σ and λ are substitutions, then $\sigma \cdot \lambda$ denotes their composition, i.e. a substitution, the result of the application of which to an expression Ex is equal to $(Ex \cdot \sigma) \cdot \lambda$.

A sequent is an expression of the form $\Phi_1, ..., \Phi_p => \Psi_1, ..., \Psi_q$, where $\Phi_1, ..., \Phi_p, \Psi_1, ..., \Psi_q$ are formulas except that as in the case of [18], its succedent and antecedent are finite multisets. Note that without lost of generality, we can regard that $\Phi_1, ..., \Phi_p, \Psi_1, ..., \Psi_q$ are closed formulas and that all our sequents contain no more than one formula in their succedents (i. e. $q \leq 1$).

A tree is understood in the usual sense. A tree labeled by sequents is called a sequent tree.

In what follows, S^* denotes the sequent $\forall x_1 \exists y_1(R_1(x\ 1) \vee R_2(y_1)), \forall x_2 \exists y_2((R_1(y_2) \wedge \neg R_3(x_2)) \supset R3(x2)) => \forall y_3 \exists x_3(R_2(x_3) \vee R_3(y_3))$ (R_1, R_2, R_3 are predicate symbols) that is deducible in **G** [18] and will be used in several examples clarifying introduced notions and obtained results.

For formulas Φ and Ψ, we understand the notions of positive ($\Psi\lfloor\Phi^+\rfloor$) and negative ($\Psi\lfloor\Phi^-\rfloor$) occurrences of Φ in Ψ in the sense of the paper [19].

More precisely, let a formula Φ have one or more occurrences in a formula Ψ. Let us fix a certain occurrence of Φ in Ψ. This occurrence is called positive (negative) according to the following inductive definition:

– $\Psi\lfloor\Phi^+\rfloor$ holds, if Φ coincides with Ψ;
– $\Psi\lfloor\Phi^+\rfloor$ ($\Psi\lfloor\Phi^-\rfloor$) holds, if Ψ is of the form $\Psi_1 \wedge \Psi_2$, $\Psi_2 \wedge \Psi_1$, $\Psi_1 \vee \Psi_2$, $\Psi_2 \vee \Psi_1$, $\Psi_2 \supset \Psi_1$, $\forall x\Psi_1$, or $\exists x\Psi_1$ and $\Psi_1\lfloor\Phi^+\rfloor$ ($\Psi_1\lfloor\Phi^-\rfloor$) holds;
– $\Psi\lfloor\Phi^-\rfloor$ ($\Psi\lfloor\Phi^+\rfloor$) holds, if Ψ is of the form: $\Psi_1 \supset \Psi_2$ or $\neg\Psi_1$ and $\Psi_1\lfloor\Phi^-\rfloor$ ($\Psi_1\lfloor\Phi^+\rfloor$) holds.

Moreover (cf. [19]), a selected occurrence of a formula in a sequent $\Gamma => \Delta$ is called a positive (negative) one, if this occurrence is fixed as a positive in a formula from Δ (from Γ) or as a negative in a formula from Γ (from Δ).

If a formula Φ is of the form $\forall x\Phi'$ ($\exists x\Phi'$) and Φ has a positive (negative) occurrence in a formula Ψ or a sequent S, then $\forall x$ ($\exists x$) is called a positive quantifier in Ψ or S, respectively; $\exists x$ ($\forall x$) is called a negative quantifier in Ψ or S, if $\exists x\Phi'$ ($\forall x\Phi'$) has a positive (negative) occurrence in Ψ or S, respectively.

By the eigenvariable convention, any quantifier cannot have more than one (positive or negative) occurrence in a formula or sequent. Hence, the following definitions do not lead to misunderstanding.

The variable of a positive quantifier occurring in a formula Ψ (a sequent S) is called a parameter in Ψ (in S); the variable of a negative quantifier occurring in Ψ (in S) is called a dummy in Ψ (in S).

Note that we preserve the name "variable" for denoting both dummies and parameters in the cases, when it is only important that they occur in a formula or sequent under consideration.

For S^*, we have: x_1, x_2, and x_3 are dummies and y_1, y_2, and y_3 parameters.

Because the properties "to be a dummy" and "to be a parameter" are invariant for any variable x w.r.t. any rule application in our sequent calculi, the satisfaction of the following convention will not lead to an ambiguity: if the application of a quantifier rule eliminates a positive (negative) quantifier Qx (Q is $\forall$ or $\exists$) and substitutes a variable y for x, then y is considered as a parameter (dummy).

For technical reason only, we write x in order to indicate that when a quantifier rule is applied, x is replaced by a new parameter (denoted by x) and, analogously, write x in order to indicate that when a quantifier rule is applied, x is replaced by a new dummy (denoted by x).

We use the usual definition of a sequent calculus; at that, the deduction of a sequent in it has the form of an inference search tree growing "from top to bottom" according to the applications of inference rules "from top to bottom".

An inference search tree is called a proof tree, if all its leaves are labeled by axioms. A sequent *S* is deducible in a sequent calculus, if there exists a proof tree for S in the calculus and, maybe, some additional conditions are satisfied. Note that any sequent calculus under consideration does not contain the cut rule and that all its rules satisfy the subformula property (see, for example, the paper [9] or [18]).

4. Admissible substations

When quantifier rules are applied, some substitution of selected terms for variables is made. To do this step of deduction sound, certain restrictions are put on the substitution. The substitution, satisfying these restrictions, is said to be admissible. Below we investigate the classical notion of admissible substitution and show how it can be modified in such a way that efficient enough sound and complete sequent calculi can be finally obtained. We follow the paper [20], in which the sound and complete calculus **G** [14] is taken for the demonstration of peculiarities of quantifier handling in sequent calculi reflecting in the logical "engines" of **RuSAD** and **EnSAD**.

4.1. Gentzen's admissible substitutions

Gentzen quantifier rules are usually of the following form [14]:

$$(\exists: \textit{left})\text{-rule:}\quad \frac{\Gamma,\ \exists x\Phi \Rightarrow \Delta}{\Gamma,\ \Phi|^x_y \Rightarrow \Delta}, \qquad (\forall: \textit{right})\text{-rule:}\quad \frac{\Gamma \Rightarrow \forall x\Phi,\ \Delta}{\Gamma \Rightarrow \Phi|^x_y,\ \Delta},$$

$$(\forall: \textit{left})\text{-rule:}\quad \frac{\Gamma,\ \forall x\Phi \Rightarrow \Delta}{\Gamma,\ \Phi|^x_t \Rightarrow \Delta}, \qquad (\exists: \textit{right})\text{-rule:}\quad \frac{\Gamma \Rightarrow \exists x\Phi,\ \Delta}{\Gamma \Rightarrow \Phi|^x_t,\ \Delta}.$$

where *y* is a new parameter, the term *t* is required to satisfy the eigenvariable condition, that is, *t* should be free for the variable *x* in the formula Φ; at that, $\Phi|^x_y$ and $\Phi|^x_t$ denote the results of the replacement of *x* by y and *t*, respectively. The restriction of the substitution of *t* for *x* leads to the Gentzen (classical) notion of admissible substitutions, which proves to be sufficient for the needs of the proof theory. But it becomes inconvenient from the point of view of efficiency of computer-oriented sequent calculi. It is clear from the following example.

Rewrite the sequent S^* in the form $\Phi^*_1, \Phi^*_2 \Rightarrow \Psi^*$, where Φ^*_1 is $\forall x_1 \exists y_1(R_1(x\ 1) \vee R_2(y_1))$, Φ^*_2 is $\forall x_2 \exists y_2((R_1(y_2) \wedge \neg R_3(x_2)) \supset R_3(x_2))$, and Ψ^* is $\forall y_3 \exists x_3(R_2(x_3) \vee R_3(y_3))$.

Later, it will be shown that S^* is deducible in **G**, while here we note that for establishing this, quantifier rules should be applied to all the quantifiers occurring in Φ^*_1, Φ^*_2, and Ψ^*.

Therefore, the Gentzen classical notion of admissible substitution yields 90 (= 6!/(2!*2!*2!)) different orders of the quantifier rule applications to the sequent $\Phi^*_1, \Phi^*_2 \Rightarrow \Psi^*$. And only one of them (subsequently eliminating the quantifiers in the order: $\forall y_3$, $\forall x_2$, $\exists y_2$, $\forall x_1$, $\exists y_1$, $\exists x_3$) leads to establishing the provability of S^*. Thus, in the worse case, we should check all 90 possible quantifier rules applications. It is clear that due to skolemazation, resolution type methods permit to avoid this redundant work.

4.2. Kanger's admissible substitutions

To optimize the procedure of quantifier rules applications, S. Kanger suggested in [8] his sequent calculus, denoted here by **K**. In the *K* calculus, a proof search is being split into stages, leading to the construction of a ``pattern" of a deduction tree, maybe, containing parameters and dummies. To complete the deduction process, once in a while an attempt is made to convert a "pattern" into a proof tree by substituting certain terms for dummies. In the case of failure, the process is continued.

The main difference between **K** and **G** consists in splitting the inference search process into stages and using special modifications of the above-given quantifier rules. Thus, the calculus **K** also contains

($\exists$: *left*) and ($\forall$: *right*), while the rules ($\forall$: *left*) and ($\exists$: *right*) are replaced by the rules (cf. [8]):

($\forall$': *left*)-rule: $\dfrac{\Gamma,\ \forall x\Phi \Rightarrow \Delta}{\Gamma,\ \Phi|^{x}_{z} \Rightarrow \Delta}\quad [z/t_1, ..., t_n]$ ($\exists$': *right*): $\dfrac{\Gamma \Rightarrow\ \exists x\Phi,\ \Delta}{\Gamma \Rightarrow \Phi|^{x}_{z},\ \Delta}\quad [z/t_1, ..., t_n]$

where $t_1, ..., t_n$ are some of the terms that can be constructed from constant, parameters, and functional symbols occurring in the conclusion of the rules, z is a dummy, and $[z/t_1, ..., t_n]$ denotes that when an attempt is made to convert a "pattern" into a proof tree, a dummy z can be replaced by one of the terms $t_1, ..., t_n$ only. This replacement of dummies by terms is made in the end of every stage, and at every stage, inference rules are applied in a certain order.

This scheme of the deduction construction in the **K** calculus leads to the notion of a Kanger-admissible substitution, which is more efficient than the classical one. Thus, for the above-given example it yields only 6 (=3!) variants of different possible orders of the quantifier rule applications (and none of these variants is preferable). Despite this, the Kanger-admissible substitutions still do not allow to attain efficiency comparable with that when skolemization is made. The reason for this is that, as in case of the classical admissible substitution, it is required to select a certain order of the quantifier rule applications when an input sequent is deduced, and, if it proves to be unsuccessful, another order of applications should be tried, and so on.

4.3. Word admissible substitutions

On the example of the **mG** calculus from [20], being a modification of the **G** calculus, let us demonstrate how the below-given new notion of an admissible substitution permits to get rid of the dependence of the deduction efficiency in sequent calculi on different possible orders of quantifier rule applications.

The main idea of this notion is to determine by the quantifier structures of formulas of an input sequent and a substitution under consideration, whether there exists a sequence of desired quantifier rules applications or not.

Let W be a set of sequences (words) of parameters and dummies and σ a substitution. Put $\Theta(W, \sigma) = \{\langle z, t, w\rangle : z$ is a variable of σ, t is a term of σ, $w \in W$, and z lies in w to the left of some parameter from $t\}$. The substitution σ is said to be word admissible (w-admissible) for W if, and only if, (1) all the variables of σ are dummies and (2) in $\Theta(W, \sigma)$, there are no elements $\langle z_1, t_1, w_1\rangle, ..., \langle z_n, t_n, w_n\rangle$ such that $z_1 \rightarrow t_2 \in \sigma, ..., z_{(n-1)} \rightarrow t_n \in \sigma, z_n \rightarrow t_1 \in \sigma$ $(n > 0)$.

Let us make some remarks about the **mG** calculus. It deals with formulas, except that a certain sequence of parameters and dummies is attached to each formula from sequents. That is why the **mG** calculus is defined on pairs of the form $\langle w, \Phi\rangle$, where Φ is a formula and w a word (sequence) of parameters and dummies.

An expression of the form $\langle w_1, \Phi_1\rangle, ..., \langle w_m, \Phi_m\rangle \Rightarrow \langle v_1, \Psi_1\rangle, ..., \langle v_n, \Psi_n\rangle$, where $w_1, ..., w_m, v_1, ..., v_n$ are sequence of parameters and dummies and $\Phi_1, ..., \Phi_m, \Psi_1, ..., \Psi_n$ are formulas, is called an a-sequent. At that, the empty sequence is always added to all the formulas of an initial sequent producing a so-called input a-sequent for **mG**.

Consider the following quantifier rules.

$\dfrac{\Gamma,\ \langle w, \exists x\Phi\rangle \Rightarrow \Delta}{\Gamma,\ \langle wy, \Phi|^{x}_{y}\rangle \Rightarrow \Delta}$ ($\exists^*$: *left*), $\dfrac{\Gamma,\ \langle w, \forall x\Phi\rangle \Rightarrow \Delta}{\Gamma,\ \langle wy, \Phi|^{x}_{y}\rangle \Rightarrow \Delta}$ ($\forall^*$: *right*),

$\dfrac{\Gamma,\ \langle w, \forall x\Phi\rangle \Rightarrow \Delta}{\Gamma,\ \langle wy, \Phi|^{x}_{y}\rangle \Rightarrow \Delta}$ ($\forall^*$: *left*), $\dfrac{\Gamma,\ \langle w, \exists x\Phi\rangle \Rightarrow \Delta}{\Gamma,\ \langle wy, \Phi|^{x}_{y}\rangle \Rightarrow \Delta}$ ($\exists^*$: *right*),

where w is a word, Φ a formula, y a parameter, and z a dummy.

The calculus **mG** can be defined as the **G** calculus expanded on the case of w-sequents and containing the just-given quantifier rules instead on its own quantifier rules.

Applying first rules "from top to bottom" to an input a-sequent for **mG** and afterwards to its "heirs" "from left to right", and so on, we finally obtain a so-called inference search tree for the input a-sequent.

An inference tree Tr for an input a-sequent is called a proof tree in **mG** if, and only if, there exists such a substitution of terms for variables, say, σ, that (1) after application of σ to all the formulas from the a-sequents of all the leaves of Tr, these a-sequents become axioms, that is, each of them is of the form $\Gamma => \Delta$ with such Γ and Δ that Γ and Δ contain the same formula, and (2) the substitution σ is w-admissible for the set of all the words of parameters and dummies from all the leaves of *Tr*.

As it was shown in [20], a sequent *S* is deducible in **G** if, and only if, a proof tree for the input a-sequent corresponding to *S* can be constructed in **mG**. That is **mG** is a sound and complete calculus for classical first-order logic in the sense of this "coextensivity" of **G** and **mG.**

If we consider the above-given sequent S^* and the corresponding input a-sequent $\langle , \Phi^*_1\rangle, \langle , \Phi^*_2\rangle => \langle , \Psi^*\rangle$ for **mG**, then applying only quantifier rules eliminating all quantifiers in Φ^*_1, Φ^*_2, and Ψ^* in any order and after this, applying only propositional rules to the result, we can construct an inference tree, say, Tr, all leaves of which are the following a-sequents:

$\langle x_1y_1, R_1(x_1)\rangle, \langle x_2y_2, R_3(x_2)\rangle => \langle y_3x_3, R_2(x_3)\rangle, \langle y_3x_3, R_3(y_3)\rangle,$

$\langle x_1y_1, R_1(x_1)\rangle => \langle y_3x_3, R_2(x_3)\rangle, \langle y_3x_3, R_3(y_3)\rangle, \langle x_2y_2, R_1(y_2)\rangle,$

$\langle x_1y_1, R_2(y_1)\rangle, \langle x_2y_2, R_3(x_2)\rangle => \langle y_3x_3, R_2(x_3)\rangle, \langle y_3x_3, R_3(y_3)\rangle,$

$\langle x_1y_1, R_2(y_1)\rangle => \langle y_3x_3, R_2(x_3)\rangle, \langle y_3x_3, R_3(y_3)\rangle, \langle x_2y_2, R_1(y_2)\rangle,$

where x_1, x_2, x_2 are dummies and y_1, y_2, y_3 parameters.

For a substitution $\sigma^* = \{x_1 \rightarrow y_2, x_2 \rightarrow y_3, x_3 \rightarrow y_1\}$, we see that the application of σ^* to all the just-given leafs converts them into axioms of **mG**. Moreover, σ^* is w-admissible for the set $\{x_1y_1, x_2y_2, x_3y_3\}$. Thus, *Tr* is a proof tree in **mG**, which implies the deducibility of S^* in the **G** calculus.

Note that the proof of the proposition about the soundness and completeness of **mG** is carried out in [16] in such a way that if in it we replace the **G** calculus by another (sound and complete) sequent calculus, say, **G**' , with the usual quantifier rules, then the replacement of them by ($\exists^*$: *left*)-, ($\forall^*$: *right*)-, ($\forall^*$: *left*)-, and ($\exists^*$: *right*)-rules leads to a calculus, say, **mG**' defined on w-sequents, such that **G**' and **mG**' possess the same properties w.r.t. deducibility that **G** and **mG** have.

5. RuSAD and automated theorem proving

The above-given example demonstrates that inference search in **mG** has the following properties provided by using the w-admissibility instead of the Gentzen or Kanger one:

– the selection of an order of quantifier rules applications does not affect the final result, which means the following: (1) if a selected order of the applications of quantifier rules in **mG** leads to the construction of a proof tree, then the same proof tree can be constructed in the case of the selection of any another order of the quantifier rule applications eliminating the same quantifiers that were eliminated in using the selected order; (2) if a selected order of applications of quantifier rules in mG cannot lead to the construction of a proof tree, then any another order of the quantifier rule applications eliminating the same quantifiers that were eliminated in the selected order cannot lead to the construction of a proof tree.

– the selection of a substitution can be made at any suitable moment; as a result, equality handling can be separated from deduction.

These positive features of **mG** demonstrate the usefulness of incorporating the notion of a w-admissible substitution into computer-oriented inference search. Below, we make the reconstruction of the **AG** calculus from [7] denoted by **mAG** and take it as a basic calculus when constructing the sequent logical engine of **RuSAD**.

5.1. Calculus of w-sequents

A basic object of the **mAG** calculus is a w-sequent. It may be considered as a special generalization of the standard notion of a sequent. We will deal with w-sequents having only one object (goal) in its succedent, which allows making inference search in **mAG** goal-driven.

An ordered triple $\langle w, \Phi, E\rangle$ is called an ensemble if, and only if, w is a sequence (a word) of dummies and parameters, Φ is a first-order formula, and *E* is a set of pairs of terms t_1, t_2 (equations of the form $t_1 \approx t_2$).

A w-sequent is an expression of the form $\langle w_1, \Phi_1, E_1\rangle,..., \langle w_n, \Phi_n, E_n\rangle => \langle w, \Psi, E\rangle$, where $\langle w_1, \Phi_1, E_1\rangle$, ..., $\langle w_n, \Phi_n, E_n\rangle$, and $\langle w, \Psi, E\rangle$ are ensembles.

Ensembles in the antecedent of a w-sequent are called premises, and an ensemble in the succedent of a w-sequent is called a goal of this w-sequent.

The **mAG** calculus contains goal-splitting rules and premise-splitting rules transforming a sequent under consideration into w-sequents obligatorily with new goals and, maybe, new premises.

The goal-splitting rules make decomposition of Ψ in $\langle w_1, \Phi_1, E_1\rangle, \ldots, \langle w_n, \Phi_n, E_n\rangle \Rightarrow \langle w, \Psi, E\rangle$ by its principal logical connective, while premise-splitting rules realizes a possible interaction of Ψ with Φ_i, which leads to generating a new w-sequent (new w-sequents). The sets $E_1, \ldots, E_n$, and E define the terms to be substituted for dummies in order to transform each equation of the form $t_1 \approx t_2$ from $E_1, \ldots, E_n$, and E to an identity of the form $t \approx t$ by applying to $E_1, \ldots, E_n$, and E a substitution chosen in a certain way. The words (sequences) $w_1, \ldots, w_n$, and w participate in checking whether a substitution generated during a proof search is w-admissible.

An initial w-sequent is constructed as follows. Suppose we want to establish the deducibility of a usual (original) sequent S of the form $\Phi_1, \ldots, \Phi_n \Rightarrow \Psi$ (for example, in the calculus **G**), where $\Phi_1, \ldots, \Phi_n$ and Ψ are closed formulas. Then w-sequent $\langle , \Phi_1, \emptyset\rangle, \ldots, \langle , \Phi_n, \emptyset\rangle \Rightarrow \langle , \Psi, \emptyset\rangle$ is declared as an initial one for $\Phi_1, \ldots, \Phi_n \Rightarrow \Psi$.

During a proof search in **mAG** an inference tree is constructed. At the beginning of a search process it consists of an initial w-sequent. The subsequent nodes of the inference tree are generated accordingly to the rules described below. Note that inference trees grow "from top to bottom".

Goal Splitting Rules

These rules are used for the elimination of the principal logical connective from the formula-goal of a w-sequent under consideration. The application of any such a rule results in the generation of a new w-sequent (new w-sequents) with only one goal and, possibly, with new premises. The elimination of propositional connectives is done on the basis of the well-known transformations of formulas in classical first-order logic and can be easily expressed in the form of derivative rules of a standard Gentzen-type calculus.

In the below-given rules, Γ is a multiset of premises, w a word consisting of dummies and parameters, E a set of equations, and Φ_1, Φ_2, and Φ formulas.

$(\Rightarrow\supset_1)$-rule: $$\frac{\Gamma \Rightarrow \langle w, \Phi_1 \supset \Phi_2, E\rangle}{\Gamma, \langle w, \Phi_1, E\rangle \Rightarrow \langle w, \Phi_2, E\rangle}$$

$(\Rightarrow\supset_2)$-rule: $$\frac{\Gamma \Rightarrow \langle \Phi_1 \supset \Phi_2, E\rangle}{\Gamma, \langle w, \neg\Phi_2, E\rangle \Rightarrow \langle w, \neg\Phi_1, E\rangle}$$

$(\Rightarrow\vee_1)$-rule: $$\frac{\Gamma \Rightarrow \langle w, \Phi_1 \vee \Phi_2, E\rangle}{\Gamma, \langle w, \neg\Phi_1, E\rangle \Rightarrow \langle \Phi_2, E\rangle}$$

$(\Rightarrow\vee_2)$-rule: $$\frac{\Gamma \Rightarrow \langle \Phi_1 \vee \Phi_2, E\rangle}{\Gamma, \langle w, \neg\Phi_2, E\rangle \Rightarrow \langle w, \Phi_1, E\rangle}$$

$(\Rightarrow\wedge)$-rule: $$\frac{\Gamma \Rightarrow \langle w, \Phi_1 \wedge \Phi_2, E\rangle}{\Gamma \Rightarrow \langle w, \neg\Phi_1, E\rangle \quad \Gamma \Rightarrow \langle w, \Phi_2, E\rangle}$$

$(\Rightarrow\neg)$-rule: $$\frac{\Gamma \Rightarrow \langle w, \neg\Phi, E\rangle}{\Gamma \Rightarrow \langle w, \Phi', E\rangle}$$

where Φ' is the result of one-step moving the sign $\neg$ into Φ, if Φ is not an atomic formula; otherwise Φ' coincides with Φ.

Quantifier Rules

$(\Rightarrow\forall)$-rule: $$\frac{\Gamma \Rightarrow \langle w, \forall x\Phi, E\rangle}{\Gamma \Rightarrow \langle wy, \Phi|^x_y\, E\rangle}$$

$(\Rightarrow\exists)$-rule: $$\frac{\Gamma \Rightarrow \langle w, \exists x\Phi, E\rangle}{\langle w, \forall x'(\neg\Phi|^x_{x'}), E\rangle \Rightarrow \langle wz, \Phi|^x_z, E|^x_z\rangle}$$

where x' and z are new dummies and y is a new parameter.

Auxiliary Goal Rules

The Auxiliary Goal rules (**AG**-rules) are applied, when a formula, say, L, in the goal of a w-sequent under consideration is a literal and one of the premises of the sequent contains a negative (w.r.t. the sequent) occurrence of a literal, say, L', such that $L \approx L'$ modulo $\Sigma(L, L')$. (We simply write $L \approx L'$, when $\Sigma(L, L')$ is immaterial.)

The fixing of a selected occurrence of L' produces such a sequence of rules eliminating logical connectives in premises that leads to the "goal-driven" generation of a w-sequent, containing a premise with L' as its formula.

In all the below-given rules, Φ_1 and Φ_2 are formulas, L and L' are literals and $L \approx L'$ modulo $\Sigma(L, L')$, w and w' are words consisting of dummies and parameters, E, E', and E'' are sets of equations such that $E'' = E \cup E' \cup \Sigma(L, L')$. For a formula Φ, the expression $\Phi\lfloor L^-\rfloor$ ($\Phi\lfloor L^+\rfloor$) denotes that a selected occurrence of L in Φ is negative (positive) in Φ.

$(\supset_1 \Rightarrow)$-rule: $$\frac{\Gamma, \langle w, \Phi_1\lfloor L^-\rfloor \supset \Phi_2, E\rangle \Rightarrow \langle w', L', E'\rangle}{\Gamma, \langle w, (\neg\Phi_1)\lfloor L^+\rfloor, E\rangle \Rightarrow \langle w', L', E''\rangle \quad \Gamma \Rightarrow \langle w, \neg\Phi_2, E\rangle}$$

($\supset_2$ =>)-rule: $$\frac{\Gamma, \langle w, \Phi_1 \supset \Phi_2\lfloor L^+\rfloor, E\rangle => \langle w', L', E'\rangle}{\Gamma, \langle w, \Phi_2\lfloor L^+\rfloor, E\rangle => \langle w', L', E''\rangle \quad \Gamma => \langle w, \Phi_1, E\rangle}$$

($\vee_1$ =>)-rule: $$\frac{\Gamma, \langle w, \Phi_1\lfloor L^+\rfloor \vee \Phi_2, E\rangle => \langle w', L', E'\rangle}{\Gamma, \langle w, \Phi_1\lfloor L^+\rfloor, E\rangle => \langle w', L', E''\rangle \quad \Gamma => \langle w, \neg\Phi_2, E\rangle}$$

($\vee_2$ =>)-rule: $$\frac{\Gamma, \langle w, \Phi_1 \vee \Phi_2\lfloor L^+\rfloor, E\rangle => \langle w', L', E'\rangle}{\Gamma, \langle w, \Phi_2\lfloor L^+\rfloor, E\rangle => \langle w', L', E''\rangle \quad \Gamma => \langle w, \neg\Phi_1, E\rangle}$$

($\wedge_1$ =>)-rule: $$\frac{\Gamma, \langle w, \Phi_1\lfloor L^+\rfloor \wedge \Phi_2, E\rangle => \langle w', L', E'\rangle}{\Gamma, \langle w, \Phi_1\lfloor L^+\rfloor, E\rangle, \langle \Phi_2, E\rangle => \langle w', L', E'\rangle}$$

($\wedge_2$ =>)-rule: $$\frac{\Gamma, \langle w, \Phi_1 \wedge \Phi_2\lfloor L^+\rfloor, E\rangle => \langle w', L', E'\rangle}{\Gamma, \langle w, \Phi_1, E\rangle, \langle \Phi_2\lfloor L^+\rfloor, E\rangle => \langle w', L', E'\rangle}$$

($\neg$ =>)-rule: $$\frac{\Gamma, \langle w, \neg(\Phi\lfloor L^-\rfloor), E\rangle => \langle w', L', E'\rangle}{\Gamma, \langle w, \Phi'\lfloor L^+\rfloor, E\rangle => \langle w', L', E'\rangle},$$

where Φ' is the result of one-step moving the sign $\neg$ into Φ, if Φ is not an atomic formula; otherwise Φ' coincides with Φ.

Quantifier Rules

($\forall$ =>)-rule: $\dfrac{\Gamma, \langle w, \forall x\Phi\lfloor L^+\rfloor, E\rangle => \langle w', L', E'\rangle}{\Gamma, \langle wy, (\Phi\lfloor L^+\rfloor)|^x_y, E\rangle => \langle w', L', E'\rangle}$, ($\exists$ =>)-rule: $\dfrac{\Gamma, \langle w, x\Phi\lfloor L^+\rfloor, E\rangle => \langle w', L', E'\rangle}{\Gamma, \langle wz, (\Phi\lfloor L^+\rfloor)|^x_z, E\rangle => \langle w', L', E'\rangle}$,

where *y* is a new parameter and *z* a new dummy.

Termination Rules

(=> $\#_1$)-rule: $$\frac{\Gamma, \langle w, L, E\rangle => \langle w', L', E'\rangle}{\Gamma => \langle w, \#, E''\rangle}$$

where *L* and *L'* are literals such that $L \approx L'$ modulo $\Sigma(L, L')$ and $E'' = E \cup E' \cup \Sigma(L, L')$.

(=> $\#_2$)-rule: Let *Tr* be an inference search tree and *Br* some its branch ending by a tree leaf $\Gamma => \langle w, L, E\rangle$, where *L* is a literal. Let *Br* contain a w-sequent $\Gamma' => \langle w', L', E'\rangle$, such that *L' is a literal and* $L \approx L''$ *modulo* $\Sigma(L, L'')$ *holds for the complement literal* *L''* of *L'* and $E'' = E \cup E' \cup \Sigma(L, L'')$. Then the sequent $\Gamma => \langle w, \#, E''\rangle$ is said to be deducible from $\Gamma => \langle w, L, E\rangle$ by the (=> $\#_2$)-rule.

Premise-Adding Rule

(ρ =>)-rule: After the ($\forall$ =>)-rule application, the triple $\langle w, \forall x'(\Phi|^x_{x'}), E\rangle$ can be added to the antecedent of any w-sequent containing $\langle wx, (\Phi\lfloor L^+\rfloor)|^x_{x'}, E\rangle$ as its premise in a current sequent tree, where x' is a new variable.

Axioms. Axioms are w-sequents of the form $\Gamma => \langle w, \#, E\rangle$, where # denotes the empty formula.

Proof Tree. An inference tree *Tr* in **mAG** is called a proof tree in **mAG** if, and only if, all the leaves of *Tr* are axioms, there exists the most general simultaneous unifier σ of all the equations from the axioms of *Tr*, and σ is a w-admissible substitution for the set of all the words from *Tr*.

5.3. Main results for mAG calculus

The soundness and completeness of the **mAG** calculus have the following forms, one of which is of the form of coextensetivity.

Proposition 1. For closed formulas $\Phi_1, \ldots, \Phi_n$, and Ψ, the sequent $\Phi_1, \ldots, \Phi_n => \Psi$ is deducible in a Gentzen-type (sound and complete) calculus with standard quantifier rules (for example, in G) if, and only if, a proof tree can be constructed in the **mAG** calculus for the initial w-sequent $\langle, \Phi_1, \varnothing\rangle, \ldots, \langle, \Phi_n, \varnothing\rangle => \langle, \Psi, \varnothing\rangle$.

Corollary 1. A formula Ψ is valid in classical first-order logic without equality if, and only if, a proof tree can be constructed in the **mAG** calculus for the initial w-sequent $=> \langle, \Psi, \varnothing\rangle$.

5.4. Peculiarities of proof search in RuSAD

Logical engine of **RuSAD** was constructed as containing of two independent parts. The first, resolution part was based on the well-known deductive system “negative resolution+paramodulation”, while the second one practically presents a computer realization of the **mAG** calculus constructed in such a way that it satisfies at least the **EA**-requirements (ii) - (iv), which did not get representation in publications on **EA**. The subsequent content is intended to eliminate this disadvantage.

The satisfying of **mAG** to the **EA**-requirements (ii) - (iv) was the main reason for taking **mAG** as a basis for constructing the logical engine of **RuSAD**. An additional reason was that **mAG** also partially satisfies (i) in the case of "coloring" premises of an input w-sequent into definitions, auxiliary propositions, and theorem preliminaries (theorem conditions) depending on in which premise of the sequent, a considered linguistic unit of an original **TL**-text (definition, auxiliary proposition, or theorem condition) is translated. This means that after the translation of an original **TL**-text into an input w-sequent, any its premise has a label "coloring" it into a definition, auxiliary proposition, or theorem condition.

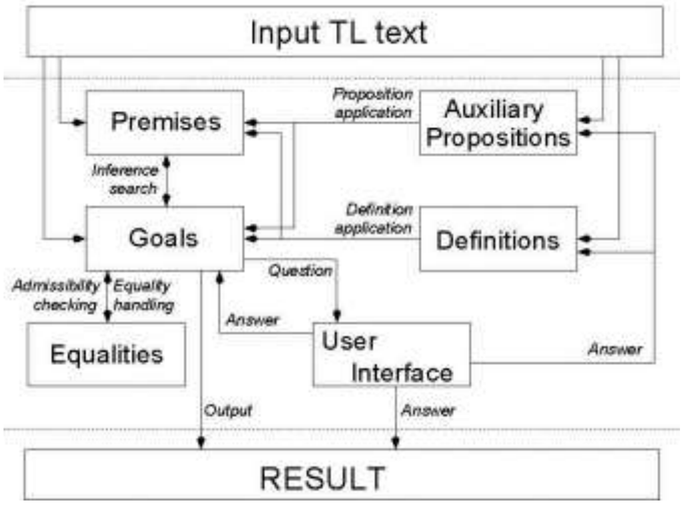

Figure 1: **Scheme of proof search in RuSAD's logical engine**

An input w-sequent "colored" by such a way imposes the following order of involving its premises in the proof search process: First of all, an attempt to prove a theorem under consideration is made by using only the theorem preliminaries. In the case of failure, one or another definition is involved in the proof search process, and in the case if the use of preliminaries and definitions do not lead to success, auxiliary propositions are allowed to use.

More precisely. Let T denote a theorem that should be proved in the assumption that there are theorem preliminaries $C_1, ..., C_m$, definitions $D_1, ..., D_n$, and auxiliary propositions $A_1, ..., A_r$ written in **TL**. The **TL**-parser translates them into first-order formulas and, as a result, an initial w-sequent $\langle, \Phi(C_1), \varnothing\rangle, ..., \langle, \Phi(C_m), \varnothing\rangle, \langle, \Phi(D_1), \varnothing\rangle, ..., \langle, \Phi(D_m), \varnothing\rangle, \Phi(A_1), \varnothing\rangle, ..., \langle, \Phi(A_r), \varnothing\rangle => \langle, \Phi(T), \varnothing\rangle$ is constructed, where $\Phi(C_1), …, \Phi(C_n)$ are first-order formulas presenting $C_1, ..., C_m$, $\Phi(D_1), …, \Phi(D_n)$ are first-order formulas presenting $D_1, ..., D_n$; $\Phi(A_1), ..., \Phi(A_r)$ are first-order formulas presenting $A_1, ..., A_r$; $\Phi(T)$ is a first-order formula presenting T. Then the **RuSAD** sequent engine makes an attempt to reach the goal $\langle, \Phi(T), \varnothing\rangle$ using firstly $\langle, \Phi(C_1), \varnothing\rangle, ..., \langle, \Phi(C_m), \varnothing\rangle$. In the case of failure, it makes an attempt to use additionally some or all $\langle, \Phi(D_1), \varnothing\rangle, ..., \langle, \Phi(D_m), \varnothing\rangle$. And only after this, if there is no success, the sequent engine involves some or all $\langle, \Phi(A_1), \varnothing\rangle, ..., \langle, \Phi(A_r), \varnothing\rangle$ into the proof search process. We refer to Fig. 1 for a relatively full description of information processing in the **RuSAD** sequent engine, in which the unification algorithm was incorporated for equation ("equality") handling and a w-admissibility technique was built-in for the admissibility checking.

Note that it is presupposed for a human to make intervention in the proof search process with the aim to change the direction of proof search by, for example, setting his own order of the use of premises.

A theorem T is **RuSAD**-valid in a context of a **TL**-text *Txt* if, and only if, the **RuSAD** system can prove T using only facts (notions, definitions, lemmas, etc.) from the text *Txt* in the assumption that it has an infinite memory to store data and that the system has an infinite time to operate.

The **RuSAD** system is sound and correct in the following sense.

Theorem 1. Let *Txt* be a self-contained noncontradictory **TL**-text for a theorem *T* and $Txt^* => T^*$ be a sequent, in which Txt^* and T^* are the results of the translation of respectively *Txt* and *T* into first-order formulas. Then *T* is a **RuSAD**-valid theorem in the context of the **TL**-text *Txt* if, and only if, the sequent $Txt^* => T^*$ is deducible in a Gentzen-type sound and complete calculus with standard quantifier rules.

Finally note that the trial exploitation of the **RuSAD**'s sequent engine shows good enough results. For example, when solving some tasks, this engine produced only several dozen new w-sequents, while the **RuSAD**'s resolution engine generates several hundred new clauses, when solving the same tasks.

6. EnSAD, automated theorem proving, and proof verification

The **mAG** calculus showes that words in its quantifier rules are used only for fixing the quantifier structures of formulas in an input sequent; at that, both these structures and words remain unchanged during proof search since they are invariant w.r.t. the application of any **mAG** inference rule.

This leads to the idea that if we preliminary remember the quantifier structures of formulas from an input sequent and develop a special technique for handling them, we will be able to completely refuse from quantifier rules and get a quantifier-rules-free calculus. This idea was traced in a number of works of the first author and found its most complete reflection in [11]. But in all these papers, an attention is mainly concentrated only on a quantifier-handling technique, while below we show how it is possible to construct certain modifications of the **mAG** calculus combining this quantifier-handling technique with the goal-driven applying of propositional (quantifier-free) rules. At that, the results from [16] lead to satisfying the above-formulated strong coextensivity property providing a simple enough way for obtaining results similar to given in Proposition 1 for other sequent calculi.

6.1. Strong Admissible Substitutions

A formula Φ (sequent *S*) containing only variables from *Vr* is called an original one.

Every dummy or parameter v of an original formula Φ (an original sequent *S*) generates a countable set of new variables of the form ${}^kv$ ($k = 1,2,...$) called indexed variables and this set is denoted by $Vr^+(v)$; at that, k is called an index. The union of all the sets $Vr^+(v)$ taken on all the dummies or parameters v occurred in a formula Φ (sequent *S*) is denoted by $Vr^+(\Phi)$ ($Vr^+(S)$). We will simply write Vr^+ in case, if Φ or *S* is immaterial.

If $v_1, ..., v_r$ is the list of all the dummies and parameters of an initial formula Φ (an original sequent *S*) and *k* is a natural number, then ${}^k\Phi$ (${}^kS$) is the result of the simultaneous replacement in Φ (*S*) of each occurrence of v_j by ${}^kv_j$ (j = 1, ..., r).

For a formula Φ (a sequent *S*), $\mu(\Phi)$ ($\mu(S)$) denotes the result of the removing of all the quantifiers from Φ (from *S*).

If Φ (*S*) is an initial formula (an initial sequent) and *v* is its parameter or dummy, then *v* is defined as a parameter or dummy in $\mu(\Phi)$ (in $\mu(S)$)respectively. Moreover, if *v* is a parameter (dummy) in Φ or *S*, then, by definition, ${}^kv$ is a parameter (dummy) in $\mu({}^k\Phi)$ or $\mu({}^kS)$. Thus, *v* is a parameter or dummy in $\mu(\Phi)$ ($\mu(S)$) if, and only if, ${}^kv$ is a parameter or dummy in $\mu({}^k\Phi)$ ($\mu({}^kS)$).

The result of adding upper-left indexes to all variables from an original formula or sequent is called its copy. Additionally, we require that two copies of the same original formula or sequent are copies each other, that is the relation “to be a copy” is transitive.

Let Φ be an original formula and different quantifiers *Qx* and *Q'y* occur in Φ, where *x* and *y* are variables and *Q* and *Q'* are $\forall$ or $\exists$. We write $x <_\Phi y$ if, and only if, in Φ, the selected occurrence of *Q'y* is in the scope of the selected occurrence of *Qx*. For example, if Φ is $\forall x \neg \exists y P(x,y)$, then $x <_\Phi y$.

For an original sequent *S* of the form $\Phi_1, ..., \Phi_p => \Psi_1, ..., \Psi_q$, we have $x <_S y$ if, and only if, $x <_{\Phi i} y$ or $x <_{\Psi j} y$ ($1 \leq i \leq p, 1 \leq j \leq q$).

Because of the convention about bound variables occurred in sequents, this definition of $<_S$ is correct and does not lead to ambiguity.

Moreover, $<_S$ is an irreflexive and transitive relation.

For an original sequent S, we extend $<_S$ to the case of the set of variables $Vr^+(S)$ in the following way: ${}^kx <_S {}^ry$ if, and only if, $x <_S$ y. Obviously, this extension of $<_S$ is irreflexive and transitive.

Any substitution σ induces a (possibly, empty) relation $\ll_\sigma$ in the following way: $y \ll_\sigma x$ if, and only if, there exists $x \rightarrow t \in \sigma$ such that x is a dummy, the term t contains y, and y is a parameter. For example, consider a substitution $\sigma = \{{}^1x \rightarrow f({}^2y, {}^1v, {}^1z)\}$, where ${}^1x$ and ${}^1v$ are dummies and ${}^2y$ and ${}^1z$ parameters. Then, ${}^2y \ll_\sigma {}^1x$ and ${}^1z \ll_\sigma {}^1x$. The relation $\ll_\sigma$ is irreflexive and transitive.

Let S be an original sequent and let $<_S$ be the above introduced relation on the set $Vr^+(S)$. A substitution σ is strong admissible (s-admissible) for $Vr^+(S)$ if, and only if, for every $x \rightarrow t \in \sigma$, x is a dummy and the transitive closure $\lhd_{S,\sigma}$ of $<_S \cup \ll_\sigma$ is an irreflexive relation.

6.2. Quantifier-Rules-Free Calculus

Now, we have all the necessary for constructing a quantifier-rules-free calculus denoted by **pAG** being a modification of the propositional part of **mAG** that deals with so-called s-sequents consisting of ordered pairs of the form $\langle\Phi, E\rangle$, where E a set of equations and Φ is a quantifier-free formula, in which indexed variables occur only.

As in the case of **mAG**, s-sequents of **pAG** cannot contain more than one goal in their succedents.

In the below-given rules, Γ is a multiset of premises being ordered pairs, E is a set of equations ("equalities"), and Φ_1, Φ_2, and Φ are quantifier-free formulas.

Goal Splitting Rules

These rules are used for elimination of the principal logical connective from a goal.

(=> ${}^k\supset_1$)-rule: $\dfrac{\Gamma => \langle\Phi_1 \supset \Phi_2, E\rangle}{\Gamma, \langle\Phi_1, E\rangle => \langle\Phi_2, E\rangle}$ (=> ${}^k\supset_2$)-rule: $\dfrac{\Gamma => \langle\Phi_1 \supset \Phi_2, E\rangle}{\Gamma, \langle\neg\Phi_2, E\rangle => \langle\neg\Phi_1, E\rangle}$

(=> ${}^k\vee_1$)-rule: $\dfrac{\Gamma => \langle\Phi_1 \vee \Phi_2, E\rangle}{\Gamma, \langle\neg\Phi_1, E\rangle => \langle\Phi_2, E\rangle}$ (=> ${}^k\vee_2$)-rule: $\dfrac{\Gamma => \langle\Phi_1 \vee \Phi_2, E\rangle}{\Gamma, \langle\neg\Phi_2, E\rangle => \langle\Phi_1, E\rangle}$

(=> ${}^k\wedge$)-rule: $\dfrac{\Gamma => \langle\Phi_1 \wedge \Phi_2, E\rangle}{\Gamma => \langle\neg\Phi_1, E\rangle \quad \Gamma => \langle\Phi_2, E\rangle}$ (=> ${}^k\neg$)-rule: $\dfrac{\Gamma => \langle\neg\Phi, E\rangle}{\Gamma => \langle\Phi', E\rangle}$

where Φ' is the result of one-step moving the sign $\neg$ into Φ, if Φ is not an atomic formula; otherwise Φ'coincides with Φ.

Auxiliary Goal Rules

The fixing of a selected occurrence of L' produces a sequence of rules that leads to the "goal-driven" generation of an s-sequent, containing a premise. This sequence can be viewed as an application of a "large-block" inference rule initiated by L'.

In all the below-given rules, Φ_1 and Φ_2 are formulas, L and L' are literals such that $L \approx L'$ *modulo* $\Sigma(L, L')$; E, E', and E'' are sets of equations such that $E'' = E \cup E' \cup \Sigma(L, L')$.

(${}^k\supset_1$ =>)-rule: $\dfrac{\Gamma, \langle\Phi_1\lfloor L^-\rfloor \supset \Phi_2, E\rangle => \langle L', E'\rangle}{\Gamma, \langle(\neg\Phi_1)\lfloor L^+\rfloor, E\rangle => \langle L', E''\rangle \quad \Gamma => \langle\neg\Phi_2, E\rangle}$

(${}^k\supset_2$ =>)-rule: $\dfrac{\Gamma, \langle\Phi_1 \supset \Phi_2\lfloor L^+\rfloor, E\rangle => \langle L', E'\rangle}{\Gamma, \langle\Phi_2\lfloor L^+\rfloor, E\rangle => \langle L', E''\rangle \quad \Gamma => \langle\Phi_1, E\rangle}$

(${}^k\vee_1$ =>)-rule: $\dfrac{\Gamma, \langle\Phi_1\lfloor L^+\rfloor \vee \Phi_2, E\rangle => \langle L', E'\rangle}{\Gamma, \langle\Phi_1\lfloor L^+\rfloor, E\rangle => \langle L', E''\rangle \quad \Gamma => \langle\neg\Phi_2, E\rangle}$

(${}^k\vee_2$ =>)-rule: $\dfrac{\Gamma, \langle\Phi_1 \vee \Phi_2\lfloor L^+\rfloor, E\rangle => \langle L', E'\rangle}{\Gamma, \langle\Phi_2\lfloor L^+\rfloor, E\rangle => \langle L', E''\rangle \quad \Gamma => \langle\neg\Phi_1, E\rangle}$

(${}^k\wedge_1$ =>)-rule: $\dfrac{\Gamma, \langle\Phi_1\lfloor L^+\rfloor \wedge \Phi_2, E\rangle => \langle L', E'\rangle}{\Gamma, \langle\Phi_1\lfloor L^+\rfloor, E\rangle, \langle\Phi_2, E\rangle => \langle L', E''\rangle}$

(${}^k\wedge_2$ =>)-rule: $\dfrac{\Gamma, \langle\Phi_1 \wedge \Phi_2\lfloor L^+\rfloor, E\rangle => \langle L', E'\rangle}{\Gamma, \langle\Phi_1, E\rangle, \langle\Phi_2\lfloor L^+\rfloor, E\rangle => \langle L', E''\rangle}$

(${}^k\neg$ =>)-rule: $\dfrac{\Gamma, \langle\neg(\Phi\lfloor L^-\rfloor), E\rangle => \langle L', E'\rangle}{\Gamma, \langle\Phi'\lfloor L^+\rfloor, E\rangle => \langle L', E\rangle}$

where Φ' is the result of one-step moving the sign $\neg$ into Φ, if Φ is not an atomic formula; otherwise

Φ' coincides with Φ.

Termination Rules

$(=> {}^{k}\#_1)$-rule: $$\frac{\Gamma, \langle L, E\rangle => \langle L', E'\rangle}{\Gamma => \langle \#, E''\rangle}$$

where L and L' are literals such that $L \approx L'$ modulo $\Sigma(L, L')$ and $E'' = E \cup E' \cup \Sigma(L, L')$.

$(=> {}^{k}\#_2)$-rule: Let Tr be an inference search tree and Br some its branch ending by a tree leaf $\Gamma => \langle L, E\rangle$, where L is a literal. Suppose Br contains an s-sequent $\Gamma' => \langle L', E'\rangle$ such that L' is a literal and $L \approx L''$ modulo $\Sigma(L, L'')$ holds for the complement literal L'' of L' and $E'' = E \cup E' \cup \Sigma(L, L'')$. Then the s-sequent $\Gamma => \langle \#, E''\rangle$ is said to be deducible from $\Gamma => \langle L, E\rangle$ by the $(=> {}^{k}\#_2)$ -rule.

Premise-Adding Rules

$({}^{k}\rho_1 =>)$-rule: $$\frac{\langle \Phi[x], E\rangle, \Gamma => \langle \Psi, E'\rangle}{\langle {}^{k}\Phi[x], {}^{k}E\rangle, \langle \Phi, E\rangle, \Gamma => \langle \Psi, E'\rangle}$$

$({}^{k}\rho_2 =>)$-rule: $$\frac{\langle \Phi[x], E\rangle, \Gamma => \langle \Psi, E'\rangle}{\langle \neg\, {}^{k}\Psi[x], {}^{k}E'\rangle, \langle \Phi, E\rangle, \Gamma => \langle \Psi, E'\rangle}$$

where $\Phi[x]$ ($\Psi[x]$) denotes that x is some dummy occurring in Φ (Ψ). The formula ${}^{k}\Phi[x]$ (${}^{k}\Psi[x]$) is the result of the simultaneous replacement in $\Phi[x]$ ($\Psi[x]$) of the upper-left indexes of both x and all variables y such that $x <_S y$ by the natural number k being new w.r.t. an inference tree under consideration (S denotes the upper sequent of these rules); ${}^{k}E$ (${}^{k}E'$) is the result of the corresponding replacement in E (E') of the upper-left indexes in x and all such y.

Axioms. Axioms are w-sequents of the form $\Gamma => \langle \#, E\rangle$, where # denotes an empty formula.

Starting sequent. If S is an original sequent of the form $\Phi_1, \ldots, \Phi_m => \Psi_1, \ldots, \Psi_n$, then the sequent $\langle \mu({}^{1}\Phi_1), \varnothing\rangle, \ldots, \langle \mu({}^{1}\Phi_m), \varnothing\rangle => \langle \mu({}^{1}\Psi_1), \varnothing\rangle, \ldots, \langle \mu({}^{1}\Psi_n), \varnothing\rangle$ is called a starting sequent for S.

Proof Tree. Let S be an initial (original) sequent with one formula in its succedent. An inference tree Tr in the **pAG** calculus is called a proof tree in **pAG** for the starting sequent for S if, and only if, all the leaves of Tr are axioms, there exists the most general simultaneous unifier σ of all the equations from the axioms of Tr, and σ is an s-admissible substitution for $Vr+(S)$.

6.3. Main Results for pAG Calculus

Due to the w-admissibility and s-admissibility are invariant w.r.t. any inference rules applications, it is not difficult to prove the equivalence of this notions in the following sense: if for an initial (original) sequent S, Tr is a proof tree constructed in **mAG** for the corresponding input w-sequent S' and a substitution σ is w-admissible for Tr, then Tr can be converted into such a proof tree Tr' in **pAG** for the starting s-sequent $=> \langle \mu({}^{1}S), \varnothing\rangle$ that σ will be an s-admissible substitution for $Vr^{+}(S)$, and vice versa, if for an initial sequent S, Tr' is a proof tree in **pAG** for the starting s-sequent $=> \langle \mu({}^{1}S), \varnothing\rangle$ and σ is an s-admissible substitution for $Vr^{+}(S)$, then Tr' can be converted into such a proof tree Tr in **mAG** for the input w-sequent for S that σ is a w-admissible substitution for Tr. This leads to the following results.

Proposition 2. For closed formulas $\Phi_1, \ldots, \Phi_n$, and Ψ, the sequent $\Phi_1, \ldots, \Phi_n => \Psi$ is deducible in a Gentzen-type (sound and complete) calculus with standard quantifier rules (for example, in **G**) if, and only if, a proof tree can be constructed in the **pAG** calculus for the starting s-sequent $\langle \mu({}^{1}\Phi_1), \varnothing\rangle, \ldots, \langle \mu({}^{1}\Phi_n), \varnothing\rangle => \langle \mu({}^{1}\Psi), \varnothing\rangle$.

Corollary 2. A formula Ψ is valid in classical first-order logic without equality if, and only if, a proof tree can be constructed in the **pAG** calculus for the starting s-sequent $=> \langle \mu({}^{1}\Psi), \varnothing\rangle$.

Let us construct a proof tree in **pAG** for the above-given sequent S^{*}.

1. $\langle \mu({}^{1}\Phi^{*}_1), \varnothing\rangle, \langle \mu({}^{1}\Phi^{*}_2), \varnothing\rangle => \langle \mu({}^{1}\Psi^{*}), \varnothing\rangle$ (a starting s-sequent)
2. $\langle \neg R_2({}^{1}x_3), \varnothing\rangle, \langle \mu({}^{1}\Phi^{*}_1), \varnothing\rangle, \langle \mu({}^{1}\Phi^{*}_2, \varnothing\rangle => \langle (R_3({}^{1}x_2)), \varnothing\rangle$ (by $(=> {}^{k}\vee_1)$-rule from 1)
   2.1. $\langle \neg R_2({}^{1}x_3), \varnothing\rangle, \langle \mu({}^{1}\Phi^{*}_1), \varnothing\rangle, \langle \neg R_3({}^{1}x_2), \varnothing\rangle => \langle \neg R_3({}^{1}y_3), \{{}^{1}x_2 \approx {}^{1}y_3\}\rangle$ (by $({}^{k}\supset_2 =>)$-rule from 2)
   2.2. $\langle \neg R_2({}^{1}x_3), \varnothing\rangle, \langle \mu({}^{1}\Phi^{*}_1), \varnothing\rangle => \langle R_1({}^{1}y_2) \wedge \neg R_3({}^{1}x_2), \{{}^{1}x_2 \approx {}^{1}y_3\}\rangle$ (by $({}^{k}\supset_2 =>)$-rule from 2)
      2.1.1. $\langle \neg R_2({}^{1}x_3), \varnothing\rangle, \langle \mu({}^{1}\Phi^{*}_1), \varnothing\rangle => \langle \#, \{{}^{1}x_2 \approx {}^{1}y_3\}\rangle$ (by $(=> {}^{k}\#_1)$-rule from 2.1)
      2.2.1. $\langle \neg R_2({}^{1}x_3), \varnothing\rangle, \langle \mu({}^{1}\Phi^{*}_1), \varnothing\rangle => \langle \neg R_3({}^{1}x_2), \{{}^{1}x_2 \approx {}^{1}y_3\}\rangle$ (by $({}^{k}\wedge =>)$-rule from 2.2)

2.2.2. $\langle\neg R_2({}^1x_3), \varnothing\rangle, \langle\mu({}^1\Phi^*_1), \varnothing\rangle => \langle R_1({}^1y_2), \{{}^1x_2 \approx {}^1y_3\}\rangle$ (by (${}^k\wedge =>$)-rule from 2.2)

2.2.1.1. $=> \langle\#, \{{}^1x_2 \approx {}^1y_3\}\rangle$ (by ($=>{}^k\#_2$)-rule from 2.2.1 and 2)

2.2.2.1. $\langle\neg R_2({}^1x_3), \varnothing\rangle, \langle R_1({}^1x_1), \varnothing\rangle => \langle R_1({}^1y_2), \{{}^1x_1 \approx {}^1y_2, {}^1x_2 \approx {}^1y_3\}\rangle$ (by (${}^k\vee_2 \rightarrow$)-rule from 2.2.2)

2.2.2.2. $\langle\neg R_2({}^1x_3), \varnothing\rangle => \langle\neg R_2({}^1y_1), \{{}^1x_1 \approx {}^1y_2, {}^1x_2 \approx {}^1y_3\}\rangle$ (by (${}^k\vee_2 =>$)-rule from 2.2.2)

2.2.2.1.1. $\langle\neg R_2({}^1x_3), \varnothing\rangle, \langle R_1({}^1x_1), \varnothing\rangle => \langle\#, \{{}^1x_1 \approx {}^1y_2, {}^1x_2 \approx {}^1y_3\}\rangle$ (by ($=> {}^k\#_1$)-rule from 2.2.2.1)

2.2.2.2.1. $\langle\neg R_2({}^1x_3), \varnothing\rangle => \langle\#, \{{}^1x_1 \approx {}^1y_2, {}^1x_2 \approx {}^1y_3, {}^1x_3 \approx {}^1y_1\}\rangle$ (by ($=> {}^k\#_1$)-rule from 2.2.2.2)

This tree contains four branches: 1, 2, 2.1, 2.1.1; 1, 2, 2.2, 2.2.1, 2.2.1.1; 1, 2, 2.2, 2.2.2, 2.2.2.1., 2.2.2.1.1; and 1, 2, 2.2, 2.2.2, 2.2.2.2., 2.2.2.2.1. Its leaves 2.1.1, 2.2.1.1, 2.2.2.1.1, and 2.2.2.2.1 are axioms. These axioms contain the equations ${}^1x_1 \approx {}^1y_2$, ${}^1x_2 \approx {}^1y_3$, ${}^1x_3 \approx {}^1y_1$ producing the most general simultaneous unifier $\{{}^1x_1 \rightarrow {}^1y_2, {}^1x_2 \rightarrow {}^1y_3, {}^1x_3 \rightarrow {}^1y_1\}$ being an s-admissible one for $Vr^+(S^*)$. By Proposition 2, the sequent S^* is deducible in the **G** calculus.

Also draw your attention to the fact that the given inference is purely propositional.

6.4. Peculiarities of Theorem Proving and Proof Verification in EnSAD

One of the main distinguishes of **EnSAD** from **RuSAD** is that the **EnSAD** system allows verifying a proof of a theorem written in **ForTheL** and inserted into a self-contained **ForTheL** environment, while the **RuSAD** system is intended only for proving a theorem under consideration. In this connection, for solving a verification task, there is in **EnSAD** a module for generating goals that are sequentially passed to the logical engine of **EnSAD** for automatic establishing the validity of a goal in question. A proof is regarded to be correct, if all the generated goals are valid. (This module becomes useless in the case, if **EnSAD** solves the task of automated theorem proving.) Another distinguish is that along with its native engine, **EnSAD** gives the possibility to use one of such of well-known provers as Vampire, SPASS, E Prover, Otter, and Prover9. (Remind that **RuSAD** contained only its own resolution engine based on the negative hyperresolution.)

6.4.1. Architecture of EnSAD

The action scheme of **EnSAD** could be described as follows. A user communicates with it using texts written in **ForTheL**. He may submit a problem like “prove the following proposition” or “verify whether the given mathematical text is correct”. The text, provided it is syntactically correct, is sent to a subsystem, a so-called “reasoner”. The reasoner makes analysis of a problem under consideration and formulates a number of tasks submitting them to an the **EnSAD** logical engine being a prover. If the prover finishes the job, the result of its work (e.g. a proof verification trace) is displayed to the user and the work is over. If it fails, then a diagnostic is made and its result supplies to the reasoner for repairing the situation. In particular, the reasoner can decide that a certain auxiliary proposition might be useful and starts the search for those in existing mathematical archives. After finding it, the service begins a new proof search cycle with a modified problem and the process goes on.

According to this scheme, the **EnSAD** system initially conceived as a linguistic-deductive system designed to assist a mathematician in his scientific and teaching activities, has three levels of the processing of input **ForTheL**-texts (see Fig. 2).

The architecture of **EnSAD** having three levels structure, a short description of which are given below.

At the first (linguistic) level, the Parser module accepts a **ForTheL** text, checks its syntactical correctness and converts the text into a normalized form for a further processing. The result of translation is a number of goal statements to be sequentially deduced from their predecessors. The FOL submodule is a parser for a “dialect” of the first-order language. The **EnSAD** system can also connect with the famous TPTP library [21] for receiving theorem-proving tasks.

At the second (reasoning) level, the Verification Manager module scans the normalized text sentence by sentence. Each sentence is first sent to the “evidence collector”, which accumulates so-called term properties for terms occurred in the sentence. Term properties are literals that tell the system something important about a given term occurrence. The most important mission of term properties is to hold information about term “types”, which is usually expressed by a statement of the form “t is a notion”. Some simple properties, like the nonemptiness, are also highly useful.

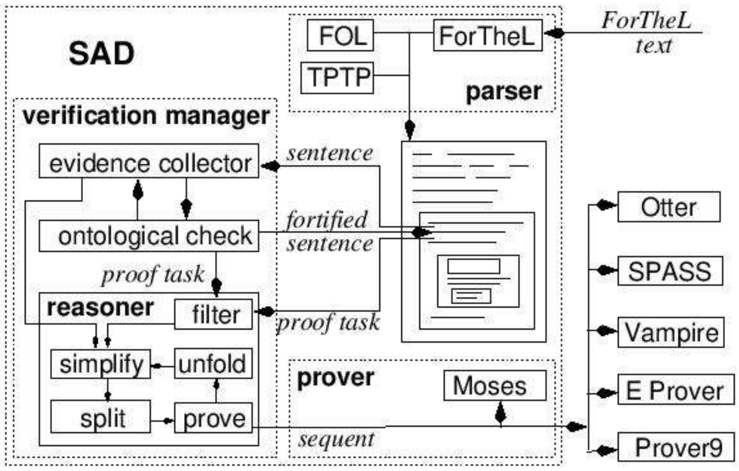

Figure 2: **Architecture of the EnSAD system**

At the second (reasoning) level, the Verification Manager module scans the normalized text sentence by sentence. Each sentence is first sent to the “evidence collector”, which accumulates so-called term properties for terms occurred in the sentence. Term properties are literals that tell the system something important about a given term occurrence. The most important mission of term properties is to hold information about term “types”, which is usually expressed by an atomic statement of the form “t is a notion”. Some simple properties, like the nonemptiness, are also highly useful.

The Evidence Collector submodule is a simple syntactical procedure that scans the context of a given occurrence and checks what can be “easily” deduced from the properties already known. For example, let S be declared as a set of integer numbers and x be declared as an element of S. Then, anywhere in view of these declarations, the term -x will be known to be an integer.

The Ontological Checker submodule uses certain ontological connections between notions occurred in a text for fortifying its certain properties with the purpose to form a certain proof task.

Proof tasks are sequentially processed by the Reasoner module. In the verification mode, Reasoner is intended for splitting a proof task in question into a number of subtasks for a prover. It either makes reduction of the main goal to several simpler subgoals or proposes an alternative subgoal. In particular, its toolkit contains some simplification methods on the propositional level. This module is redundant in the case, when the **EnSAD** system solves an automated theorem-proving problem.

At the third (deductive) level, the Prover module carries out a proof search in classical first-order logic with equality using its own Moses prover or one of the following (external w.r.t. **EnSAD**) provers: Vampire [22], Otter [23], SPASS [24], and E Prover [25]. Remind that the Moses prover is based on a goal-driven sequent calculus exploiting the notion of an s-admissible substitution, which permits to preserve the initial signature of a task in question so that equations accumulated during proof search can be sent to a specialized solver, e.g. to an external computer algebra system.

At the final stage, **EnSAD** outputs the result of its session. Note a user can influence to solving a task under consideration by changing some system parameters.

Now, the **EnSAD** system can perform the following:

• Inference Search: establishing of deducibility of a first-order formula/sequent;

• Theorem Proving: proving of a proposition in the context of a **ForTheL**-text;

• Text Verification: verifying of a self-contained mathematical **ForTheL**-text.

A theorem *T* is called **EnSAD**-valid in a context of a **ForTheL**-text *Txt* if, and only if, **EnSAD** can prove *T* using only facts (notions, definitions, lemmas, etc.) from the text *Txt* in the assumption that it has an infinite memory to store data and that it has an infinite time to operate.

The **EnSAD** system is sound and logically correct in the following sense.

Theorem 2. Let *Txt* be a self-contained noncontradictory **ForTheL**-text for a theorem *T* and Txt^* => T^* be a sequent, in which Txt^* and T^* are the results of the translation of respectively *Txt* and *T* into first-order formulas. Then *T* is an **EnSAD**-valid theorem in the context of the **ForTheL**-text *Txt* if, and only if, the sequent Txt^* => T^* is deducible in a Gentzen-type sound and complete calculus with standard quantifier rules.

A proof *Pr* of a theorem *T* is **EnSAD**-correct in a context of a **ForTheL**-text *Txt* if, and only if, **EnSAD** can verify *Pr* using only facts (notions, definitions, lemmas, etc.) from the text *Txt* in the assumption that it has an infinite memory to store data and that it has an infinite time to operate.

Theorem 3. Let *Txt* be a self-contained noncontradictory **ForTheL**-text for a theorem *T* together with its proof *Pr* and Txt^* => T^* be a sequent, in which Txt^* and T^* are the results of the translation of respectively *Txt* and *T* into first-order formulas. If the proof Pr is **EnSAD**-correct in the context of the **ForTheL**-text *Txt*, then the sequent Txt^* => T^* is deducible in a Gentzen-type sound and complete calculus with standard quantifier rules.

6.4.2. Examples of theorem proving and proof verification in EnSAD

Below, two examples of the processing of **ForTheL**-texts (being available on the web-site at the pages http://nevidal.org/help-thm.en.html and http://nevidal.org/help-txt.en.html) are presented. The first demonstrates the ability of **EnSAD** not only to prove theorems, but also to establish the validity of different logical task, for example, the validity of the childish statement known as Schuberts Steamroller Problem and given as Proposition that concerns the relationships between animals and plants. The second one shows the ability of the **EnSAD** system to verify a given proof of theorems relating to Number Theory and inserted into a self-contained **ForTheL**-text. (In the first case, the Moses prover was used and in the second one, the SPASS prover was used.)

6.4.2.1. Example of solving Schubert's steamroller problem

Receiving the below-given problem given in Proposition, **EnSAD** processes it and outputs the result on this session and some statistical data.

```
============================================================
[animal/-s] [plant/-s] [eat/-s]
Signature Animal. An animal is a notion.
Signature Plant. A plant is a notion.
Let A,B denote animals. Let P denote a plant.
Signature EatAnimal. A eats B is an atom.
Signature EatPlant. A eats P is an atom.
Signature Smaller. A is smaller than B is an atom.
Axiom CruelWorld. Let B be smaller than A and eat some plant. Then A eats all plants
or A eats B.
Signature Wolf. A wolf is an animal.
Signature Fox. A fox is an animal smaller than any wolf.
Signature Bird. A bird is an animal smaller than any fox.
Signature Worm. A worm is an animal smaller than any bird.
Signature Snail. A snail is an animal smaller than any bird.
Signature Grain. A grain is a plant.
Axiom Everybody. There exist a wolf and a fox and a bird and a worm and a snail
and a grain.
Axiom WormGrain. Every worm eats some grain.
Axiom SnailGrain. Every snail eats some grain.
Axiom BirdWorm. Every bird eats every worm.
Axiom BirdSnail. Every bird eats no snail.
Axiom WolfGrain. Every wolf eats no grain.
Axiom WolfFox. Every wolf eats no fox.
Proposition. There exist animals A,B such that A eats B and B eats every grain.
============================================================
 [ForTheL] stdin: parsing successful
[Reason] stdin: theorem proving started
[Reason] line 32: goal: There exist animals A,B such that A eats B and B eats
```

```
   every grain.
[Reason] stdin: theorem proving successful
[Main] sections 45 - goals 1 - subgoals 3 - trivial 1 - proved 1
[Main] symbols 68 - checks 58 - trivial 57 - proved 0 - unfolds 0
[Main] parser 00:00.00 - reason 00:00.00 - prover 00:00.00/00:00.00
[Main] total 00:00.01
=================================================================
```

6.4.2.2. Example of verifying a proof of a theorem relating to Number Theory

In this example, iif is an abbreviation for “if and only if”. A test beginning with # is a comment that a user can write for a better understanding of a given ForTheL-text.

```
=================================================================
#
# Axioms of zero and the successor
#
[number/-s]
Signature NatSort.  A number is a notion.
Let A,B,C stand for numbers.
Signature NatZero.  The zero is a number.
Let X is nonzero stand for X is not equal to zero.
Signature NatSucc.  The successor of A is a nonzero number.
Axiom SuccEquSucc.
   If the successor of A is equal to the successor of B
   then A and B are equal.
Signature NatSum.   The sum of A and B is a number.
Axiom AddZero.      The sum of A and zero is equal to A.
Axiom AddSucc.      The sum of A and the successor of B
      is equal to the successor of the sum of A and B.
#
# We take the following facts as axioms, too
#
Axiom ZeroOrSucc.
   Every nonzero number is the successor of some number.
Axiom AssoAdd.
   The sum of A and the sum of B and C is equal to the sum of (the sum of A and B)
      and C.
Axiom InjAdd.
   If the sum of A and B is equal to the sum of A and C then B and C are equal.
Axiom Diff.
   There exists C such that
      A is the sum of B and C or B is the sum of A and C.
#
# Definition of order on natural numbers
#
Definition DefLess.
   A is less than B  iff  B is equal to the sum of A and the successor of some number.
Let X is greater than Y stand for Y is less than X.
#
# Theorems with basic properties of order
#
Theorem NReflLess.
   A is not less than A.
Proof.
   Assume the contrary.
   Take a number C such that A is equal to the sum of A and the successor of C.
   Then the successor of C is zero (by AddZero,InjAdd).
   We have a contradiction.
qed.

Theorem TransLess.
   Assume A is less than B and B is less than C.
   Then A is less than C (by DefLess).
Proof.
   Let M be a number and N be the successor of M.
   Let P be a number and Q be the successor of P.
   Assume the sum of A and N is equal to B.
```

```
    Assume the sum of B and Q is equal to C.
    Let S be the sum of N and Q.
    S is the successor of the sum of N and P (by AddSucc).
    The sum of A and S is equal to C (by AssoAdd).
  qed.

  Theorem ASymmLess.
    If B is less than A then A is not less than B.

  Theorem TotalLess.
    Let A,B be nonequal.
    Then A is less than B or B is less than A.
  Proof.
    Take C such that A is the sum of B and C or B is the sum of A and C.
    If C is zero then B is equal to A.
    Hence C is the successor of some number.
    If B is the sum of A and C then A is less than B.
    Then A is the sum of B and C or A is less than B.
    If A is the sum of B and C then B is less than A.
    Hence the thesis.
  qed.
  =========================================================
```

After receiving this ForTheL-text, **EnSAD** establishes the correctness of the theorem proofs and output the below-given verification trace ended by statistical data.

```
=========================================================
[ForTheL] stdin: parsing successful
[Reason] stdin: verification started
[Reason] line 46: goal: Take a number C such that A is equal to the sum of A and the successor of C.
[Reason] line 48: goal: Then the successor of C is zero (by AddZero,InjAdd).
[Reason] line 49: goal: We have a contradiction.
[Reason] line 43: goal: A is not less than A.
[Reason] line 61: goal: S is the successor of the sum of N and P (by AddSucc).
[Reason] line 62: goal: The sum of A and S is equal to C (by AssoAdd).
[Reason] line 54: goal: Then A is less than C (by DefLess).
[Reason] line 66: goal: If B is less than A then A is not less than B.
[Reason] line 72: goal: Take C such that A is the sum of B and C or B is the sum of A and C.
[Reason] line 73: goal: If C is zero then B is equal to A.
[Reason] line 74: goal: Hence C is the successor of some number.
[Reason] line 75: goal: If B is the sum of A and C then A is less than B.
[Reason] line 76: goal: Then A is the sum of B and C or A is less than B.
[Reason] line 77: goal: If A is the sum of B and C then B is less than A.
[Reason] line 78: goal: Hence the thesis.
[Reason] line 70: goal: Then A is less than B or B is less than A.
[Reason] stdin: verification successful
[Main] sections 65 - goals 16 - subgoals 19 - trivial 2 - proved 14
[Main] symbols 128 - checks 89 - trivial 89 - proved 0 - unfolds 5
[Main] parser 00:00.01 - reason 00:00.02 - prover 00:10.18/00:00.07
[Main] total 00:10.22
=========================================================
```

7. Current state and possible future work

In general, the entire time of research on **EA** can be divided into the following stages:

- 1962-1969: pre-attempts to follow **EA**;
- 1970-1992: research on **EA** leaded to the construction and trial operation of **RuSAD**;
- 1998-2008: research on **EA** leaded to the construction and trial operation of **EnSAD**;
- 2009-present: investigations on **EA** on a computer-oriented proof search in non-classical logics.

And although the further development of **EnSAD** was stopped in 2008, anyone can carry out a series of experiments with the system that is available online on “nevidal.org/sad.en.html”.

Note that by now, a number of tests has been made with the **EnSAD** system. They are related to proof search in first-order logic, theorem proving in the **ForTheL**-environment, and verification of self-contained **ForTheL**-texts. The most interesting ones concern verification, among which are: Newman's lemma, Cauchy-Bouniakowsky-Schwarz inequality from mathematical analysis, Ramsey’s finite and infinite theorems, Chinese remainder theorem, Bezouts identity in terms of abstract rings, Tarskis fixed point theorem, Furstenbergs proof of the in finitude of primes, and some properties of finite groups.

The trial operation of the **EnSAD** system and a number of current achievements made in automated reasoning in the **EA**-style have shown the desirability of improving the capabilities of **EnSAD** in the following directions (studied and not implemented).

On the linguistic level. The nearest objective can be the incorporation of the existing **ForTheL** language into a LaTeX-environment to reach the reading of **ForTheL**-texts in the form closest to usual mathematical texts. Besides, there are drafts of Russian and Ukrainian versions of the **ForTheL** language. Therefore, there exists the possibility to construct the next bidirectional translators: **English ForTheL**-texts ⇔ **Russian ForTheL**-texts, English **ForTheL**-texts ⇔ **Ukrainian ForTheL**-texts, and **Russian ForTheL**-texts ⇔ **Ukrainian ForTheL**-texts, which will give the opportunity for using such a multilingual extension of **EnSAD** by a person who knows only one of these languages as well as for making an automatic translation from one of these languages into another. (Of course, one can try to construct a German, French, and/or other version of the **ForTheL** language, thereby strengthening this multilingual **EnSAD** component.)

On the reasoning level. The improving of heuristic possibilities of the **EnSAD** system is presupposed to do by incorporating in **EnSAD** the human-like reasoning methods depending on the subject domain in question concentrating the main attention on inductive theorem proving methods.

On the deductive level. On the basis of the research made on computer-oriented proof search in classical and non-classical sequent logics, one can try to construct a toolkit giving the possibility to “puzzle” one or another system logical engine depending on a desire of an EnSAD user or a subject domain under consideration.

8. Conclusion

Features of the **EnSAD** system indicate that **EnSAD** is designed and implemented with taking into account modern achievements in the field of construction of computer mathematical services. In this regard, we draw attention to that the **ForTheL** language is based on fundamental logical and set-theoretic relations. Therefore, it is suitable for representing any (not only mathematical) texts, if the latter are formalized in terms of first-order logic. A **ForTheL**-text can be created either by a human or a computer and after this it can be sent to **EnSAD** or even another system directly or via a network.

As for the deductive component of **EnSAD**, the existing theoretical results and the experience accumulated during the trial operation of **EnSAD** can serve as a good basis for a further improvement of the **EA**-style deductive technique in the direction of increasing the efficiency of inference search in classical logic and making reasoning in non-classical logics.

Of particular note is the ability of **EnSAD** to combine deduction with analytical transformations performed by a system external to **EnSAD**. This is provided by a special proof technique and properties of the **EA**-style deductive formalism permitting to preserve the signature of an original problem and make equality handling in isolation from deduction.

In the long run, the **EA**-approach to automated reasoning and further development of the **EnSAD** system can lead to the creation of an info-structure for the remote multilingual presentation and complex processing of mathematical knowledge, what can make the system useful for both teaching and academic daily activity of a person.

9. References

[1] V.M. Glushkov. “Some problems in the theories of automata and artificial intelligence.” Cybernetics and System Analysis, Vol. 6, Issue 2, Springer, New York, 1970: 17-27. DOI: 10.1007/BF01070496.

[2] P.C. Gilmore. “A program for the production of proofs for theorems derivable within the first-order predicate calculus from axioms.” Proceedings of the International Conference on

Information Processing, Unesco, Paris, June 15-20 (1959): 265-273.

[3] Hao Wang. “Towards mechanical mathematics.” IBM Journal of Research and Development, Vol. 4, Issue 1, Jan. (1960): 2-22. DOI: 10.1147/rd.41.0002.

[4] J.A. Robinson, and A. Voronkov. “Handbook of Automated Reasoning” (in 2 volumes). Elsevier and MIT Press, June (2001): 2122 pp. ISBN: 0-444-50813-9.

[5] V. Lifschitz. “Mechanical theorem proving in the USSR: The Leningrad school.” Delphic Associates Inc., Jan. (1986): 206 pp.

[6] G. Mints. “Proof theory in the USSR (1925-1969).” Journal of Symbolic Logic, Vol. 56, Issue 2, June (1991): 385-424. DOI: 10.2307/2274689

[7] F. V. Anufriyev. “Algoritm poiska dokazatel'stv teorem v logicheskikh ischisleniyakh.” [An algorithm of search for proofs of theorems in logical calculi.] Teoriya Avtomatov, GIC AN USSR, Kiev, 1969: 3-26. (In Russian).

[8] S. Kanger. “Simplified proof method for elementary logic.” Studies in Logic and the Foundations of Mathematics, North-Holland Publishing Company, Amsterdam, 1963: 87-94.

[9] G. Gentzen. “Untersuchungen uber das logische schliessen. I.” Mathematische Zeitschrift, Vol. 39, 1935: 176–210. https://doi.org/10.1007/BF01201353.

[10] V. M. Glushkov, Yu. V. Kapitonova, A. A Letichevskii, K. P. Vershinin, and N. P. Malevanyi. “Construction of a practical formal language for mathematical theories.” Cybernetics and Systems Analysis, Vol. 8, Issue 5, 1972: 730-739. https://doi.org/10.1007/BF01068445

[11] J. Robinson. “A machine-oriented logic based on resolution principle.” Journal of the ACM, Vol. 12, Issue 1, 1965: 23-41. DOI: 10.2307/2270500.

[12] A. Lyaletski, K. Verchinine, A. Degtyarev, and A. Paskevich. “System for Automated Deduction (SAD): Linguistic and deductive peculiarities.” Advances in Soft Computing (Intelligent Information Systems, 11th International Symposium, IIS 2002, Sopot, Poland, June 2002), 2002: 413-422. DOI: 10.1007/978-3-7908-1777-5_44.

[13] K. Vershinin, and A. Paskevich. “ForTheL – the language of formal theories.” International Journal of Information Theories and Applications, Volume 7, Issue 3, 2000: 120-126.

[14] K. Vershinin, A. Lyaletski, and A. Paskevich. “System for Automated Deduction (SAD): A tool for proof verification.” Proceedings, Lecture Notes in Computer Science (Automated Deduction - CADE-21, 21st International Conference on Automated Deduction, Bremen, Germany, July 17-20, 2007), Vol. 4603, 2007: 398-403.

[15] A. Paskevych. Méthodes de formalisation des connaissances et des raisonnements mathématiques: aspects appliqués et théoriques. PhD thesis, Univ. Paris 12, 2007. (In French.)

[16] A. Lyaletski. “Mathematical text processing in EA-style: a sequent aspect.” Journal of Formalized Reasoning (Special Issue: Twenty Years of the QED Manifesto), Vol. 9, Issue 1, 2016: 235-264. DOI: 10.6092/issn.1972-5787/4569.

[17] A. Lyaletski, M. Morokhovets, and A. Paskevich. “Kyiv school of automated theorem proving: a historical chronicle.” Logic in Central and Eastern Europe: History, Science, and Discourse. University Press of America, 2012: 431-469.

[18] J.H. Gallier. “Logic for computer science: Foundations of automatic theorem proving.” Dover Publications, June (2015): 534 pp. ISBN-10: 0486780821.

[19] G. Mints. “Teorema Erbrana”. [Herbrand theorem.] Matematicheskaya Teoriya Logicheskogo Vyvoda, Nauka, Moskva, 1967: 311-350. (In Russian.)

[20] A. Lyaletski. “Gentzen calculi and admissible substitutions.” Actes Preliminaieres, du Symposium Franco-Sovietique “Informatika-91”, Grenoble, France (October 16-19, 1991), 1991: 99-111.

[21] The TPTP problem library for automated theorem proving. URL: http://www.tptp.org/.

[22] Vampire's Home Page. URL: http://www.vprover.org/.

[23] Otter homepage. URL: https://www.mcs.anl.gov/research/projects/AR/otter/.

[24] SPASS theorem prover. URL: https://www.mpi-inf.mpg.de/departments/automation-of-logic/software/spass-workbench/classic-spass-theorem-prover.

[25] The E Theorem Prove. URL: https://wwwlehre.dhbw-stuttgart.de/~sschulz/E/E.html.