# SAD, a System for Automated Deduction: a Current State

Konstantin Verchinine[1], Anatoli Degtyarev[2], Alexander Lyaletski[3], and Andrey Paskevich[3]

[1] Math-Info Department, Paris 12 University,
61, avenue du General De Gaulle, 94010 Creteil, France
tel.: +33 (6) 851 291 2022, e-mail: verko@logique.jussieu.fr

[2] Department of Computer Science, University of Liverpool,
Liverpool L69 7ZF, United Kingdom
tel.: +44 151 794 6792, e-mail: a.degtyarev@cs.man.ac.uk

[3] Cybernetics Department, Taras Shevchenko Kyiv National University,
2, Glushkov avenue, building 6, 03022 Kyiv, Ukraine
tel.: +38 (044) 266 3108, e-mail: lav@tc.unicyb.kiev.ua, andrey@raptor.kiev.ua

**Abstract.** In this paper [4] a state-of-the-art of a system of automated deduction called SAD is described. An architecture of SAD corresponds well to a modern vision of the Evidence Algorithm programme advanced by Academician V.Glushkov. The system is intended for accumulating mathematical knowledge and using it in a regular and efficient manner for processing a self-contained mathematical text for "doing mathematics", in particular, for proving a given statement that always is considered as a part of a self-contained mathematical text. The specific feature of SAD is that a mathematical text under consideration are formalized using a certain formal language, which is closely connected with a "natural" one and can be translated into a kind of the first-order language in order to use a special sequent technique having the following peculiarities: goal-oriented reducing an assertion to be proven to a number of auxiliary assertions, efficient logical inferring in a signature of an initial theory using a special quantifier technique, and separating deduction from equality processing (equation solving). The language may be used as a tool to write and verify mathematical papers, theorems, and formal specifications, to perform model checking, and so on. The calculus is oriented to constructing some natural proof search methods such as, for example, using definitions and applying auxiliary propositions.

## 1 Introduction

This paper is devoted to the description of a state-of-the-art of a system for automated deduction. An architecture of the system corresponds well to a modern vision of the Evidence Algorithm, EA, advanced by Academician V.Glushkov [1] as a programme of automated theorem-proving in mathematics. Its main objective was to assist to working mathematicians in mathematical text processing, i.e. in computer-aided constructing and verifying long, but in some sense "evident" proofs. That is why V. Glushkov proposed to explore simultaneously formalized languages for presenting mathematical texts in the form most appropriate for a user, a formal notion of evolutionary developing computer-made proof step, EA information environment that has influence on the evidence of a proof step, and man-assisted search for a proof.

Ideas underlying EA correspond well to the contemporary trends of computer mathematical services construction. In this connection, investigations of EA-style mathematical text processing are renewed. As a result, the system for automated deduction called SAD is constructed now at a new level of understanding the problem of automated theorem-proving, which takes into account existing approaches to developing systems for "doing mathematics". Here we note that the name SAD is taken from the reason to underline continuity in investigations concerning a system for

[4] Partially supported by INTAS 2000-447.

automated theorem-proving, which was implemented under the same name in Russian [2] at the end of 1970s and which can be considered as the first step to the realization of main principles of the programme EA (some historical remarks concerning EA can be found in [3]).

The system SAD is intended for accumulating mathematical knowledge and using it in a regular and efficient manner for processing a self-contained mathematical text in order to prove a given statement that always is considered as a part of the text. Note that the term "a mathematical text is self-contained" denotes that the text contains all the necessary for proving a theorem under consideration.

Two peculiarities are inherent in SAD: (a) mathematical texts under consideration are formalized using a special formal language, which is closely connected with natural languages used in mathematical publications; (b) proof search is based on a specific sequent-type calculus, which is oriented to a possibility to formalize "natural reasoning style". In this connection, the paper is devoted mainly to the description of these peculiarities.

## 2 Languages in SAD

According to EA, the core of a mathematical text processing technique is a so-called "evidence routine" that establishes the evidence of a proven (verified) step in the terms of some deduction technique. To achieve this, various tools enforcing the "evidence routine" should be provided: search for auxiliary assertions or some other relevant information, use of analytical transformations (i.e. use of computer algebra systems in a modern terminology), application of a proof technique usual for a man, and so on. And all these tools should be able to exchange data using a certain formalized language (languages).

A number of languages have been constructed for EA. All of them were based on the following principles. These languages should have a formalized syntax and semantics. They should permit to write theory axioms, definitions, theorems, and proofs in order to provide getting self-contained texts. Accordingly, a thesaurus of the languages should be separated from its grammar to be extendible. Besides, the languages should be close to natural ones used in mathematical papers. This provides a user-friendly interface for creating a text or processing it in an interactive mode.

The first sketch of these languages was given in [4]. Then, a so-called language TL was constructed under the direction of V. Bodnarchuk, and its description appeared in [5] as the result of investigations into EA carried out by a research group led by K. Vershinin at the end of 1970s.

By now, ForTheL (FORmal THEory Language) [6] is the last representative of the above languages. The main objective of the construction of ForTheL is to provide an initial environment for evolutionary development of the "evidence routine" and to enforce proving tools of EA in dependence on a problem to be solved and a mathematical text to be processed. Thus, ForTheL can be used as a tool to write and to verify mathematical publications and formal specifications. Also, it can be used as a universal interface for exchanging data between different kinds of declarative mathematical knowledge bases and as a tool for integrating computer mathematical services.

Any ForTheL-text can be viewed as a collection of phrases possessed of some structure that provides certain "navigation rules" in the text. Therefore, it seems to be reasonable to distinguish two components of the language ForTheL, which are orthogonal each other in some sense. The first component may be considered as a language of ForTheL-phrases, and the second one defines a structure of a whole ForTheL-text: that is it contains rules for aggregating phrases, for naming sections, for defining scopes of variables, etc.

Let us give a brief description of syntax and semantics of ForTheL.

*SYNTAX OF FORTHEL.* From the syntactical point of viwe, any ForTheL-text is a collection of sections. In its turn, each section can contain lower level sections and/or phrases. Boundaries of sections are marked with separators that additionally define a partial order being a kind of a subordination relation in a set of all sections of a text under consideration. A syntax of separators can be enough flexible, but we prefer to follow the usual practice of notation accepted in mathematical papers.

Some sections, for example, theorems, proofs, and definitions play a special role in the language. They are distinguished by a special syntax of their separators, which also reflects the usual mathematical practice.

Sections containing only phrases are called “primary”. Any phrase either affirms or asserts something. The distinction between affirmations and assertions is also expressed with simple syntactical rules.

*SEMANTICS OF FORTHEL.* A semantics of a ForTheL-text is determined by a task solved by a user. Some examples of such a task are: “check an appropriateness of a text”, “check a soundness of a statement in a text”, “find a counter example for a statement in a text”, “find out a subject of a text”, “search for some information in a text, which are relevant to ...”, and so on.

Let us consider, for example, the second task. Let a statement under consideration be sound in a text if it can be formally deduced from its “predecessors”.

This task requires a proper definition of the notion of “logical predecessor”, which can be easily expressed with inspecting a text, where separators play the role of road signs, and with replacing sections by their “main affirmations”: as a result, a section “theorem” can be replaced by its formulation.

*FORTHEL1-ENVIRONMENT.* According to a current approach to proof search in the EA-style, a mathematical text to be processed should be written as a ForTheL-text. Then, a ForTheL-text under consideration is automatically transformed into a so-called ForTheL1-text, which consists of sentences that, on the one hand, are analogs of 1st-order logic formulas, and, on the other hand, preserves a signature of the original ForTheL-text, its syntax and structure, i.e. a partition into sections such as definitions, auxiliary propositions, and a theorem to be proved. After transforming a ForTheL-text into a ForTheL1-text, the construction of an information ForTheL1-environment for proof search procedures can be made.

## 3 Deduction in SAD

Deduction in SAD requires a special sequent formalism using a proof environment constructed on the base of a ForTheL1-text containing definitions, auxiliary propositions, and a theorem to be proven. Here we note that the first paper on an EA-style heuristic procedure for theorem proof search in Group Theory appeared in 1966 [8]. In that paper an attempt was done to make allowance for some proof search methods used in mathematical papers at an informal level. Then that formal technique was extended to certain fragments of Set Theory. Its final completion appeared as a specific calculus [9], which was meant for ascertaining the validity of 1st-order classical logic formulas. Its further development performed by A.Degtyarev and A.Lyletski gave arise to the first representative [10] of an EA-style family of so-called a-sequent calculi, which was ”extended” to a number of a-sequent calculi later (see, for example, [11, 12]). That is why a deductive technique of SAD is based on a subsequent modification of a sequent formalism described in [11] as the Gentzen-type calculus gS.

The most important changes in the modification concern quantifiers rules and handling premises of sequents. Unlike the calculus gS, where new premises can be added to an a-sequent, an initial set of premises stays the same when inference search is made in the modification described below as a calculus GD. Also note that in GD a set of equations (constraints) is changed every time when a so-called auxiliary goal rule is applied. As to quantifier rules, we must note that to avoid irrelevant duplications of premises, which are observed in gS, a special technique of bound variables processing has been constructed.

*PRELIMINARIES.* We consider classical first-order logic with the universal and existential quantifiers and with the propositional connectives of implication ($\supset$), disjunction ($\vee$), conjunction ($\wedge$), and negation ($\neg$). Below, atomic formulas and literals are denoted by $L$ or $M$, formulas are denoted by $F$, $G$, or $P$, and the empty formula denotes by $\#$. All letters can be subscripted.

The expression $F^{\neg}$ denotes the result of one-step carrying of the negation into a formula $F$: $(\forall x P)^{\neg}$ is $\exists x \neg P$, $(\exists x P)^{\neg}$ is $\forall x \neg P$, $(P \supset Q)^{\neg}$ is $P \wedge \neg Q$, $(P \vee Q)^{\neg}$ is $\neg P \wedge \neg Q$, $(P \wedge Q)^{\neg}$ is $\neg P \vee \neg Q$, $(\neg P)^{\neg}$ is $P$, and $A^{\neg}$ is $\neg A$.

We define *positive* ($P\lfloor F^{+}\rfloor$) *and negative* ($P\lfloor F^{-}\rfloor$) *occurrences* of a formula $F$ in a formula $P$ :in the usual way. by the following:

I. $F\lfloor F^{+}\rfloor$ always holds.

II. If $F$ occurs in $P$ then:

| | | | | | |
|---|---|---|---|---|---|
| $P\lfloor F^{+}\rfloor$ | implies | $(\neg P)\lfloor F^{-}\rfloor$ | $P\lfloor F^{-}\rfloor$ | implies | $(\neg P)\lfloor F^{+}\rfloor$ |
| $P\lfloor F^{+}\rfloor$ | implies | $(P \wedge Q)\lfloor F^{+}\rfloor$ | $P\lfloor F^{-}\rfloor$ | implies | $(P \wedge Q)\lfloor F^{-}\rfloor$ |
| $P\lfloor F^{+}\rfloor$ | implies | $(Q \wedge P)\lfloor F^{+}\rfloor$ | $P\lfloor F^{-}\rfloor$ | implies | $(Q \wedge P)\lfloor F^{-}\rfloor$ |
| $P\lfloor F^{+}\rfloor$ | implies | $(P \vee Q)\lfloor F^{+}\rfloor$ | $P\lfloor F^{-}\rfloor$ | implies | $(P \vee Q)\lfloor F^{-}\rfloor$ |
| $P\lfloor F^{+}\rfloor$ | implies | $(Q \vee P)\lfloor F^{+}\rfloor$ | $P\lfloor F^{-}\rfloor$ | implies | $(Q \vee P)\lfloor F^{-}\rfloor$ |
| $P\lfloor F^{+}\rfloor$ | implies | $(P \supset Q)\lfloor F^{-}\rfloor$ | $P\lfloor F^{-}\rfloor$ | implies | $(P \supset Q)\lfloor F^{+}\rfloor$ |
| $P\lfloor F^{+}\rfloor$ | implies | $(Q \supset P)\lfloor F^{+}\rfloor$ | $P\lfloor F^{-}\rfloor$ | implies | $(Q \supset P)\lfloor F^{-}\rfloor$ |
| $P\lfloor F^{+}\rfloor$ | implies | $(\forall x P)\lfloor F^{+}\rfloor$ | $P\lfloor F^{-}\rfloor$ | implies | $(\forall x P)\lfloor F^{-}\rfloor$ |
| $P\lfloor F^{+}\rfloor$ | implies | $(\exists x P)\lfloor F^{+}\rfloor$ | $P\lfloor F^{-}\rfloor$ | implies | $(\exists x P)\lfloor F^{-}\rfloor$ |

In what follows, $W$ denotes a set of first-order formulas.We assume that no two quantifiers in formulas from $W$ contain the same variable. Also, we assume that the notion of the scope of a quantifier is known to a reader.

For every variable $v$ from $W$, we introduce a countable set of new variables of the form ${}^{k}v$, where $k = 0, 1, 2, \ldots$ These new variables are called *indexed*.

Any expression under consideration can contain both an indexed and unindexed variables.

A variable $v$ is said to be *unknown w.r.t. W* if there exists a formula $P \in W$ such that $P\lfloor(\forall v F)^{+}\rfloor$ or $P\lfloor(\exists v F)^{-}\rfloor$ holds. Correspondingly, $v$ is said to be *fixed w.r.t. W* if there exists $P \in W$ such that $P\lfloor(\forall v F)^{-}\rfloor$ or $P\lfloor(\exists v F)^{+}\rfloor$ holds. Obviously, all variables in formulas from $W$ are either unknown or fixed (in [13] they are called "dummies" and "parameters").

A set $W$ of formulas induces an antisymmetric relation $\prec_W$ on a set of bound variables from $W$ by the following: $u \prec_W w$ holds if and only if for some formula $F$ from $W$ a quantifier from $F$ containing $w$ occurs in the scope of a quantifier from $F$ containing $u$.

We treat the notion of a substitution as in [14]. Any substitution component is considered to have the form $t/x$, where $x$ is a variable (denominator), and $t$ is a term (numerator) of a substitution. Also, we assume that a reader is familiar with the notion of a simultaneous unifier of sets of expressions.

Every substitution $\sigma$ defines an *antisymmetric relation* $\ll^{\sigma}_{W}$ in the following way: $w \ll^{\sigma}_{W} u$ if and only if $t/u \in \sigma$, $u$ is an unknown variable w.r.t. $W$, $w$ is fixed w.r.t. $W$, and $t$ is a term containing $w$.

A substitution $s$ is said to be *admissible* for a set $W$ of formulas if and only if (i) denominators of $\sigma$ are unknown variables w.r.t. $W$ and (ii) the transitive closure of $\prec_W \cup \ll^{\sigma}_{W}$ is an antisymmetric relation.

*Remark.* It may be checked that the above admissible substitution notion is equivalent to the one from [15] (also, see [11]).

A *pasting substitution* for a set $W$ of formulas is a substitution $\pi$, which satisfies the following condition: every its component has the form ${}^{m}v/{}^{k}v$ and for every variable $u \prec_W v$, it is true that ${}^{m}u/{}^{k}u \in \pi$.

An *equation* is a pair of terms $s$ and $t$, which is written as $s \approx t$.

Let $L$ be a literal of the form $R(t_1, \ldots, t_n)$ $(\neg R(t_1, \ldots, t_n))$ and $M$ be a literal of the form $R(s_1, \ldots, s_n)$ $(\neg R(s_1, \ldots, s_n))$, where $R$ is a predicate symbol. Then $\Sigma(L, M)$ denotes the set of equations $\{t_1 \approx s_1, \ldots, t_n \approx s_n\}$. In this case $L$ and $M$ are said to be *equal modulo* $\Sigma(L, M)$.

*CALCULUS GD.* A basic object of the goal-driven calculus GD under consideration is an e-sequent, which may be considered as a special generalization of the standard notion of a sequent [16] and which is closely connected with the notion of an a-sequent from [11].

An *e-sequent* is an expression of the form $\Gamma \to \Delta, [\Lambda]\ \langle E \rangle$, where $\Gamma$ and $\Delta$ are sequences of formulas, $\Lambda$ is a sequence of literals, and $E$ is a set of equations. Formulas from $\Gamma$ are called *premises*, formulas from $\Delta$ are called *goals*, and literals from $\Lambda$ are called *framed literals*. Here we consider e-sequents containing only one goal. We assume also that all variables occurring in $\Lambda$, $\Delta$, and $E$ are indexed, and all variables occurring in $\Gamma$ are unindexed.

*Axioms* of GD are e-sequents of the form $\Gamma \to \#, [\Lambda]\ \langle E \rangle$.

Below we are interested in e-sequent trees, which are considered to grow "from top to bottom" in accordance with inference rules applications in GD. Note that we identify a tree node with its label.

*INFERENCE RULES.* The calculus GD contains the following *inference rules* applied "from top to bottom":

*Goal-Splitting Rules (GS):*

$(\to \supset)_1$: $$\frac{\Gamma \to F \supset G, [\Lambda]\ \langle E \rangle}{\Gamma \to G, [\Lambda]\ \langle E \rangle}$$ $(\to \supset)_2$: $$\frac{\Gamma \to F \supset G, [\Lambda]\ \langle E \rangle}{\Gamma \to \neg F, [\Lambda]\ \langle E \rangle}$$

$(\to \vee)_1$: $$\frac{\Gamma \to F \vee G, [\Lambda]\ \langle E \rangle}{\Gamma \to G, [\Lambda]\ \langle E \rangle}$$ $(\to \vee)_2$: $$\frac{\Gamma \to F \vee G, [\Lambda]\ \langle E \rangle}{\Gamma \to F, [\Lambda]\ \langle E \rangle}$$

$(\to \wedge)$: $$\frac{\Gamma \to F \wedge G, [\Lambda]\ \langle E \rangle}{\Gamma \to F, [\Lambda]\ \langle E \rangle \quad \Gamma \to G, [\Lambda]\ \langle E \rangle}$$ $(\to \neg)$: $$\frac{\Gamma \to \neg F, [\Lambda]\ \langle E \rangle}{\Gamma \to F^{\neg}, [\Lambda]\ \langle E \rangle}$$

$(\to \forall)$: $$\frac{\Gamma \to \forall\, {}^{k}v\, F, [\Lambda]\ \langle E \rangle}{\Gamma \to F, [\Lambda]\ \langle E \rangle}$$ $(\to \exists)$: $$\frac{\Gamma \to \exists\, {}^{k}v\, F, [\Lambda]\ \langle E \rangle}{\Gamma \to F, [\Lambda]\ \langle E \rangle}$$

| | |
|---|---|
| *Auxiliary-Goal Rule (AG):* | $\dfrac{\Gamma_1, F\lfloor M^+ \rfloor, \Gamma_2 \to L, [\Lambda] \ \langle E \rangle}{\Gamma_1, F, \Gamma_2 \to {}^l(\neg F), [L, \Lambda] \ \langle E \cup \Sigma(L, {}^l M) \rangle}$ |
| *Termination Rule 1 (T1):* | $\dfrac{\Gamma_1, M, \Gamma_2 \to L, [\Lambda] \ \langle E \rangle}{\Gamma_1, M, \Gamma_2 \to \#, [\Lambda] \ \langle E \cup \Sigma(L, {}^l M) \rangle}$ |
| *Termination Rule 2 (T2):* | $\dfrac{\Gamma \to L, [\Lambda_1, M, \Lambda_2] \ \langle E \rangle}{\Gamma \to \#, [\Lambda_1, M, \Lambda_2] \ \langle E \cup \Sigma(\widetilde{L}, M) \rangle}$ |

In AG, the formula $F$ is not a literal. (So, T1 is necessary for the completeness of GD.) In AG and T1, the literals $L$ and ${}^l M$ are equal modulo $\Sigma(L, {}^l M)$, where the index $l$ is a new index. In T2, the literals $\widetilde{L}$ and $M$ are equal modulo $\Sigma(\widetilde{L}, M)$, where $\widetilde{L}$ denotes a literal $L'$, if $L$ is $\neg L'$, and $\widetilde{L}$ denotes $\neg L$ otherwise.

The requirement to GD "to be goal-driven" denotes that whenever the AG rule is applied to some e-sequent with a literal-goal $L$, some positive occurrence $M$ of $L$ (modulo $\Sigma(L, M)$) is fixed in some premise $F$ and all subsequent rules applications are performed w.r.t. the fixed occurrence $M$ until it is possible.

We consider that every usual sequent $P_1, \ldots, P_n \to G$ induces the *initial e-sequent* $P_1, \ldots, P_n, \neg G \to {}^0 G, [\ ] \ \langle\ \rangle$. (It is supposed that the formulas $P_1, \ldots, P_n$, and $G$ do not contain indexed variables).

When a proof of an initial e-sequent $S$ is searching for, an *inference tree* $Tr$ w.r.t. $S$ is constructing. At the beginning of inference search, $Tr$ contains only $S$. The subsequent nodes are generated using rules of GD.

Let $Tr$ be an inference tree and $W$ be a set of formulas occurring in succedents of all e-sequents obtained by $AG$-rule applications in $Tr$. A substitution $\sigma$ is *admissible for* $Tr$ if and only if $\sigma$ is admissible for $W$.

An inference tree $Tr$ is considered to be a *proof tree w.r.t.* $S$ if and only if the following conditions are satisfied: (i) every leaf of $Tr$ is an axiom and (ii) if $E$ is a union of all sets of equations from of leaves of $Tr$, then there are a pasting substitution $\pi$ and a substitution $\sigma$ such that $\sigma$ is a simultaneous unifier of all the equations from $E \circ \pi$ and $\sigma$ is admissible for $Tr \circ \pi$, where $E \circ \pi$ and $Tr \circ \pi$ denote the results of applying $\pi$ to all terms from $E$ and to all expressions from $Tr$, respectively.

The following results take place for GD (cf. [17]).

**Proposition 1.** *Let $P_1, \ldots, P_n$ form a consistent set of formulas. A formula $G$ is the logical consequence of $P_1, \ldots, P_n$ if and only if there exists a proof tree w.r.t. the initial e-sequent $P_1, \ldots, P_n, \neg G \to {}^0 G, [\ ] \langle\ \rangle$ in the calculus GD.*

**Proposition 2.** *A formula $G$ is valid if and only if there exists a proof tree w.r.t. the initial e-sequent $\neg G \to {}^0 G, [\ ] \langle\ \rangle$ in the calculus GD.*

*INFERENCE SEARCH IN FORTHEL1-ENVIRONMENT.* After translating a ForTheL-text to be processed into a correspondent ForTheL1-text $TxT$, an assertion $T$ to be proved is separated from $Txt$ as a ForTheL1-section "theorem", and a remained part (including assumptions, definitions and auxiliary propositions) of $TxT$ is considered as conditions of $T$. Then an initial e-sequent induced by $T$ and $Txt$ is constructed with $T$ and $Txt$ (without $T$) in its succedent and antecedent, respectively. It was noted above that any ForTheL1-sentence can be treated as an analog of a 1st-order logic formula. This enables to construct formula images of such units of a ForTheL1-text as a theorem to be proved, definitions, and auxiliary propositions, and to treat a self-contained ForTheL1-text as a set of 1st-order formulas. Therefore, it is possible to understand unambiguously such terms as "ForTheL1-text consistency", "logical consequence of a theorem from a given ForTheL1-text", and "validity" (of a theorem to be proved) without special defining a semantics of the ForTheL1-language.

**Corollary 1.** *A ForTheL1-theorem $T$ is a logical consequence of a consistent ForTheL1-text $Txt$ (which does not include $T$) if and only if a proof tree w.r.t. an initial sequent induced by $T$ and $Txt$ can be constructed in the calculus GD.*

**Corollary 2.** *A ForTheL1-theorem $T$ is valid if and only if a proof tree w.r.t. an initial sequent induced by $T$ only can be constructed in the calculus GD.*

As a side result, we can note that a rather rich collection of rules in GD enables to construct various proof search strategies, which reflect proofs construction methods from usual mathematical texts and allowed to a user to influence in theorem proving, when an interactive mode of proof search is used. If these strategies (with or without participation of a human) ensure an exhaustive search, then the corollaries 1 and 2 guarantee the soundness and completeness of a strategy under consideration.

## 4 Remarks about Implementation of SAD

By now, some original units of SAD have been constructed and implemented for providing a closed cycle of the following transformations of self-contained mathematical texts: "understanding" of a ForTheL-text, its translation into a ForTheL1-text, and logical inference search using ForTheL1-notations.

These units include a ForTheL-parser along with a translator from ForTheL to ForTheL1 and a prover.

The ForTheL-parser implemented permits to extend a ForTheL-thesaurus, to understand a human-like kind of a ForTheL-text, and to use symbolic abbreviations.

The translator transforms an input ForTheL-text into its ForTheL1-representation.

The prover (based on the above calculus GD) searches for a proof in the frame of a ForTheL1-environment.

Note that recent version of SAD have implemented in the language **C** for the Linux platform. Parsing components have been generated with the instrument tool Bison.

*TRANSLATOR FROM FORTHEL TO FORTHEL1.* The translator solves two main tasks: it performs parsing of an input text along with generating output data, and it maintains the vocabularies of introduced definitions and of functional and predicate symbols.

The translator uses an English version of a ForTheL grammar. Besides, we investigate a possibility to implement Ukrainian and Russian versions of ForTheL. This can give a way for automatic translation of any English ForTheL-text into a correspondent Russian (and/or Ukrainian) ForTheL-text and vise versa.

The grammar of the ForTheL language defines both a structure of a separate ForTheL-sentence and a structure of a whole ForTheL-text.

Since the thesaurus of FoRTheL is not fixed and can be extended, the grammar is based on schemes (or templates) of language constructions. These schemes are classified by their kinds (a notion, a function, a predicate) and by their arity. Thus, a scheme of the construction `function` from $\pi_1$ to $\pi_2$ is "notion_leader prepos ? prepos ?". In all, the ForTheL grammar contains eleven distinct schemes.

In order to recognize words in accordance with their different forms, the translator attempts to build a pattern for every word taken from an appropriate definition. For example, a pattern for the noun `element` will be `elemen???` and a pattern for the verb `cut` will be `cut????`. So far we did not find neither a case of an inconvenient pattern built by this rules nor a case of a collision between two patterns.

*PROVER OF SAD.* The prover of SAD consists of a so-called executing "main sequence" and of several supplementary subsystems responsible for processing of data structures. The "main sequence" includes three stages of actions executed subsequently: processing of input data, generating of a deductive environment, and proving of a theorem under consideration.

At the first stage, the prover builds data structures representing a problem under consideration.

At the second one, it establishes pairs of contrary literals, evaluates corresponding "basic" substitutions, constructs structures for efficient search, and attempts to avoid irrelevant steps from a process of inference search. Note that since the prover is based on the calculus GD not changing a set of initial premises when an inference search is made, it uses a tree of literal goals instead of a tree of e-sequents. This significantly reduces needs for computing resources.

At the third stage, the prover examines a search space using the well known depth-first search with backtracking. In order to increase its efficiency, some strategies are used, and unifiers for "systems" of equations are calculated only when it is "reasonable".

The most important supplementary subsystem is a module that deals with "systems" of equations and acts as a mediator between the prover and some external module for finding equation solutions.

As present, modules for finding of a most simultaneous general unifier and for checking of the admissibility of a generated unifier are implemented. Note also than depending on a problem under consideration, the prover has a possibility to use an another relevant solver of systems of equations. This possibility has arisen due to the absence of preliminary skolemization with preserving of inference search in a signature of an initial theory.

Now we consider different approaches to the equality handling problem keeping all positive features of SAD. In particular, we examine methods based on the elimination of equalities with flattening. Appropriate modules for flattening are implemented.

## 5 Example of Theorem Proving in SAD

Let us give an example of a correct self-contained ForTheL-text, which deals with some elementary notions of the non-standard analysis taken from [7], and demonstrate three above stages of mathematical text processing in SAD. (Note that below `nst[A]` represents a set $A$ in a non-standard universe $^*\mathbf{U}$, and a predicate "`x is close to y`" affirms that a point `x` from $^*\mathbf{U}$ is close to a point `y` from the standard universe.)

```
If set _ A is a subset of the set _ B
          then nst[A] is a subset of nst[B].
Definition 1. A is a subset of B iff all elements
              of the set _ A belong to the set _ B.
Definition 2. M is closed iff for all t if some element of
              nst[M] is close to t then t belongs to the set_M.
```

```
Definition 3. M is compact iff every element of nst[M]
              is close to some element of the set _ M.
Theorem 1. Closed subset of a compact set is compact.
```

(In ForTheL, the underscore symbol declares a variable following it.)

Using this ForTheL-text, the translator generates the following ForTheL1-text. (Note that there are new variables in it of the form $\_i$. These variables are implicitly introduced by the quantifying words "`some`", "`every`", "`all`".)

```
FORALL A (set[A] IMPLIES FORALL B (set[B] IMPLIES (subset[A, B]
         IMPLIES subset[nst[A], nst[B]]))).
DEFINITION 1. FORALL A (set[A] IMPLIES FORALL B (set[B]
              IMPLIES (subset[A, B] IFF FORALL _1 (in[_1, A]
              IMPLIES in[_1, B])))).
DEFINITION 2. FORALL M (set[M] IMPLIES (closed[M] IFF
              FORALL t (EXISTS _2 (in[_2, nst[M]]
              AND close[_2, t]) IMPLIES in[t, M]))).
DEFINITION 3. FORALL M (set[M] IMPLIES (compact[M] IFF
              FORALL _3 (in[_3, nst[M]] IMPLIES EXISTS _4
              (in[_4, M] AND close[_3, _4])))).
THEOREM 1. FORALL _6 ((set[_6] AND compact[_6]) IMPLIES
           FORALL _5 ((subset[_5, _6] AND closed[_5])
           IMPLIES compact[_5])).
```

Having the above ForTheL1-environment, the prover of SAD can prove Theorem 1. Because of the limited size of the paper, here we omit search process, giving only a protocol of generating new goals in a proof tree for Theorem 1. We hope that the protocol gives a possibility to get although an approximate representation about searching for proofs in SAD. Note that we write corresponding equations to the right of literal goals. (In order to simplify the representation, we do not demonstrate goals of the form `set[`$x$`]`.)

```
compact[_5]
   -compact[M]   //  M = _5
   -in[_3, nst[M]]
       in[_1, nst[A]]   //  _1 = _3, A = M
       subset[nst[A], nst[B]]
         -subset[nst[A], nst[B]]   //  A = A, B = B
          subset[A, B]
            -subset[_5, _6]   //  _5 = A, _6 = B
            -in[_1, nst[B]]
                in[_3, nst[M]]   // _3 = _1, M = B
                compact[M]
                  -compact[_6]   //  M = _6
               -close[_3, _4]
                   close[_3, _4]   //  _3 = _3, _4 = _4
                  -compact[M]   //  M = _5
                   in[_4, M]
                     -in[t, M]   //  t = _4, M = M
```

```
close[_2, t]   //  _2 = _3, t = _4
in[_2, nst[M]]   //  _2 = _3, M = M
compact[M]   //
  -compact[_5]   //  M = _5
```

Using the above protocol it is not hard to construct a correspondent proof tree for Theorem 1 in the calculus GD. The correspondent proof tree can transform into a more natural form, representing negative literals as assumptions and positive ones as goals. The text below serves as an appropriate example.

```
To show that  compact[_5]  holds, it is sufficient to prove that
in[_3, nst[_5]] IMPLIES EXISTS _4 (in[_4, _5] AND close[_3, _4]).

Suppose that  in[_3, nst[_5]]  holds.

Then we obtain  subset[nst[_5], B] IMPLIES in[_3, B].

Then we obtain  subset[_5, B] IMPLIES in[_3, nst[B]].

Since  subset[_5, _6]  holds, we obtain  in[_3, nst[_6]].

Then we obtain  compact[_6] IMPLIES EXISTS _4 (in[_4, _6] AND close[_3, _4]).

Since  compact[_6]  holds, we obtain  (in[_4, _6] AND close[_3, _4]).

Then we obtain  close[_3, _4].

Since  EXISTS _4 (in[_4, _5] AND close[_3, _4]) IMPLIES compact[_5]  holds,
we obtain  in[_4, _5] IMPLIES compact[_5].

To show that  in[_4, _5]  holds, it is sufficient to prove that
closed[_5] AND EXISTS _2 (in[_2, nst[_5] AND close[_2, _4]).

Since  closed[_5]  holds, it is sufficient to prove that
EXISTS _2 (in[_2, nst[_5] AND close[_2, _4]).

Since  in[_3, nst[_5]]  holds in the branch, it is sufficient
to prove that  close[_3, _4].

Since  close[_3, _4]  holds, our proof is finished.
```

## 6 Conclusion

Now, there exists a number of projects and systems, which have common feartures with the EA programme or which are close to it in their ideas. Below, we give only a brief remark on such projects and systems.

The project MIZAR [18] has the most closed relation to the EA programme. The objective of that project is the construction of computer systems for mathematical texts processing. A collection of mathematical languages, which are convenient both for mathematicians and computer processing, forms a basis for the MIZAR system. The core of a MIZAR mathematical language is some king of the 1st-order language. The soundness of mathematical texts can be checked by the system.

The main objective of the THEOREMA project [19] is to construct an unified environment (for both logical methods and program mathematical tools) for solving mathematical problems (including numeric computations, algebraic transformations, and theorem proving). Project THEOREMA is designed to offer such an environment for a mathematician that enables to pass through the whole cycle of problem solving.

The final objective of the QED project [20] is the development of an integrate distributed computer environment containing the totality of important and valid mathematical knowledge.

The system OMEGA [21] is developed in order to support theorem proving in mathematics and mathematical education. The system includes a proof planner and a unified collection of tools for the forming subproblems, for searching for proofs for subproblems, and for representing proofs. Both a computer algebra system and well-known general-purpose theorem provers are integrated into OMEGA as external units.

The system ISABELLE [22] is used as a tool for creating an environment for interactive theorem proving. A mathematical knowledge base, which includes a library of concrete mathematics and various packages for advanced mathematical concepts, is used. The system supports a "fashion" of theorem proving usual for mathematicians by reasoning in terms of a given application domain.

A number of the above and some other projects, systems, and groups (for example, the DReaM group, Mechanized Reasoning Group, CAAR group, etc.) are members of the CALCULEMUS project [23] interested in the integration of the deductive and computational power of both deduction systems and computer algebra systems.

Finally, we note that the above features of SAD demonstrate that SAD satisfies well to modern trends in constructing computer mathematical services. In this connection, we note the following.

Since fundamental logical and set-theoretical relations are the basis of the language ForTheL, the language is suitable for the representation of any declarative knowledge formalized in terms of classical logic. A ForTheL-text can be easily created by a human and then it can be passed to a prover or an another mathematical service directly or through the Net.

As for the deductive technique of EA, the implemented prover may be considered as a good base for the further development of the EA-style technique in the direction of the efficiency improvement and of the construction of interactive deductive tools for mathematical text processing.

In the nearest future our end objective is to complete the construction of the following "chain" of transformations: *ForTheL-text with a selected theorem to be proven* $==\gg$ *self-contained ForTheL1-text* $==\gg$ *search for a proof of the theorem* $==\gg$ *a human-like proof of the theorem.*

The above peculiarities of SAD demonstrate a possibility of SAD to combine reasoning capabilities with numeric calculation methods and symbolic computation tools of other systems being external with respect to SAD. This merit of SAD appears owing to the specific technique of mathematical text processing, which takes into account expressive features of the language ForTheL as well as the "fashion" of logical inference search in SAD.

In this connection, the authors hope for that results obtained in the frame of investigations connected with SAD can be helpful in attacking the following problems: distributed automated theorem proving, checking self-contained mathematical texts for soundness, remote training in the mathematical disciplines, extracting knowledge from mathematical papers, and constructing knowledge bases for mathematical theories.

## References


1. Glushkov, V. Some problems of automata theory and artificial intelligence (in Russian). Kibernetika **2** (1970) 3–13
2. Glushkov, V. A System for Automated Deduction (in Russian). In: Avtomatizatsiya obrabotki matematicheskikh tekstov, Institute of Cybernetics, Kiev (1980).
3. Degtyarev, A., Kapitonova, Yu., Letichevsky, A., Lyaletski, A., Morokhovets, M. A brief historical sketch on Kiev school of automated theorem proving. In: Proc. of the 2nd International THEOREMA Workshop. Linz, Austria (June 1998) 151–156.
4. Glushkov, V., Kostyrko, V., et al. On a language for description of formal theories (in Russian). In: Teoreticheskaya kibernetika **3**, Institute of Cybernetics, Kiev (1970).
5. Glushkov, V., Vershinin, K., et al. On a formal language for description of mathematical texts (in Russian). In: Avtomatizatsiya poiska dokazatel'stv teorem v Matematike, Institute of Cybernetics, Kiev (1974) 3–36.
6. Vershinin, K., Paskevich, A. ForTheL — the language of formal theories. IJ Information Theories and Applications **7-3**, 121–127.
7. Davis, M. Applied non-standard analysis (Translated from English). Mir, Moskva (1980).
8. Anufriyev, F., Fediurko, V., et al. On one algorithm of theorem proof search in Group Theory (in Russian). Kibernetika **1** (1966) 23–29.
9. Anufriyev, F. An algorithm of theorem proof search in logical calculi (in Russian). In: Teoriya avtomatov, **1**, Institute of Cybernetics, Kiev (1969).
10. Degtyarev, A., Lyaletski, A. Logical inference in SAD (in Russian). In: Matematicheskiye osnovy sistem iskusstvennogo intellekta. Institute of Cybernetics, Kiev (1981).
11. Degtyarev, A., Lyaletski, A., and Morokhovets, M. Evidence algorithm and sequent logical inference search. In H. Ganzinger, D. McAllester, and A. Voronkov, editors, Logic for Programming and Automated Reasoning (LPAR'99), volume 1705 of Lecture Notes in Computer Science, Springer Verlag (Sept. 1999) 44–61.
12. Degtyarev, A., Lyaletski, A., and Morokhovets M. On the EA-style integrated processing of self-contained mathematical texts. In M. Kerber and M. Kohlhase, editors, Symbolic Calculation and Automated Reasoning: The CALCULEMUS-2000 Symposium, A K Peters (2001) 126–141.
13. Kanger, S. Simplified proof method for elementary logic. In: Comp. Program. and Form. Sys.: Stud. in Logic. North-Holl., Publ. Co. (1963).
14. Robinson, J. A machine-oriented logic based on resolution principle. In: J. of the ACM (1965) 23–41.
15. Lyaletski, A. Gentzen calculi and admissible substitutions. In: Actes preliminaries, du Simposium Franco-Sovetique "Informatika-91". Grenoble, France (1991) 99–111.
16. Gentzen, G. Untersuchungen uber das Logische Schliessen. Math. Zeit. **39** (1934) 176–210.
17. Lyaletski, A., Paskevich, A. Goal-driven inference search in classical propositional logic. In: Proc. of the Inter. Workshop STRATEGIES'2001. Siena, Italy (2001).
18. http://mizar.org/
19. http://www.theorema.org/HomePage.html
20. http://www.mcs.anl.gov/qed/
21. http://www.ags.uni-sb.de/~omega/
22. http://www.cl.cam.ac.uk/Research/HVG/Isabelle/
23. http://www.mathweb.org/calculemus/