Theorem Proving and Proof Verification in the System SAD

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

[1] Faculty of Cybernetics, Kyiv National Taras Shevchenko University, Kyiv, Ukraine
[2] Math-Info Department, Paris 12 University, Creteil, France
lav@unicyb.kiev.ua andrey@raptor.kiev.ua verko@logique.jussieu.fr

Abstract. In this paper, the current state of the System for Automated Deduction, SAD, is described briefly. The system may be considered as the modern vision of the Evidence Algorithm programme advanced by Academician V. Glushkov in early 1970s. V. Glushkov proposed to make investigation simultaneously into formalized languages for presenting mathematical texts in the form most appropriate for a user, formalization and evolutionary development of computer-made proof step, information environment having an influence on the evidence of a proof step, and man-assisted search for a proof. In this connection, SAD supports a number of formal languages for representing and processing mathematical knowledge along with the formal language ForTheL as their top representative, uses a sequent formalism developed for constructing an efficient technique of proof search within the signature of an initial theory, and gives a new way to organize the information environment for sharing mathematical knowledge among various computer services. The system SAD can be used to solve large variety of theorem proving problems including: establishing of the deducibility of sequents in first-order classical logic, theorem proving in ForTheL-environment, verifying correctness of self-contained ForTheL-texts, solving problems taken from the online library TPTP. A number of examples is given for illustrating the mathematical knowledge processing implemented in SAD.

1 Introduction

The **evidential paradigm** [1] of computer-aided "doing" mathematics is considered as the core of the Evidence Algorithm programme (EA) advanced by V. Glushkov [2]. It is intended to provide a mathematician with rich and flexible languages to formalize mathematical knowledge and with methods to prove a given proposition or to verify a given proof, steps of which should be "evident" for a verificator. Thus, the notion of evidence of a proof step reflects the deductive power of proof search methods being applied.

An implementation of the evidential paradigm must satisfy the following requirements: a formal language should be expressive enough to present mathematical knowledge in a natural "human-like" form; the evidence of proof steps should evolutionary develop in accordance with the progress in the field of automated reasoning and other fields of computer mathematics; an information environment should also evolve and influence the evidence of a proof step; interactive technique should permit both a man-assisted searching and cooperation with external computer mathematical services, possibly, via Internet.

The evidential paradigm is oriented to *integration of numerical calculations, analytical transformations, and deduction.*

Numerical paradigm uses the methods and techniques of operation with numbers. For a wide variety of problems of applied mathematics, these methods give an approximate (or a precise) solution by constructing finite sequences of operations over finite sets of numbers. Usually, numerical methods start with the construction of a mathematical continual model. After this, this model is transformed into its discrete representation that permits to construct different algorithms of numerical calculation for a computer [3]. As an example, we can mention famous libraries of numerical algorithms implemented as Fortran batches.

Analytical (symbolic) paradigm allows a computer to make complex symbolic transformations, to find analytical solution of (systems of) equations, to plot functions graphs, to build mathematical models of deterministic processes. Different computer algebra systems can serve as representatives of this paradigm.

The following reasons gave rise to this approach: (i) existence of a big number of labor-consuming tasks requiring routine analytical transformations before numeric calculations, (ii) reduction of efforts and a time spent in the case of solving a wide range of natural-scientific problems, (iii) compactness of representation and visual demonstrability of analytical solutions as compared with their numerical analogs.

Deductive paradigm makes use of a declarative way of representation of mathematical knowledge. This means that all existent knowledge has the form of a formal text, and additional knowledge is obtained (deduced) from it with the help of different rules of reasoning (inference rules).

The whole variety of systems that implement this paradigm can be roughly classified as follows:

- automated inference search systems (provers), such as Otter [4], Vampire [5], SPASS [6];
- interactive tactic-based proof assistants; the most well-known of them are Isabelle [7], PVS [8], and Coq [9];
- mathematical text processors: Mizar [10] and Theorema [11].

The evidential paradigm may be considered as a further development of the third type of the deductive paradigm in the direction of integration of deduction with symbolic calculations that is oriented to a declarative approach using the mentioned-above language ForTheL [12] as a communication tool between different computer mathematical services. This is one of peculiarities that distinguish Evidence Algorithm from Mizar, the system which has received a wide recognition all over the world.

A number of reasons for the appearance of *integration paradigm* have arisen by now (cf. CALCULEMUS Initiative, `http://www.calculemus.net/`). First of all, solving a really difficult problem requires simultaneously computations, analytical transformations, and automated reasoning. Also, in spite of the fact that computer algebra systems are powerful and flexible tools, results of their work often require thorough logical checking because they can be invalid in general. On the other hand, although systems of automated reasoning can guarantee the validity of results, their efficiency remains low as compared to the one of computer algebra systems because they usually lack specific decision procedures and heuristics.

Note that there exist two approaches to integrating computer mathematical services: (i) integration on the stage of design, when the construction of a software system *a priori* presupposes use of both of its modules and different mathematical services as its "own parts" (Mizar, Theorema, and EA) and (ii) integration on the stage exploitations, that means "assembling a new system" from systems acceptable for solving a task under consideration and available at a time when the new system is "constructed" (here a great role is played by computer mathematical services that exchange data through network with the help of special protocols). The projects OMEGA [13], OpenMath [14], and MathWeb [15] can be mentioned as investigations in the last direction; note that they contain languages for specification and exchange of data that can be used not only for solving mathematical problems, but for many other purposes.

2 System for Automated Deduction

By now, the first approximation to the evidential paradigm is implemented in the System for Automated Deduction, SAD [16]. The system can be used online via Internet (http://ea.unicyb.kiev.ua) for the following purposes:

- parsing and translation of ForTheL-texts (some examples can be found on the web-site);
- inference search in first-order classical logic on the basis of an original sequent calculus [17–19]; the user may supply his own problems or refer to the problems in TPTP problem library [20];
- theorem proving in the context of a self-contained ForTheL-text [21];
- verification of self-contained ForTheL-texts [22].

Any session of SAD consists of three main steps. First, SAD translates an input text (written in ForTheL or in a first order language) into its internal representation. In the translated text, the system selects a number of goals (statements to be proved) and, for each goal, determines the set of its logical predecessors (statements to be used in the proof). Finally, the system tries to deduce each goal from its predecessors. For this purpose, SAD may use its native first-order prover, as well as an external program.

There exist six main modules in SAD: [ForTheL], [FOL], [TPTP], [Reason], [Explore], and [Moses]. Below we give a brief description of them.

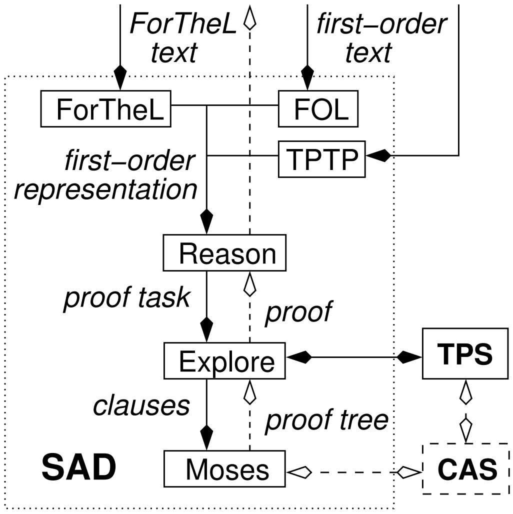

Fig. 1. Architecture of SAD

Modules [ForTheL] *and* [FOL]*.* The modules [ForTheL] and [FOL] perform parsing of ForTheL-texts and first-order texts, respectively. Every of these modules converts an input text to a corresponding internal representation. This conversion preserves the structure of an initial text and translates phrases into an ordered set of first-order formulas. For input texts written in the first-order language, translation is not needed, so [FOL] just eliminates "syntactical sugar".

Module [TPTP]*.* Recently, we have added a new interface module that connects SAD with the wide-known TPTP library that contains several thousands of logical problems for automated proving systems [20]. The system SAD downloads problems and axioms directly from the web-site of TPTP (one can also supply his own problem in the syntax of TPTP), and [TPTP] translates a task into its internal representation.

Module [Reason]*.* This module runs a verification cycle, formulates verification tasks, and tries to solve them using its own reasoning capabilities together with the prover of SAD. In particular, [Reason] processes proofs by case analysis and by induction reasoning, simplifies a goal taking into account assumptions and affirmation written in the text, splits a complex goal to a number of simpler subgoals, and so on.

The role of the module [Reason] will be demonstrated below on an example taken from a real verification session.

Module [Explore]*.* This module serves for purely technical purposes: it acts as a mediator between SAD and external theorem provers that can be used instead of (or together with) the own prover of SAD.

Module [Moses]. This module is the prover of SAD and is intended for inference search. It is implemented on the base of the sequent calculus GD (see below).

The prover looks through the search space using bounded depth first search with iterative deepening and backtracking. In order to increase the efficiency of inference search, [Moses] uses special constraints and folding-up technique [23]. Also, solutions of accumulated systems of equations are found only in reasonable cases. A special heuristics is adopted for definition application.

In order to provide SAD with equality handling capabilities, [Moses] implements a certain variation of Brand's modification method [24].

Note that since the calculus GD does not change initial premises, the prover performs operations with a tree of literal goals instead of a tree of sequents. This drastically reduces needs in computational resources.

Due to the absence of preliminary skolemization, the prover of SAD is capable to use a relevant solver for solving equation systems. The submodule of [Moses] responsible for equation handling acts as a mediator between the prover and external solvers. It checks a substitution found by such a way for admissibility and builds additional equations if necessary. The procedure computing the most general unifier is used as a default equation solver.

3 Linguistic tools of SAD

The language ForTheL (Formal Theory Language) [12] is a language with formally defined syntax and semantics. It is intended for representation of mathematical texts including axioms, definitions, theorems and proofs. ForTheL is designed to be close to the natural language of real-life mathematical publications issued by human mathematicians. There are two reasons to pursue a verbose "natural" style instead of using the unified notation of some traditional logical language.

The first reason is to provide our framework with a user-friendly interface. A text composed with correct English sentences will hopefully more readable and easier to write than a collection of formulas built with quantifiers, parentheses, lambdas, junctors, and so on.

Second, a natural human text usually contains a certain useful information which lies beyond the scope of classical logic. For example, we distinguish definitions from ordinary axioms, theorems from intermediate statements inside a proof, and we tend to use them in different ways when reading a text. In a natural sentence, we meet nouns, which denote classes of entities; adjectives and verbs, which act as attributes and restrict classes; adjectives and verbs, which act as predicates and may relate different entities. However, where human language makes distinctions, the language of mathematical logic unifies: subjects, objects, attributes, predicates all become predicate symbols; axioms, theorems, definitions, ordinary statements all become formulas.

Our intention is to preserve these distinctions in formalization. We investigate how a human mathematician treats definitions as compared with axioms or lemmas, class-nouns as compared with adjectives, proofs by case analysis as compared with proofs following other schemes, and so on. Basing on these observations, we improve reasoning capabilities of a machine, we implement heuristic routines directing proof search or reducing a search space with the help of "non-logical" knowledge extracted from a formal, yet "natural-like" input text.

From the syntactical point of view, any ForTheL-text is a sequence of sections. Each section contains phrases and/or sections of lower levels. Top-level sections (such as theorems, axioms, definitions) play in ForTheL-texts the same role they do in usual mathematical texts. Typical sections of lower level are proofs, proof blocks, and proof cases. The phrases of ForTheL are either assumptions (then they begin with "`let`" or "`assume`") or affirmations.

The grammar of ForTheL-phrases imitates the grammar of English sentences. Phrases are built using nouns, which denote notions (classes) or functions, verbs and adjectives, which denote predicates, and prepositions and conjunctions, which define the logical meaning of a complex sentence. For example, "`Every closed subset of every compact set is compact.`" is a simple ForTheL-affirmation. One can introduce a concise symbolic notation for predicates and functions and write statements in the form of usual first-order formulas.

The logical connectives along with some predicates and notions are predefined in ForTheL. All other syntactical units must be introduced explicitly. We use special *introductors* to declare new lexical primitives. For example, the introductor "`[a divisor/divisors of x]`" extends the current vocabulary with a new noun denoting an unary notion (equivalent forms of the same word may be listed with slashes). Also, one may introduce a new primitive as a synonim: "`[x divides/divide y @ x is a divisor of y]`" introduces a new verb.

An affirmation in a text, e.g. the statement of a theorem, can be provided with a proof section. A ForTheL-proof is a sequence of assumptions, affirmations (which may have their own proofs), and proof cases.

The simple example of a self-contained ForTheL-text is given below:

```
[a set/sets] [an element/elements of x]
[x is in/from y @ x is an element of y]
[a subset/subsets of x] [x is empty]

Definition DefSubset. Let S be a set.
  A subset of S is a set X such that every element of X is in S.

Definition DefEmpty.  Let S be a set.
                      S is empty iff S has no elements.

Axiom ExEmpty.        There exists an empty set.

Theorem.              Let S be a set.
                      S is a subset of every set iff S is empty.
```

This text will be translated into the following first-order representation.

```
Definition DefSubset. forall S (aSet(S) implies
 (forall x_1 (aSubsetOf(x_1,S) iff (aSet(x_1) and forall x_2
```

```
  (aElementOf(x_2,x_1) implies aElementOf(x_2,S)))))).

Definition DefEmpty. forall S (aSet(S) implies
 (isEmpty(S) iff not exists x_1 aElementOf(x_1,S))).

Axiom ExEmpty. exists x_1 (aSet(x_1) and isEmpty(x_1)).

Theorem. assume aSet(S).
 (forall x_2 (aSet(x_2) implies aSubsetOf(S,x_2))) iff isEmpty(S).
```

4 Theorem proving in SAD

Since the internal representation of an input problem is based on a kind of the first-order language, SAD uses its prover for inference search in first-order classical logic. It was mentioned above that a special sequent formalism was developed and implemented in [Moses], the prover of SAD. In order to make this paper self-contained, we give a brief description of a certain modification GD' of the calculus GD introduced in [18] and used in [Moses].

Admissible substitutions. Let Γ be a set of closed formulas such that no two distinct quantifiers in Γ bind the same variable. Let $\mathcal{V}_\Gamma$ stand for the set of *indexed variables* $\{\, {}^k v \mid v \text{ occurs in } \Gamma,\ k \in \mathbb{N} \,\}$. The expression ${}^k F$ will denote the formula F with each variable v replaced with an indexed variable ${}^k v$.

We write $F\lfloor G^+ \rfloor$ ($F\lfloor G^- \rfloor$) to denote a formula F with a positive (respectively, negative) occurrence of a subformula G. A variable ${}^k v \in \mathcal{V}_\Gamma$ is said to be *unknown* in Γ if for some formula $P \in \Gamma$, $P\lfloor (\forall v F)^+ \rfloor$ or $P\lfloor (\exists v F)^- \rfloor$. Correspondingly, x is said to be *fixed* in Γ if for some $P \in \Gamma^*$, $P\lfloor (\forall v F)^- \rfloor$ or $P\lfloor (\exists v F)^+ \rfloor$. Obviously, each variable in $\mathcal{V}_\Gamma$ is either unknown or fixed.

The set Γ induces a partial ordering $\prec_\Gamma$ on $\mathcal{V}_\Gamma$ as follows: ${}^k u \prec_\Gamma {}^m w$ if and only if (a) $k = m$, and (b) a quantifier on w occurs in the scope of a quantifier on u (that is, some formula $F \in \Gamma$ is of the form $(\ldots \mathcal{Q}_1\, u\, (\ldots \mathcal{Q}_2\, w\, (\ldots) \ldots) \ldots)$.

Given a substitution σ, we define another partial ordering $\ll^\sigma_\Gamma$ on $\mathcal{V}_\Gamma$ as follows: ${}^m w \ll^\sigma_\Gamma {}^k u$ if and only if ${}^k u$ is unknown in Γ and ${}^m w$ is fixed in Γ and occurs in ${}^k u\sigma$.

A substitution σ is said to be *admissible* in Γ if and only if (a) σ substitutes for unknown variables only (that is, $x\sigma \neq x$ implies that x is unknown in Γ), and (b) the transitive closure of $\prec_\Gamma \cup \ll^\sigma_\Gamma$ is a partial ordering.

A substitution π is called *pasting* in Γ if and only if (a) π just changes indexes of indexed variables (that is, $x\pi \neq x$ implies that x is of the form ${}^k v$ and $x\pi$ is of the form ${}^m v$), and (b) if ${}^k u \prec_\Gamma {}^k v$ and ${}^k v\pi = {}^m v$ then ${}^k u\pi = {}^m u$.

Inference rules. The main object of the calculus GD' is an *a-sequent*, which is an expression of the form $[\Lambda]\ \Gamma \to G$, where Λ is a list of literals, called *framed literals*, Γ is a list of formulas, called *premises*, and G is a *goal formula*. In our setting, Λ and G contain indexed variables only, and Γ does not contain indexed variables at all.

When a proof of an initial a-sequent is searched in GD', an *inference tree* is constructed. At the beginning of a search process the tree consists of the initial a-sequent. The tree grows from top to bottom, with the subsequent nodes generated in accordance with the inference rules of GD'.

At some moments of inference, branches in the tree may be *terminated*: a leaf node containing an equation of the form $\langle L_1 \approx L_2 \rangle$ (where L_1, L_2 are literals) is added to such a branch and no more expansions are possible on it.

Below, $F^\neg$ denotes the negation of F with the negation sign moved inside F:

$$(\forall x\, P)^\neg = \exists x\, \neg P \qquad (P \vee Q)^\neg = \neg P \wedge \neg Q \qquad (P \supset Q)^\neg = P \wedge \neg Q$$
$$(\exists x\, P)^\neg = \forall x\, \neg P \qquad (P \wedge Q)^\neg = \neg P \vee \neg Q \qquad (\neg P)^\neg = P \qquad A^\neg = \neg A$$

The expression ${}^*F$ stands for ${}^kF$ where k is a fresh index with respect to the whole inference tree. In the following inference rules, A, B are unifiable atoms and L, M are unifiable literals.

Goal-Splitting Rules (GS):

$(\rightarrow \forall)$: $\dfrac{[\Lambda]\ \Gamma \rightarrow \forall x\, F}{[\Lambda]\ \Gamma \rightarrow F}$ $\qquad$ $(\rightarrow \exists)$: $\dfrac{[\Lambda]\ \Gamma \rightarrow \exists x\, F}{[\Lambda]\ \Gamma \rightarrow F}$

$(\rightarrow\supset)_1$: $\dfrac{[\Lambda]\ \Gamma \rightarrow F \supset G}{[\Lambda]\ \Gamma \rightarrow G}$ $\qquad$ $(\rightarrow\supset)_2$: $\dfrac{[\Lambda]\ \Gamma \rightarrow F \supset G}{[\Lambda]\ \Gamma \rightarrow F^\neg}$

$(\rightarrow \vee)_1$: $\dfrac{[\Lambda]\ \Gamma \rightarrow F \vee G}{[\Lambda]\ \Gamma \rightarrow G}$ $\qquad$ $(\rightarrow \vee)_2$: $\dfrac{[\Lambda]\ \Gamma \rightarrow F \vee G}{[\Lambda]\ \Gamma \rightarrow F}$

$(\rightarrow \wedge)$: $\dfrac{[\Lambda]\ \Gamma \rightarrow F \wedge G}{[\Lambda]\ \Gamma \rightarrow F \qquad [\Lambda]\ \Gamma \rightarrow G}$ $\qquad$ $(\rightarrow \neg)$: $\dfrac{[\Lambda]\ \Gamma \rightarrow \neg F}{[\Lambda]\ \Gamma \rightarrow F^\neg}$

Auxiliary-Goal Rule (AG): $\qquad$ $\dfrac{[\Lambda]\ \Gamma, F\lfloor B^+ \rfloor, \Delta \rightarrow A}{[\Lambda, \neg A]\ \Gamma, F, \Delta \rightarrow {}^*F^\neg}$

$$\dfrac{[\Lambda]\ \Gamma, F\lfloor B^- \rfloor, \Delta \rightarrow \neg A}{[\Lambda, A]\ \Gamma, F, \Delta \rightarrow {}^*F^\neg}$$

Termination-by-Framed-Literal Rule (TF): $\qquad$ $\dfrac{[\Lambda_1, M, \Lambda_2]\ \Gamma \rightarrow L}{\langle M \approx L \rangle}$

Termination-by-Premise Rule (TP): $\qquad$ $\dfrac{[\Lambda]\ \Gamma, M, \Delta \rightarrow L}{\langle {}^*M \approx L \rangle}$

The following *goal-drivenness constraint* is applied to the inferences in GD'. Whenever an AG-rule is applied and the atom A (possibly, negated) goes to the end of the list of framed literals, the corresponding occurrence of B in ${}^*F^{\neg}$ must be fixed and remembered. We require that the subsequent applications of goal-splitting rules never remove that occurrence of B from the proof. In other words, that occurrence of B must form a literal goal after a number of splittings. Then, we require the branch containing this goal to be terminated by an application of TF-rule, with the equation $\langle A \approx B \rangle$ in the leaf.

Proof trees. Let us consider an inference tree T where every branch is terminated. Let Γ denote the set of premises in the initial a-sequent. Let E denote the overall system of substitutional equations in the leaves of the tree.

The inference tree T is considered to be a *proof tree* whenever there exist substitutions σ and π such that: (a) σ is admissible in Γ, (b) π is pasting in Γ, and (c) $\sigma \circ \pi$ is a solution for E (that is, $E\pi\sigma$ is a set of identities).

The initial a-sequent of a proof tree is called *deducible* in GD'.

Theorem 1 (Soundness and Completeness of GD'). *Let Γ be a consistent set of formulas and G be a formula. The sequent $\Gamma \to G$ is deducible in Gentzen's calculus* ***LK*** *[25] if and only if the a-sequent $[\ ]\,\Gamma, \neg G \to {}^0G$ is deducible in the calculus GD'.*

Note that equation solving is separated from application of inference rules. Substitutional equations may be accumulated and be sent to a solver in arbitrary moment of search. This property of GD' allows to organize a flexible proof search procedure. We underline that the calculus GD' does not rely upon preliminary skolemization, so that equations are formulated in the initial signature and a theory-specific equation solver may be used.

Let us demonstrate the deductive technique of SAD on a simple propositional problem. Assume that we want to prove the sequent $A \lor (B \lor A) \to A \lor B$. Let $\Gamma = \{A \lor (B \lor A), \neg(A \lor B)\}$. A GD'-inference used to prove the a-sequent $[\ ]\,\Gamma \to A \lor B$ is shown below:

$$\dfrac{\dfrac{\dfrac{[\ ]\,\Gamma \to A \lor B}{[\ ]\,\Gamma \to A}\,(\to \lor)_1}{[\neg A]\,\Gamma \to \neg A \land \neg(B \lor A)}\,(AG)}{\dfrac{[\neg A]\,\Gamma \to \neg A}{\langle \neg A \approx \neg A \rangle}\,(TF) \qquad \dfrac{[\neg A]\,\Gamma \to \neg(B \lor A)}{\dfrac{[\neg A]\,\Gamma \to \neg B \land \neg A}{\dfrac{\dfrac{[\neg A]\,\Gamma \to \neg B}{\dfrac{[\neg A, B]\,\Gamma \to A \lor B}{\dfrac{[\neg A, B]\,\Gamma \to B}{\langle B \approx B \rangle}\,(TF)}\,(\to \lor)}\,(AG) \qquad \dfrac{[\neg A]\,\Gamma \to \neg A}{\langle \neg A \approx \neg A \rangle}\,(TF)}{}\,(\to \land)}}\,(\to \neg)}\,(\to \land)$$

Note that in any GD'-inference the premises are the same in each a-sequent. Moreover, the list of framed literals in a given node is exactly the list of the complements of literal goals above that node. So, we can present the proof tree given above in a quite abbreviated form, as the tree of literal goals. In the following inference, applications of the auxiliary-goal rule are joined with the subsequent applications of goal-splitting rules in a single inference step, and termination by a framed literal becomes termination by a contradiction in a branch:

$$\cfrac{\cfrac{[\ ]\ A \vee (B \vee A), \neg(A \vee B) \to A \vee B}{A}\ (GS)}{\cfrac{\neg A}{\langle \neg A \approx \neg A \rangle}\ (TF) \quad \cfrac{\neg B}{\cfrac{B}{\langle B \approx B \rangle}\ (TF)}\ (AG+GS) \quad \cfrac{\neg A}{\langle \neg A \approx \neg A \rangle}\ (TF)}\ (AG+GS)$$

Fetching problems from the TPTP library. As we said above, the system SAD may refer to the TPTP problem library and downloads problems and axiom sets directly from the web-site of TPTP, so that it is sufficient to give the identifier of the problem to SAD in a corresponding mode. For example, given `SET001-1.p`, SAD downloads first the text of the problem (comments are erased):

```
include('Axioms/SET001-0.ax').

input_clause(b_equals_bb,hypothesis,
    [++equal_sets(b,bb)]).

input_clause(element_of_b,hypothesis,
    [++member(element_of_b,b)]).

input_clause(prove_element_of_bb,conjecture,
    [--member(element_of_b,bb)]).
```

parses it, and then downloads the needed axiom set. The resulting text is transformed into an appropriate a-sequent and the inference search procedure is started. The following information will be displayed when the session finishes:

```
[Reason] inference search successful
[Main] session finished in 00:00.01
[Main] 00:00.00 in [TPTP] - 00:00.00 in [Reason] - 00:00.01 in [Moses]
```

Theorem proving in ForTheL-environment. Let us consider again the ForTheL-text about empty sets given above. The following text demonstrates the output of SAD when proving the last theorem. The proposition is “evident” for SAD and the proof is found in less than a second.

The proof of the proposition has the form of a tree of literals, in a similar fashion to the proof tree given above. This inference can be transformed in the usual Gentzen-type sequent proof or in the Gentzen’s natural inference.

```
Proof tree --- nodes: 17, depth: 5.
Proved in 10 msec (steps: 208, nodes: 614, depth bound: 4).
```

```
Root:
   isEmpty(S#9[1])
      -isEmpty(S#9[1])
      -aElementOf(x_1#7[2](S#9[1]),S#9[1])
         aElementOf(x_1#7[2](S#9[1]),S#9[1])
         aSubsetOf(S#9[1],x_1#8[5])
            -aSubsetOf(S#9[4],x_1#8[5])
            -isEmpty(S#9[4])
            aSet(x_1#8[5])
               -aSet(x_1#8[5])
         -aElementOf(x_1#7[2](S#9[1]),x_1#8[5])
            aElementOf(x_1#7[2](S#9[1]),x_1#8[5])
            isEmpty(x_1#8[5])
               -isEmpty(x_1#8[7])
            aSet(x_1#8[5])
         aSet(x_1#8[5])
      aSet(S#9[1])
         -aSet(S#9[8])
   -aSet(x_2#a[1])
      aSet(x_2#a[1])
      -aSubsetOf(S#9[10],x_2#a[1])
         aSubsetOf(S#9[10],x_2#a[10])
         isEmpty(S#9[10])
      -aElementOf(x_2#4[9](x_2#a[1],S#9[10]),S#9[10])
         aElementOf(x_2#4[9](x_2#a[1],S#9[10]),S#9[10])
         isEmpty(S#9[10])
         aSet(S#9[10])
      aSet(S#9[10])
```

5 Text verification in SAD

This section is devoted to the verification procedure of SAD and to the peculiarities of formalization style. We consider a real mathematical problem: Ramsey's theorem (both finite and infinite versions) as it is presented in the beginning of Graham's introductory book [26]:

Infinite Ramsey's Theorem. *For all $k, r \in \omega$ and any r-coloring $\xi : \left[\begin{smallmatrix}\omega\\ k\end{smallmatrix}\right] \to [r]$ of the k-element subsets of ω, there is always an infinite subset $S \subseteq \omega$ with all its k-element subsets having the same color.*

A set hypergraph *$H = H(V, E)$ denotes a set V together with a family E of subsets of V in presupposition that every of them contains at least two elements. The* chromatic number *of H, denoted by $\chi(H)$, is defined to be the least integer r such that there is an r-coloring of V so that no edge in E is monochromatic.*

Compactness Theorem. *If $\chi(H) > t$ and all edges of H are finite then there is a finite subhypergraph G of H with $\chi(G) > t$.*

Finite Ramsey's Theorem. *For all $k, l, r \in \omega$ there exists $N \in \omega$ such that if $n \geqslant N$ and $\xi : \left[\begin{smallmatrix} n \\ k \end{smallmatrix}\right] \to [r]$ is any r-coloring of the k-subsets of $[n]$ then some l-subset of $[n]$ has all its k-subsets with the same color.*

These propositions and their proofs were formalized and automatically verified in SAD. As an experiment of proving systems cooperation, the automated theorem prover SPASS [6] was used as the external prover to support verification and showed excellent results. The whole ForTheL-text consists of 490 lines of which 200 are used for preliminary facts: common elements of set and number theory, definitions and properties of functions and predicates used in the text. The rest 290 lines contain the formalization of the above text together with three proofs. Note that the above text together with two proofs (the proof of Finite Ramsey's Theorem is not given) takes approximately 130 lines in Graham's book. So, we can consider ForTheL (and the whole system SAD) as a rather economical tool for formalization of texts.

The given-above definitions and propositions are rewritten in ForTheL as follows (we replace some ASCII-notation with more readable notation of TEX).

Theorem RamseyInf. Let T be a finite set.
  For every ($k \in \omega$) and every countable ($S \subseteq \omega$) and every ($c : [S/k] \to T$)
    there exists an element u of T and a countable $X \subseteq S$ such that
      for every ($Q \in [X/k]$) $c(Q) = u$.
Definition DefChromC.
  Let H be a hypergraph and $L : \mathrm{Ver}\ H \to T$ for some finite set T.
  L is chromatic for H iff H has no edge that is monochromatic in H wrt L.
Definition DefChromT.
  Let H be a hypergraph and T be a finite set.
  T is chromatic for H iff there exists $L : \mathrm{Ver}\ H \to T$ chromatic for H.
Theorem Compactness.
  Assume H is a hypergraph and $\mathrm{Ver}\ H = \omega$.
  Assume every edge of H is finite.
  Let T be a finite set chromatic for every finite subgraph of H.
  Then T is chromatic for H.
Theorem RamseyFin. Let T be a finite set. Let k, l be numbers.
  Then there exists a number n such that for every ($c : [[n]/k] \to T$)
    there exists an element u of T and an element X of $[[n]/l]$
      such that for every ($Q \in [X/k]$) $c(Q) = u$.

We don't give here the whole ForTheL-text. Instead, we will consider a fragment from the proof of the Infinite Ramsey's Theorem that illustrates well the peculiarities of text verification in SAD.

The Infinite Ramsey's Theorem is proved by induction on k. In the proof of the induction step, we build a number of objects (sequences, functions, and sets) and prove their properties. The most important construction in our proof is $\{N_i\}_\omega$, a recursively defined sequence of subsets of ω. This sequence is decreasing, that is $N_{i+1} \subseteq \mathrm{cdr}\ N_i = N_i \setminus \{\min N_i\}$. We want to prove a simple yet useful consequence of this fact:

For every $(i, j \in \omega)$ if $j \leqslant i$ then $N(i) \subseteq N(j)$.
proof by induction.
Let i, j be numbers so that $j \leqslant i$.
Let I be a number so that $i = \mathrm{succ}\ I$.
Case $i \leqslant j$. Obvious.
Case $j \leqslant I$.
Then $N(I) \subseteq N(j)$ (by IH).
cdr $(N(I)) \subseteq N(I)$ (by DefDiff).
$N(i) \subseteq N(j)$ (by SubTrans).
end.
end.

The main affirmation is proved by natural induction. The systems assumes by default that the induction is held by the topmost universally quantified variable in the goal, that is, by i. Also, you can mention the induction term explicitly ("`proof by induction on i+j.`").

The verificator ([Reason]) checks that the first assumption in the proof section introduces the required variables and corresponds to the conditions in the goal. Then the appropriate induction hypothesis IH(i) is formulated:

$$\forall x\,, y \in \omega\ (x < i \supset (y \leqslant x \supset N(x) \subseteq N(y)))$$

and is added to the logical context of the rest of the proof.

In our proof, the base case ($i = 0$) is obvious, since $N(0) = \omega$ by definition. Therefore, we simply omit it and consider the induction step. We assume existence of the predecessor of i and prove the current goal $N(i) \subseteq N(j)$ by case analysis. Note that the goal was simplified in accordance with the first assumption.

When a sequence of proof cases is considered, the system checks that the case analysis is complete, i.e. the corresponding disjunction is valid. In our proof, the disjunction is succ $x \leqslant y \vee y \leqslant x$ and it is easily verified. Then the reasoner tries to prove the current goal for each case respectively. Note the reference to the induction hypothesis made in the second case section.

6 Conclusion

This paper is devoted to the description of mathematical texts processing in the framework of evidential paradigm having now the form of the implemented system SAD that is intended for different complex ways of mathematical texts processing, mainly, automated theorem proving and checking formal mathematical texts. In this connection, the system SAD exploits simultaneously a formal language for presenting mathematical texts (including theorems, their proofs, and necessary environment) in the form most appropriate for a user, as well as a formal notion of computer-made proof step based on sequent formalism of first-order logic. Mathematical text processing in SAD consists of three main steps: firstly, writing a text under consideration in a special formal language, secondly, translation of the written text into a text in the form of a set of special first-order formulas, and thirdly, proving/checking the special text with the help of sequent-type deductive tools.

Finally note that systems that constructed on the basis of the evidential paradigm (in particular, SAD) can be useful in many industrial fields such as computer-aided processing of computer knowledge, automated reasoning, checking soft and hardware, logical inference search taking into account constraints, distributed learning, e-learning, data mining from mathematical papers, construction knowledge bases for formal theories.

References

1. Lyaletski, A., Morokhovets, M.: Evidential paradigm: a current state. In: Program of International Conference “Mathematical Challenges of the 21st Century”, University of California, Los Angeles, USA (2000) 48
2. Glushkov, V.M.: Some problems of automata theory and artificial intelligence (in Russian). Kibernetika **2** (1970) 3–13
3. Bakhvalov, N.I., Zhidkov, N.P., Kobelkon, G.M.: Computational Methods (in Russian). BINOM Publisher (2003)
4. McCune, W.: Otter 3.0 reference manual and guide. Tech. Report ANL-94/6, Argonne National Laboratory, Argonne, USA (1994)
5. Riazanov, A., Voronkov, A.: The design and implementation of VAMPIRE. AI Communications **15** (2002) 91–110
6. Weidenbach, C.: System description: Spass version 1.0.0. In Ganzinger, H., ed.: Automated Deduction — CADE-16, Proc. 16th International Conference on Automated Deduction. Volume 1632 of Lecture Notes in Computer Science., Springer (1999) 378–382
7. Nipkow, T., Paulson, L.C., Wenzel, M.: Isabelle/HOL: A Proof Assistant for Higher-Order Logic. Volume 2283 of Lecture Notes in Computer Science. Springer (2002)
8. Owre, S., Rushby, J.M., Shankar, N.: PVS: A prototype verification system. In Kapur, D., ed.: Automated Deduction — CADE-11, Proc. 11th International Conference on Automated Deduction. Volume 607 of Lecture Notes in Computer Science., Springer (1992) 748–752
9. Barras, B., Boutin, S., Cornes, C., Courant, J., Filliatre, J., Giménez, E., Herbelin, H., Huet, G., Muñoz, C., Murthy, C., Parent, C., Paulin, C., Saïbi, A., Werner, B.: The Coq proof assistant reference manual — version v6.1. Technical Report 0203, INRIA (1997)
10. Trybulec, A., Blair, H.: Computer assisted reasoning with Mizar. In: Proc. 9th International Joint Conference on Artificial Intelligence. (1985) 26–28
11. Buchberger, B., Jebelean, T., Kriftner, F., Marin, M., Tomuta, E., Vasaru, D.: A survey of the Theorema project. In Küchlin, W., ed.: ISSAC’97 — Proc. International Symposium on Symbolic and Algebraic Computation, Maui, Hawaii, USA, ACM Press (1997) 384–391
12. Vershinin, K., Paskevich, A.: ForTheL — the language of formal theories. International Journal of Information Theories and Applications **7-3** (2000) 120–126

13. Benzmüller, C., Cheikhrouhou, L., Fehrer, D., Fiedler, A., Huang, X., Kerber, M., Kohlhase, M., Konrad, K., Meier, A., Melis, E., Schaarschmidt, W., Siekmann, J.H., Sorge, V.: Omega: Towards a mathematical assistant. In McCune, W., ed.: Automated Deduction — CADE-14, Proc. 14th International Conference on Automated Deduction. Volume 1249 of Lecture Notes in Computer Science., Springer (1997) 252–255
14. Caprotti, O., Cohen, A.M.: Integrating computational and deduction systems using OpenMath. In Armando, A., Jebelean, T., eds.: Electronic Notes in Theoretical Computer Science. Volume 23., Elsevier (2001)
15. Zimmer, J., Kohlhase, M.: System description: The MathWeb Software Bus for distributed mathematical reasoning. In Voronkov, A., ed.: Automated Deduction — CADE-18, Proc. 18th International Conference on Automated Deduction. Volume 2392 of Lecture Notes in Computer Science., Springer (2002) 139–143
16. Verchinine, K., Degtyarev, A., Lyaleyski, A., Paskevich, A.: SAD, a System for Automated Deduction: a current state. In Kamareddine, F., ed.: Proc. Workshop on 35 Years of AUTOMATH, Heriot-Watt University, Edinburgh, Scotland (2002)
17. Degtyarev, A., Lyaletski, A., Morokhovets, M.: Evidence Algorithm and sequent logical inference search. In Ganzinger, H., McAllester, D.A., Voronkov, A., eds.: Logic Programming and Automated Reasoning, Proc. 6th International Conference LPAR'99. Volume 1705 of Lecture Notes in Computer Science., Springer (1999) 44–61
18. Lyaletski, A., Verchinine, K., Degtyarev, A., Paskevich, A.: System for Automated Deduction (SAD): Linguistic and deductive peculiarities. In Klopotek, M.A., Wierzchon, S.T., Michalewicz, M., eds.: Intelligent Information Systems 2002, (Proc. IIS'2002 Symposium, Sopot, Poland, June 3–6, 2002). Advances in Soft Computing, Physica-Verlag (2002) 413–422
19. Aselderov, Z., Verchinine, K., Degtyarev, A., A.Lyaletski, Paskevich, A., Pavlov, A.: Linguistic tools and deductive technique of the System for Automated Deduction. In: Proc. 3rd International Workshop on the Implementation of Logics, Tbilisi, Georgia (2002) 21–24
20. Sutcliffe, G., Suttner, C.B., Yemenis, T.: The TPTP problem library. In Bundy, A., ed.: Automated Deduction — CADE-12, Proc. 12th International Conference on Automated Deduction. Volume 814 of Lecture Notes in Computer Science., Springer (1994) 252–266 Also see `http://tptp.org`.
21. Degtyarev, A., Lyaletski, A., Morokhovets, M.: On the EA-style integrated processing of self-contained mathematical texts. In Kerber, M., Kohlhase, M., eds.: Symbolic Computation and Automated Reasoning: The CALCULEMUS-2000 Symposium. A.K. Peters, Ltd., USA (2001) 126–141
22. Lyaletski, A., Verchinine, K., Paskevich, A.: On verification tools implemented in the System for Automated Deduction. In: Proc. 2nd CoLogNet Workshop on Implementation Technology for Computational Logic Systems (ITCLS'2003), Pisa, Italy (2003) 3–14
23. Letz, R., Stenz, G.: Model elimination and connection tableau procedures. In Robinson, A., Voronkov, A., eds.: Handbook for Automated Reasoning. Volume II. Elsevier Science (2001) 2017–2116
24. Brand, D.: Proving theorems with the modification method. SIAM Journal of Computing **4** (1975) 412–430
25. Gentzen, G.: Untersuchungen uber das Logische Schliessen. Math. Zeit. **39** (1934) 176–210
26. Graham, R.L.: Rudiments of Ramsey Theory. AMS (1981)