Linguistic Tools and Deductive Technique of the System for Automated Deduction

Zainutdin Aselderov[1], Konstantin Verchinine[2], Anatoli Degtyarev[3], Alexander Lyaletski[4], Andrey Paskevich[4], and Alexandre Pavlov[1]

[1] Department of Informatics and Computing Technique,
National Technical University of Ukraine "Kiev Polytechnic Institute"
Kyiv, Ukraine (e-mail: tanya@gef.kiev.ua)
[2] Math-Info Department, Paris 12 University,
Creteil, France (e-mail: verko@logique.jussieu.fr)
[3] Department of Computer Science, King's College,
Strand, London, United Kingdom (e-mail: anatoli@dcs.kcl.ac.uk)
[4] Faculty of Cybernetics, Kyiv National Taras Shevchenko University,
Kyiv, Ukraine (e-mail: lav@tc.unicyb.kiev.ua, andrey@raptor.kiev.ua)

Abstract. This paper is devoted to a brief description of some peculiarities and of the first software implementation of the System for Automated Deduction, SAD. Architecture of SAD corresponds well to a modern vision of the Evidence Algorithm that was conceived by V. Glushkov as a programme for constructing open systems for automated theorem-proving that are intended for computer-aided "doing" mathematics: i.e. for extracting and accumulating mathematical computer knowledge and for using it in a regular and efficient manner to prove a given statement that is always considered as a part of a self-contained mathematical text. In addition, the main principles of SAD reflect the current understanding of the problem of man-assisted processing of mathematical computer knowledge.

1 Introduction

By the end of 1960s Academician V.Glushkov advanced a programme on investigating automated theorem-proving, which was latter called the Evidence Algorithm, EA [1]. V.Glushkov proposed to make investigation simultaneously into formalized languages for presenting mathematical texts in the form most appropriate for a user, formalization and evolutional development of computer-made proof step, EA information environment having an influence on a current evidence of computer-made proof step, and interactive man-assistant search of proof.

Since then, a lot of investigations were made in all the above spheres[5].

Russian and English versions of the formalized mathematical languages were developed. Their syntactical analyzers were designed and implemented.

A sequential formalism was developed for construction of an efficient technique of proof search in an initial theory (without preliminary skolemization). A special approach was offered for applying definitions and auxiliary propositions that takes into account a neighbourhood of a statement under consideration. Basing on this formalism, a first-order prover was implemented. As a result, the System for Automated Deduction, SAD, appeared, and its first software version is available at present[6].

The system SAD has a number of peculiarities distinguishing it from other similar systems.

Firstly, a mathematical text under consideration is formalized with the help of a specific formal language that is close to a natural language of mathematical publications. After that, a formalized text can be automatically translated into a certain first-order language by such a way that preserves a structure of the initial text.

[5] This research is supported in part by INTAS 2000-447.
[6] See "http://ea.unicyb.kiev.ua".

Secondly, the suggested sequent formalism gives a possibility to construct different computer-oriented first-order sequent calculi for logical inference search and to formalize a "natural reasoning style". The formalism also permits to separate equation handling from deduction. This property gives an opportunity to integrate logical reasoning with symbolic calculation.

And thirdly, proof search is done in the framework of a self-contained mathematical text, containing knowledge accumulated in an information environment.

2 Languages of SAD

First of all note that any input language of the system SAD must satisfy the following requirements. Such a language should have formal syntax and semantics. It should permit to write theory axioms, definitions, theorems, and proofs by the "usual" way in order to obtain self-contained texts. Accordingly, its thesaurus should be separated from its grammar in order to be extensible. Besides, the language should be close to natural one used in mathematical papers. This feature provides with a user-friendly interface for creating and processing mathematical texts in an interactive mode.

The last representative of such a kind of a "family" of languages constructed for SAD is ForTheL (FORmal THEory Language) [2].

From the syntactical point of view, any ForTheL-text is an ensemble of sections. Each section can contain phrases and sections of the lower levels. Some of the sections, for example, theorems, proofs, and definitions play a special role in the language.

The language ForTheL contains the first-order language as its proper subset. This feature permits to write down any formulas of first-order logic as ForTheL-phrases. In addition, ForTheL includes so-called concepts and the verbal scripting of language units.

A concept can be considered as analogue of a first-order term accepting a (parameterized) set of objects as its value.

The verbal scripting allows constructing ForTheL-phrases by the same way that is used in any natural language, i.e. by using subjects, predicates, objects and attributes.

The logical connectives along with some predicates and concepts are syntactical ForTheL-units predefined. All other syntactical units must be introduced by using definitions.

The simple example of self-contained ForTheL-text related to Set Theory is:

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

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

There exists an empty set.

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

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. After that, a ForTheL-text must be automatically transformed into a so-called ForTheL1-text. A ForTheL1-text consists of sentences that, on the one hand, are analogs of first-order formulas, and, on the other hand, preserve the signature of an original ForTheL-text, its syntax and structure, i.e. partition into sections, such as definitions, auxiliary propositions, and a theorem to be proved.

After finishing the transformation of a ForTheL-text into a ForTheL1-text, the construction of an information (ForTheL1-)environment for deductive procedures can be made. That is why SAD translates a given ForTheL-text into an appropriate ForTheL1-text.

3 Deduction in SAD

As it was mentioned above, the first-order sequent formalism was developed for constructing methods of efficient proof search within a signature of an initial theory (see, for example, [3, 4]. This choice of the formalism can be explained by the fact that sequent-based methods are more close to the "natural" way of reasoning than resolution-type ones. In particular, there is no necessity for translating a problem under consideration to a set of clauses containing, perhaps, Skolem functions symbols. This property becomes important in the case if man-machine interaction is assumed.

Now deduction in SAD is based on the first-order sequent calculus GD from [4], which provides with the following possibilities:

– to preserve a structure of an original task to be solved;

– to reduce a proposition to be proven to a number of new auxiliary propositions;

– to separate equality handling from deduction in order to postpone solving "equations" up to a moment been determined by a task under consideration or by a user;

– to construct a special quantifier handling technique (without any skolemization) that permits to search for an inference within a signature of an initial theory and to achieve the search efficiency comparable with the efficiency of methods requiring skolemization;

– to accumulate "equations" to be solved without specifying terms that should be substituted for variables;

– to select necessary equation-solving technique (e.g., to apply the usual unification, AC-unification, or E-unification), to build-in specific equality handling rules (for example, the paramodulation), and/or to try to use various solvers and tools of computer algebra systems;

– to apply human-like methods of theorem proving, such as, for example, definitions and auxiliary propositions applications (cf. [5]);

– to organize a flexible interactive search mode.

Some of these possibilities are inherent to the implemented version of SAD.

4 Implementation of SAD

The system SAD is oriented to performing the following chain of transformations:

– creation of a ForTheL-text containing a proposition to be proven (a user put down it on the base of an oridinal mathematical text);

– translation of a ForTheL-text into a self-contained ForTheL1-text;

– construction of a proof environment for deduction;

– search for a proof;

– transformation of a proof obtained to the form most appropriate for a user.

Now SAD contains two pripciple modules: a translator from ForTheL to ForTheL1 and a first-order prover based on the sequent calculus GD. (In this connection, we note that the current version of SAD was implemented in the C language for the Linux platform. Parsing components are generated with Bison.)

The translator solves two main tasks: it performs parsing of an input ForTheL-text along with generating output data and it maintains a vocabulary of introduced notions, functions, and predicates.

The translator is now intended for an English version of a ForTheL grammar only. But a possibility to implement Ukrainian and Russian versions of ForTheL is under investigation. It will give a way for automatic translation of ForTheL-texts in the different languages.

The prover of SAD consists of a main subroutine called "main sequence of execution" and of several supplementary subroutines responsible for processing of data structures. The "main sequence" includes three action steps made subsequently: processing of input data, generation of a deductive environment, and search for a proof of a theorem under consideration.

At the first step, the prover generates proof environment and construct all data structures, representing a ForTheL1-text under consideration.

At the second step, it establishes pairs of "interacting" literals, evaluates corresponding "basic" substitutions, selects ways of efficient proof search, and attempts to predict irrelevant steps in 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 sequents. The last significantly reduces requirements to computational resources.

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

Important role plays a module that deals with systems of equations and acts as a mediator between the prover and some external modules for finding equations solutions. This module checks a substitution, calculated by a solver, on admissibility and forms additional equations if necessary.

A module for finding the most general unifier is used as a default equation solver. Note that depending on a problem under consideration, the prover has a possibility to use any relevant solver of systems of equations. This possibility raises due to absence of skolemization.

In order to provide with equality handling in SAD when first-order classical logic with equality is considered, a certain version of Brand's modification method [6] have been implemented by now.

5 Conclusion

The authors express the hope that results obtained in the frame of investigations in SAD can be helpful in attacking the following problems: distributed automated theorem proving, verification of mathematical papers for soundness, extraction of knowledge from mathematical papers, remote training in mathematical disciplines via Internet, constructing knowledge bases for mathematical theories, and integration of computer algebra with deduction. Also, SAD can be adapted to verification of formal specifications both software and hardware, to performance of model checking, to solving logical tasks of decision theory, and so on.

References

1. V. Glushkov. Some Problems of Automata Theory and Artificial Intelligence (in Russian). In: Kibernetika, No 2, 1970, 3– 13.
2. K. Vershinin, A. Paskevich. ForTheL – the Language of Formal Theories. In: IJ Information Theories and Applications, v. 7-3, 2000, 121–127.
3. A. Degtyarev, A. Lyaletski, M. Morokhovets. Evidence Algorithm and Sequent Logical Inference Search. In: Lecture Notes in Artificial Intelligence, v. 1705, 1999, 44–61.
4. A. Lyaletski, K. Verchinine, A. Degtyarev, A.Paskevich. SAD, a System for Automated Deduction: a Current State. In: Proceedings of the Workshop on 35 Years of Automating Mathematics, Heriot- Watt University, Edinburgh, Scotland, 10-13 April, 2002.
5. A. Degtyarev, A. Lyaletski, and M. Morokhovets. On the EA-Style Integrated Processing of Self-Contained Mathematical Texts. In: Symbolic Computation and Automated Reasoning (the book devoted to the CALCULEMUS-2000 Symposium: edited by M. Kerber and M. Kohlhase), A.K. Peters, Ltd, USA, 2001, 126–141.
6. D. Brand. Proving Theorems with the Modification Method. In: SIAM Journal of Computing, v. 4, 1975, 412–430.