System for Automated Deduction (SAD): a tool for proof verification

Konstantin Verchinine[1], Alexander Lyaletski[2], and Andrei Paskevich[3]

[1] Université Paris 12, IUT Sénart/Fontainebleau,
77300 Fontainebleau, France,
verko@capet.iut-fbleau.fr
[2] Kyiv National Taras Shevchenko University, Faculty of Cybernetics,
03680 Kyiv, Ukraine,
lav@unicyb.kiev.ua
[3] Université Paris 12, Laboratoire d'Algorithmique, Complexité et Logique,
94010 Créteil, France,
andrei@capet.iut-fbleau.fr

Abstract. In this paper, a proof assistant, called SAD, is presented. SAD deals with mathematical texts that are formalized in the ForTheL language (brief description of which is also given) and checks their correctness. We give a short description of SAD and a series of examples that show what can be done with it. Note that abstract notion of correctness on which the implementation is based, can be formalized with the help of a calculus (not presented here).

1 Introduction

The idea to use a formal language along with formal symbolic manipulations to solve complex "common" problems, has a long history. We would remind G.W. Leibniz's writings (1685) and the pioneer paper of Hao Wang [1]. It is worth noting how ambitious was the title of Wang's article! Numerous attempts to "mechanize" mathematics led to less ambitious and more realistic idea of "computer aided" mathematics as well as to the notion of "proof assistant" — a piece of software that is able to do complex deductions for you.

Mathematical text SAD deals with, is a complex object that contains axioms, definitions, theorems, and proofs of various kinds (by contradiction, by induction, by case analysis, etc). The formal semantics of a text can be given by packing the whole text in a single statement and transforming it to the corresponding logical formula (which we call the *formula image* of the text). Then the text is declared correct whenever its formula image is deducible in the underlying logic. This approach would be simple and theoretically transparent but obviously impracticable. The SAD system implements a more intricate notion of text correctness which is formalized with the help of a logical calculus and can serve as a formal specification of a "correctness verifier" (SAD included).

The SAD project is the continuation of a project initiated by academician V. Glushkov at the Institute for Cybernetics in Kiev more than 30 years ago

[2]. Its original title was "Evidence Algorithm". Three main components had to be developed: an inference engine (we call it *prover* below) that implements the basic level of evidence, an extensible collection of tools (we call it *reasoner*) to reinforce the basic engine, and a formal input language which must be close to natural mathematical language and easy to use. Actually, a working version of SAD is implemented [3–5] and available online at `http://ea.unicyb.kiev.ua`.

In a general setting, SAD may be positioned as a declarative style proof assistant/verifier that accepts input texts written in the formal language ForTheL [6, 5], uses an automated first-order prover as the basic inference engine and possesses an original reasoner. The closest to our approach is Mizar system [7] — the oldest and most known proof assistant working with proofs of declarative style.

2 ForTheL language

Like usual mathematical text, a ForTheL text consists of definitions, axioms, hypotheses, conjectures, proofs. ForTheL is a controlled natural language: its syntax follows the rules of English grammar. ForTheL sentences are of three kinds: assumptions ("`Let S be a finite set.`", "`Assume that m is less than n.`"), selections ("`Take an even prime number X.`"), and affirmations ("`If p divides n-p then p divides n.`"). Series of transformations which convert a ForTheL statement to its *formula image* determine the semantics of the statement. For example, the formula image of the statement "`all closed subsets of any compact set are compact`" is: $\forall$`A` ((`A is a set` $\wedge$ `A is compact`) $\supset$ $\forall$`B` ((`B is a subset of A` $\wedge$ `B is closed`) $\supset$ `B is compact`)). Sentences, compound sections and a ForTheL text itself are given formula images, too.

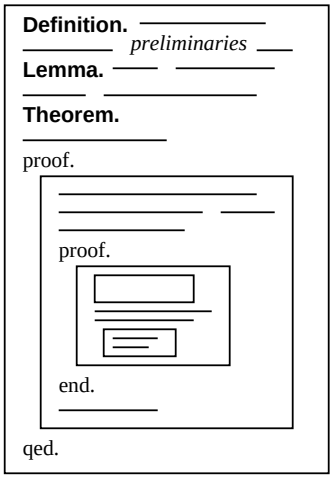

Fig. 1. ForTheL text's structure

Affirmations and selections can be accompanied with a proof. ForTheL supports various proof schemes like proof by contradiction, by case analysis, and by general induction. Proofs need not to be ultimately detailed: reasoning "steps" can be as large as the deductive facilities of a verifier (e.g. SAD) can manage. Consider for example an excerpt of a verified formalization of the Tarski's fixed point theorem:

```
Definition DefCLat. A complete lattice is a set S such that
  every subset of S has an infimum in S and a supremum in S.

Definition DefIso.  f is isotone iff for all x,y << Dom f
    x <= y  =>  f(x) <= f(y).
```

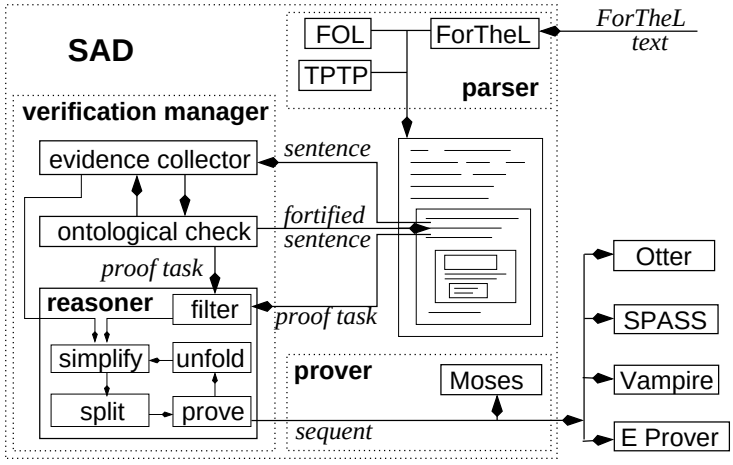

Fig. 2. Architecture of SAD

```
Theorem Tarski.
  Let U be a complete lattice and f be an isotone function on U.
  Let S be the set of fixed points of f. S is a complete lattice.
Proof.
  Let T be a subset of S.
  Let us show that T has a supremum in S.
    Take P = { x << U | f(x) <= x and x is an upper bound of T in U }.
    Take an infimum p of P in U.
    f(p) is a lower bound of P in U and an upper bound of T in U.
    Hence p is a fixed point of f and a supremum of T in S.
  end.
  Let us show that T has an infimum in S.
    Take Q = { x << U | f(x) >= x and x is a lower bound of T in U }.
    Take a supremum q of Q in U.
    f(q) is an upper bound of Q in U and a lower bound of T in U.
    Hence q is a fixed point of f and an infimum of T in S.
  end.
qed.
```

3 System for Automated Deduction

The principal components of SAD are shown in Figure 2.

Parser accepts a ForTheL text, checks its syntactical correctness and converts the text into a normalized form that will be convenient for further processing (e.g. all synonyms are replaced with their canonical representatives).

Verification manager makes her round through the normalized text section by section, checking the ontological and logical correctness. If a section (say, $\mathbb{A}$) is a

sentence, then it is first sent to the *evidence collector* that accumulates so called term properties for the term occurrences in the formula image of $\mathbb{A}$.

Term properties are literals that tell us something important about a given term occurrence. A literal (i.e. an atomic formula or its negation) L is considered to be a property of a term t in a context Γ, whenever t is a subterm of L and L is deducible in Γ. The most important purpose of term properties is to hold information about term "types", which is usually expressed by an atomic statement of the form "t `is a` $\langle class \rangle$". Some simple properties, like non-emptiness, are highly useful, too.

Fortified with the found properties, occurrences of terms and atoms are passed to the *ontological checker*. For each symbol occurrence, the checker looks through the text processed so far for an appropriate definition or signature extension. The deductive core of the system, so called *reasoner*, is used to prove the instantiated guards. The results of ontological checking (the applicable definitions) together with the collected properties are used in evidence collection for outer occurrences.

Then the verification manager processes the section $\mathbb{A}$ according to the rules of the special Calculus of Text Correctness, **CTC**, which we do not describe in this paper. Generally, if $\mathbb{A}$ contains a statement to prove, then a new verification cycle is started to verify the submitted proof, possibly empty. This statement becomes the initial *thesis* of the new cycle. The thesis may be gradually simplified as the proof proceeds: when the thesis is an implication and we assume its antecedent, or when it is a conjunction and we affirm its part. At the end of the proof (i.e. immediately, if there was no proof in the text), the current thesis is sent to the *reasoner*. In other words, a ForTheL proof hints the verification manager to split the statement being proved into several proof tasks and the rules of thesis transformation guarantee the soundness of splitting.

Reasoner deals with proof tasks of the form $\Gamma \vdash F$. This module can be viewed as a kind of automated heuristic based prover, supplied with a collection of proof task transformation rules. This collection is not intended to form a complete logic calculus. The purpose of the reasoner is not to find the entire proof on its own, but rather to simplify inference search for the *background prover*.

At present, the capabilities of the reasoner are as follows: propositional goal splitting, formula simplification with respect to accumulated term properties, simple filtering of premises according to explicit references in the text, incremental definition expansion.

The reasoner of SAD uses term properties to simplify goal formulas and formulas which arise from definition expansion: any literal that appears to hold as a property of some of its subterms can be replaced by logical constant "truth" (indeed, it can be deduced from the current context and, hence, is redundant). Similarly, a literal can be replaced by "false", if its complement occurs among the properties of its subterms.

Background prover is a combinatorial automated prover in classical first-order logic, whose duty is to complete the proofs started by the reasoner. If the back-

ground prover fails to find the inference at some instant, the reasoner may continue the proof task transformation or try an alternative way, or reject the text.

The background prover is independent from SAD by design, so that an external theorem prover can be used. Our experiments (see below) were performed with Otter [8], SPASS [9], Vampire [10], and E [11]. Note that this feature of SAD provides us with a (yet another) scale to compare automated theorem provers: trying them on relatively simple problems in complex and heavily redundant contexts rather than on hard problems with a pre-adjusted set of relevant premises (mostly the case for problems in the famous TPTP library [12]).

Also there exists the native background prover of SAD, called Moses. It is based on an original goal-driven sequent calculus [4, 5].

4 Experiments

In the course of development of SAD, we have conducted a number of essays on formalization and verification of non-trivial mathematical results:

- Ramsey's Finite and Infinite theorems.
- Stability of a refinement relation over a number of operations on program specifications [13].
- Cauchy-Bouniakowsky-Schwarz inequality.
- The square root of a prime number is irrational: 30 statements in preliminaries (integer numbers), 5 definitions, 7 lemmas, about 50 sentences in the proof of the main lemma (any prime dividing a product divides one of the factors), 10 sentences in the proof of the theorem (see [5] for detailed explanation of this experiment).
- Chinese remainder theorem and Bezout's identity in terms of abstract rings: 25 statements in preliminaries (ring axioms, operations on sets), 7 definitions (ideal, principal ideal, greatest common divisor, etc), 3 lemmas, 8 sentences in the proof of CRT, about 30 sentences in the proof of Bezout's identity.
- Tarski's fixed point theorem (cited above): 11 statements in preliminaries (ordered sets), 7 definitions (upper and lower bounds, supremum, infimum, complete lattice, isotone function, fixed point), 2 lemmas, 18 sentences in the proof of the theorem.

The texts listed above were written in ForTheL and automatically verified in SAD using different background provers. The best results were obtained with SPASS. This is due, in particular, to its original technique of handling sort-like information, which abounds in mathematical texts.

5 Conclusion

SAD is a powerful system and its power lies in its reasoning facility. Experiments show that, for example, the specific strategy of definition processing contributes a lot to the success of the whole verification process. If we use definitions

straightforwardly — convert them into formula images and add the corresponding premises to the sequent that goes into a prover — we have no chance to verify the proof of Tarski fixed-point theorem as it is formulated above, even when the winner of CASC competitions is chosen as the background prover.

SAD is not a perfect system (if any!). One can easily see how it may be improved and developed. Our research and implementation plans with respect to SAD are: extend ForTheL and SAD with some means to talk and reason about second-order objects (functions, vectors, sequences) and operations on them; develop and implement a mathematical library of SAD to accumulate verified portions of mathematical knowledge and to support further (deeper) advances in formalization.

References

1. Wang, H.: Towards mechanical mathematics. IBM J. of Research and Development **4** (1960) 2–22
2. Glushkov, V.M.: Some problems of automata theory and artificial intelligence (in Russian). Kibernetika **2** (1970) 3–13
3. 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
4. Lyaletski, A., Paskevich, A., Verchinine, K.: Theorem proving and proof verification in the system SAD. In Asperti, A., Bancerek, G., Trybulec, A., eds.: Mathematical Knowledge Management: Third International Conference, MKM 2004. Volume 3119 of Lecture Notes in Computer Science., Springer (2004) 236–250
5. Lyaletski, A., Paskevich, A., Verchinine, K.: SAD as a mathematical assistant — how should we go from here to there? Journal of Applied Logic **4**(4) (2006) 560–591
6. Vershinin, K., Paskevich, A.: ForTheL — the language of formal theories. International Journal of Information Theories and Applications **7**(3) (2000) 120–126
7. Trybulec, A., Blair, H.: Computer assisted reasoning with Mizar. In: Proc. 9th International Joint Conference on Artificial Intelligence. (1985) 26–28
8. McCune, W.: Otter 3.0 reference manual and guide. Tech. Report ANL-94/6, Argonne National Laboratory, Argonne, USA (1994)
9. Weidenbach, C., Brahm, U., Hillenbrand, T., Keen, E., Theobald, C., Topic, D.: SPASS version 2.0. In Voronkov, A., ed.: Automated Deduction: 18th International Conference, CADE-18. Volume 2392 of Lecture Notes in Computer Science., Springer-Verlag (2002) 275–279
10. Riazanov, A., Voronkov, A.: The design and implementation of VAMPIRE. AI Communications **15**(2–3) (2002) 91–110
11. Schulz, S.: System Description: E 0.81. In Basin, D., Rusinowitch, M., eds.: Automated Reasoning: 2nd International Joint Conference, IJCAR 2004. Volume 3097 of Lecture Notes in Artificial Intelligence., Springer-Verlag (2004) 223–228
12. Sutcliffe, G., Suttner, C.B., Yemenis, T.: The TPTP problem library. In Bundy, A., ed.: Automated Deduction: 12th International Conference, CADE-12. Volume 814 of Lecture Notes in Computer Science., Springer-Verlag (1994) 252–266
13. Mammar, A.: Un environnement formel pour le développement d'application bases de données. PhD thesis, Conservatoire National des Arts et Métiers, France (2002)