On Correctness of Mathematical Texts from a Logical and Practical Point of View

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

[1] Université Paris 12, IUT Sénart/Fontainebleau, 77300 Fontainebleau, France
[2] Kyiv National Taras Shevchenko University, Faculty of Cybernetics, 03680 Kyiv, Ukraine*

Abstract. Formalizing mathematical argument is a fascinating activity in itself and (we hope!) also bears important practical applications. While traditional proof theory investigates deducibility of an individual statement from a collection of premises, a mathematical proof, with its structure and continuity, can hardly be presented as a single sequent or a set of logical formulas. What is called "mathematical text", as used in mathematical practice through the ages, seems to be more appropriate. However, no commonly adopted formal notion of mathematical text has emerged so far.

In this paper, we propose a formalism which aims to reflect natural (human) style and structure of mathematical argument, yet to be appropriate for automated processing: principally, verification of its correctness (we consciously use the word rather than "soundness" or "validity").

We consider mathematical texts that are formalized in the ForTheL language (brief description of which is also given) and we formulate a point of view on what a correct mathematical text might be. Logical notion of correctness is formalized with the help of a calculus. Practically, these ideas, methods and algorithms are implemented in a proof assistant called SAD. We give a short description of SAD and a series of examples showing what can be done with it.

1 Introduction

The question in the title of the paper is one to which we would like to get an answer formally. What we need to this aim is: a formal language to write down texts, a formal notion of correctness, a formal reasoning facility. And no matter what the content of the text in question is.

The idea to use a formal language along with formal symbolic manipulations to solve complex "common" problems, already appeared in G.W. Leibniz's writings (1685). The idea seemed to obtain more realistic status only in the early sixties

* This research is being supported by the INTAS project 05-1000008-8144. Some of its parts were performed within the scope of the project M/108-2007 in the framework of the Ukrainian-French Programme "Dnipro".

S. Autexier et al. (Eds.): AISC/Calculemus/MKM 2008, LNAI 5144, pp. 583–598, 2008.

of the last century when first theorem proving programs were created [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 some more or less complex deductions for you. Usually one has in mind either long but routine inferences or a kind of case analysis with enormously large number of possible cases. Both situations are embarrassing and "fault intolerant" for humans.

Mathematical text is not a simple sequence of statements, neither a linear representation of a sequent tree, nor a λ-term coding a proof. It is a complex object that contains axioms, definitions, theorems, and proofs of various kinds (by contradiction, by induction, by case analysis, etc). What its "correctness" might stand for? The formal semantics of a text can be given by packing the whole text in a single statement and considering the corresponding logical formula (which we may call the *formula image* of the text). Then the text is declared correct whenever its formula image is deducible in the underlying logic. The approach is simple, theoretically transparent but absolutely impracticable, e.g. the precise notion of correctness obtained in this way can hardly be considered as a formal specification of a proof assistant we would like to implement. That's why we develop a specific notion of text correctness that, though being less straightforward, can be formalized with the help of a logical calculus on one hand and can serve as a formal specification on the other.

Our approach to mathematical text correctness is implemented in the proof assistant called SAD (System for Automated Deduction). 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]. The title of the original project was "Evidence Algorithm" and its goal was to help a working mathematician to verify long and tiresome but routine reasonings. To implement that idea, 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. Today, a working version of the SAD system exists [3,4,5] and is available online at `http://nevidal.org.ua`.

What is the place of SAD in the world of proof assistants w.r.t. proof representation style? Actually, we observe four major approaches to formal presentation of a mathematical proof (see also [6] for an interesting and detailed comparison).

Interactive proof assistants, such as Coq [7], Isabelle [8], PVS [9], or HOL derivatives [10], work with "imperative" proofs, series of tactic invocations.

Systems based on the Curry-Howard isomorphism, such as de Bruijn's Automath [11] or Coq, consider a proof of a statement as a lambda term inhabiting a type that corresponds to the statement. Since writing such a proof directly is difficult and time-consuming, modern systems of the kind let user build a proof in an interactive tactic-based fashion and construct the final proof term automatically.

The third branch deals with "declarative" proofs, which are structured collections of hypotheses, conjectures and claims expressed in the same language as the axioms and theorems themselves. The Mizar system [12] is the oldest and most known proof assistant working with proofs of declarative style. Isabelle, with introduction of Isar [13], accepts declarative proofs, too. Declarative proof presentation is employed and thoroughly studied in the works on Mathematical Vernacular started by N. de Bruijn [11] and later extended to Weak Type Theory [14,15] and MathLang [16]. In particular, MathLang and the ForTheL language, presented below, share a lot of similar traits owed to the common striving for a natural-like formal mathematical language.

Finally, there are systems that do not use user-given proofs and rely instead on proof generation methods: planning, rewriting, or inference search facilities to deduce each claim from premises and previously proved statements. The systems ACL2 (successor to Nqthm) [17], λCLAM [18], Theorema [19] and any classical automated prover (e.g. Otter) can be considered as proof assistants of the kind.

In a general setting, SAD may be positioned as a declarative style proof verifier that accepts input texts written in the special formal language ForTheL [20,4], uses an automated first-order prover as the basic inference engine and possesses an original reasoner (which includes, in particular, a powerful method of definition expansion).

The rest of the paper is organized as follows. In Section 2, we briefly describe the ForTheL language and write a ForTheL proof of Tarski's fixed point theorem. We define correctness of a ForTheL text with the help of a logical calculus in Section 3. We illustrate this calculus by verifying a simple text in Section 4. We conclude with a brief list of experiments on formalization performed in SAD.

2 ForTheL Language

Like any usual mathematical text, a ForTheL text consists of definitions, assumptions, affirmations, theorems, proofs, etc. Figure 1 gives an idea of what a ForTheL text looks like.

The syntax of a ForTheL sentence follows the rules of English grammar. Sentences are built of units: statements, predicates, notions (that denote classes of objects) and terms (that denote individual entities). Units are composed of syntactical primitives: nouns which form notions (e.g. "`subset of`") or terms ("`closure of`"), verbs and adjectives which form predicates ("`belongs to`", "`compact`"), symbolic primitives that use a concise symbolic notation for predicates and functions and allow to consider usual quantifier-free first-order formulas as ForTheL statements. Of course, just a little fragment of English is formalized in the syntax of ForTheL.

There are three kinds of sentences in the ForTheL language: assumptions, selections, and affirmations. Assumptions serve to declare variables or to provide some hypotheses for the following text. For example, the following sentences are typical assumptions: "`Let S be a finite set.`", "`Assume that m is greater than n.`". Selections state the existence of representatives of notions and can

be used to declare variables, too. Here follows an example of a selection: "`Take an even prime number X.`". Finally, affirmations are simply statements: "`If p divides n - p then p divides n.`". The semantics of a sentence is determined by a series of transformations that convert a ForTheL statement to a first-order formula, so called *formula image*. For example, the formula image of the statement "`all closed subsets of any compact set are compact`" is:

$$\forall\, \texttt{A}\,((\texttt{A is a set} \wedge \texttt{A is compact}) \supset \forall\, \texttt{B}\,((\texttt{B is a subset of A} \wedge \texttt{B is closed}) \supset \texttt{B is compact}))$$

The sections of ForTheL are: sentences, sentences with proofs, cases, and top-level sections: axioms, definitions, signature extensions, lemmas, and theorems. A top-level section is a sequence of assumptions concluded by an affirmation. Proofs attached to affirmations and selections are simply sequences of low-level sections. A case section begins with a special assumption called *case hypothesis* which is followed by a sequence of low-level sections (the "proof" of a case).

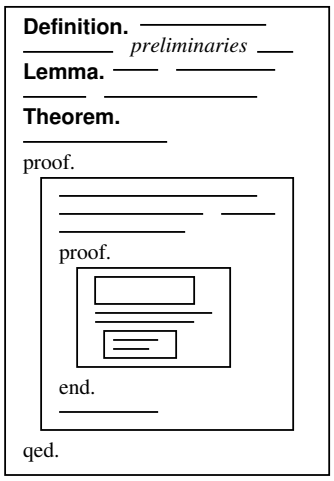

Fig. 1. ForTheL text's structure

Any section $\mathbb{A}$ or sequence of sections Δ has a formula image, denoted $|\mathbb{A}|$ or, respectively, $|\Delta|$. The image of a sentence with a proof is the same as the image of that sentence taken without proof. The image of a case section is the implication ($H \supset$ `thesis`), where H is the formula image of the case hypothesis and `thesis` is a placeholder for the statement being proved (see Section 3). The formula image of a top-level section is simply the image of the corresponding sequence of sentences.

The formula image of a sequence of sections $\mathbb{A}, \Delta$ is an existentially quantified conjunction $\exists \boldsymbol{x}_{\mathbb{A}}(|\mathbb{A}| \wedge |\Delta|)$, whenever $\mathbb{A}$ is a conclusion (affirmation, selection, case section, lemma, theorem); or a universally quantified implication $\forall \boldsymbol{x}_{\mathbb{A}}(|\mathbb{A}| \supset |\Delta|)$, whenever $\mathbb{A}$ is a hypothesis (assumption, axiom, definition, signature extension). Here, $\boldsymbol{x}_{\mathbb{A}}$ denotes the set of variables declared in $\mathbb{A}$ and can only be non-empty when $\mathbb{A}$ is an assumption or a selection. This set depends on the logical context of $\mathbb{A}$, since any variable which is declared above $\mathbb{A}$ can not be redeclared in $\mathbb{A}$. The formula image of the empty sequence is $\top$, the truth.

In this syntax, we can express various proof schemes like proof by contradiction, by case analysis, and by general induction. The last scheme merits special consideration. Whenever an affirmation is marked to be proved by induction, the system constructs an appropriate induction hypothesis and inserts it into the statement to be verified. The induction hypothesis mentions a binary relation which is declared to be a well-founded ordering, hence, suitable for

induction proofs. Note that we cannot express the very property of well-foundness in ForTheL (since it is essentially a first-order language), so that the correctness of this declaration is unverifiable and we take it for granted. After that transformation, the proof and the transformed statement can be verified in a first-order setting, and the reasoner of SAD has no need in specific means to build induction proofs. The semantics of this and other proof schemes is considered in more detail in the next section.

Is ForTheL practical as a formalization language? Our numerous experiments show that rather often a ForTheL text is sufficiently close to its hand-made prototype. 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 DefMono. f is monotone iff for all x,y << Dom f
    x <= y  =>  f(x) <= f(y).

Theorem Tarski.
  Let U be a complete lattice and f be an monotone 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 | x <= f(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 Text Correctness

We distinguish two types of correctness of a well-formed ForTheL text: ontological and logical.

Ontological correctness means that the text in question contains no occurrence of a symbol (constant, function, notion or relation) that comes from nowhere. First, every symbol must be either a signature symbol or be introduced by a definition. Second, in every occurrence of a symbol, the arguments, if any, must satisfy the guards of the corresponding definition or signature extension. Since ForTheL

is a one-sorted and untyped language, these guards can be arbitrary logical formulas. Therefore, the latter condition cannot be checked by purely syntactical means nor by type inference of any kind. Instead, it requires proving statements about terms inside complex formulas, possibly, under quantifiers. Such reasoning can be performed in a sound way using the notion of local images [21].

Ontological correctness is to ForTheL what type correctness is to typed languages. It allows early detection of formalization errors which otherwise could hardly be detected. Indeed, accidental ontological incorrectness most often implies logical incorrectness. However, it is much harder to trace a failure log of a prover back to an invalid occurrence than to discover it in the first place. Also, during ontological verification we obtain some important knowledge about the text which will be used later in logical verification.

$$\frac{\boldsymbol{x} = \mathcal{DV}_\Gamma(F) \qquad \Gamma \vdash \forall \boldsymbol{x}\,(F \supset G') \supset G \qquad \Gamma,\ (\texttt{assume}\ F) \triangleright_{G'} \Delta}{\Gamma\ \triangleright_G\ (\texttt{assume}\ \Theta_G(F)),\ \Delta}$$

$$\frac{\mathcal{DV}_\Gamma(F) = \varnothing \qquad \Gamma \triangleright_F \Lambda \qquad \Gamma \vdash (F \wedge G') \supset G \qquad \Gamma,\ (\texttt{affirm}\ F\ [\Lambda]) \triangleright_{G'} \Delta}{\Gamma\ \triangleright_G\ (\texttt{affirm}\ \Theta_G(F)\ [\Lambda]),\ \Delta}$$

$$\frac{\boldsymbol{x} = \mathcal{DV}_\Gamma(F) \qquad \Gamma \triangleright_{\exists \boldsymbol{x}\ F} \Lambda \qquad \Gamma \vdash \exists \boldsymbol{x}\,(F \wedge G') \supset G \qquad \Gamma, (\texttt{select}\ F\ [\Lambda]) \triangleright_{G'} \Delta}{\Gamma\ \triangleright_G\ (\texttt{select}\ \Theta_G(F)\ [\Lambda]),\ \Delta}$$

$$\frac{\mathcal{DV}_\Gamma(F) = \varnothing \qquad \Gamma, (\texttt{assume}\ F) \triangleright_G \Lambda \qquad \Gamma,\ (\texttt{case}\ (F \supset G)\ [\Lambda]) \triangleright_{G \vee F} \Delta}{\Gamma\ \triangleright_G\ (\texttt{case}\ (F \supset \texttt{thesis})\ [\Lambda]),\ \Delta}$$

$$\frac{\Gamma \triangleright_{\mathrm{IT}_t^\prec(G)} \Delta}{\Gamma\ \triangleright_G\ \Delta} \qquad \frac{\mathcal{DV}_{\Gamma,\Lambda}(\mathrm{IH}_t^\prec(G)) = \varnothing \qquad \Gamma \triangleright_{\mathrm{IT}_t^\prec(G)} \Lambda,\ (\texttt{assume}\ \mathrm{IH}_t^\prec(G)),\ \Delta}{\Gamma\ \triangleright_G\ \Lambda, \Delta}$$

$$\frac{\Gamma \vdash G}{\Gamma\ \triangleright_G} \qquad \frac{\Gamma \triangleright_\top \Lambda \qquad \Gamma,\ (\texttt{toplevel}\ |\Lambda|\ [\Lambda]) \triangleright_\top \Delta}{\Gamma\ \triangleright_\top\ (\texttt{toplevel}\ |\Lambda|\ [\Lambda]),\ \Delta} \qquad \frac{\Gamma,\ (\texttt{posit}\ F)\ \triangleright_\top \Delta}{\Gamma\ \triangleright_\top\ (\texttt{posit}\ F),\ \Delta}$$

Fig. 2. Calculus of Correctness **CTC**

Logical correctness is imposed on particular affirmations in the text: theorems, lemmas, intermediate statements in proofs. Any such affirmation must be deducible from its logical predecessors.

In what follows, a ForTheL section $\mathbb{A}$ will be considered as a triple $(T\,|\mathbb{A}|\,[\Lambda])$, where T denotes the section type, $|\mathbb{A}|$ is the formula image of $\mathbb{A}$, and Λ is the sequence of subsections of $\mathbb{A}$, if any. The type of $\mathbb{A}$ can be of the following: `toplevel` for any top-level section (axiom, definition, signature extension, theorem, lemma), `case` for a case section, `assume` for an assumption, `select` for a selection, `affirm` for an affirmation, `posit` for a postulate. Several remarks should be made here. A sentence with a supplied proof is considered to be of the same type as the same sentence without proof. They differ only in the third component of the triple, the list of subsections, which is empty for a sentence

without proof. Recall that the formula image of a sentence does not depend on presence of a proof, too. In a case section, the case hypothesis belongs to the formula image of the section and does not appear among its subsections. A postulate is an affirmation at the end of an axiom, definition, or signature extension; in other words, an affirmation which is not meant to be proved.

A formula F is *logically correct* in view of a sequence of sections Γ (the logical context of F), denoted $\Gamma \vdash F$, whenever F can be deduced in the classical first-order predicate calculus from the formula images of sections from Γ.

Logical correctness of a ForTheL text is deduced from logical correctness of particular formulas in view of appropriate sets of premises according to the Calculus of Text Correctness, or **CTC**, given in Figure 2.

In **CTC**, we infer sequents of the form $\Gamma \triangleright_G \Delta$. Only those sequents are allowed where every free variable of G occurs free in Γ ($\mathcal{FV}(G) \subseteq \mathcal{FV}(\Gamma)$), and neither Γ nor G contain occurrences of `thesis`.

In such a sequent, Δ is a sequence of sections whose correctness is being verified and Γ is a sequence of sections that logically precede Δ. The formula G is a *current thesis*: a formula which we want to deduce from Γ with the help of auxiliary reasoning in Δ (note the rule $\frac{\Gamma \vdash G}{\Gamma \triangleright_G}$). Verification consists in counter-applying the rules of the calculus, reducing the sequent $\Gamma \triangleright_G \Delta$ to $\vdash$-premises which are to be checked directly.

A ForTheL text Δ is said to be *logically correct* whenever $\triangleright_\top \Delta$ can be inferred.

The expression $\Theta_G(F)$ denotes the formula F where some occurrences of G are replaced with `thesis`. There may be several $\Theta_G(F)$ for given F and G. One can consider $\Theta_G(F)$ as an abbreviated form of F; when we counter-apply the rules of **CTC** during verification, we pass from $\Theta_G(F)$ back to F, i.e. we expand the abbreviation and eliminate the placeholder `thesis`.

The expression $\mathcal{DV}_\Gamma(F)$ stands for the set of variables which are declared in the formula F in view of Γ. Basically, that means that x does not occur freely in Γ and F "says" that x belongs to some class described by a notion (like in "`x is a fixed point of f`"). In a similar fashion, we define $\mathcal{DV}_\Gamma(\mathbb{A})$ and $\mathcal{DV}_\Gamma(\Delta)$ (with the proviso that only assumptions and selections can declare variables). Recall that the formula image of a sequence of sections, $|\Delta|$ actually depends on $\mathcal{DV}_\Gamma(\Delta)$ and, hence, on Γ. Note that any free variable in a well-formed text must be declared either in that very sentence or somewhere above, so that $\mathcal{DV}_\Gamma(\mathbb{A})$ contains those and only those free variables of $\mathbb{A}$ which do not occur free in Γ: $\mathcal{DV}_\Gamma(\mathbb{A}) = \mathcal{FV}(\mathbb{A}) \backslash \mathcal{FV}(\Gamma)$.

The expressions $\mathrm{IT}_t^{\prec}(G)$ and $\mathrm{IH}_t^{\prec}(G)$ stand for the *induction thesis* and *induction hypothesis*, respectively. They are defined as follows. For a given formula G of the form $\forall \boldsymbol{x}_1 (H_1 \supset \forall \boldsymbol{x}_2 (H_2 \supset \ldots \forall \boldsymbol{x}_n (H_n \supset F) \ldots))$, an arbitrary term t, and a binary relation symbol $\prec$:

$$\mathrm{IH}_t^{\prec}(G) = \forall \boldsymbol{x}'_1 (H_1\sigma \supset \forall \boldsymbol{x}'_2 (H_2\sigma \supset \ldots \forall \boldsymbol{x}'_n (H_n\sigma \supset (t\sigma \prec t \supset F\sigma)) \ldots))$$
$$\mathrm{IT}_t^{\prec}(G) = \forall \boldsymbol{x}_1 (H_1 \supset \forall \boldsymbol{x}_2 (H_2 \supset \ldots \forall \boldsymbol{x}_n (H_n \supset (\mathrm{IH}_t^{\prec}(G) \supset F)) \ldots))$$

where σ is the renaming substitution $[\boldsymbol{x}'_1/\boldsymbol{x}_1, \boldsymbol{x}'_2/\boldsymbol{x}_2, \ldots, \boldsymbol{x}'_n/\boldsymbol{x}_n]$ and $\boldsymbol{x}'_1$, $\boldsymbol{x}'_2$, …, $\boldsymbol{x}'_n$ are some fresh variables.

The induction thesis $\mathrm{IT}_t^{\prec}(G)$ is equivalent to the original thesis G on condition that $\prec$ denotes a well-founded ordering. Note that the well-foundness of $\prec$ cannot be finitely expressed in a first-order language and the calculus **CTC** takes it on trust. In other words, correctness of a ForTheL text is verified assuming the following axiom scheme of general induction: $\mathcal{I}nd = (\mathrm{IT}_{\mathbf{t}}^{\prec}(\mathbf{G}) \supset \mathbf{G})$, where $\mathbf{G}$ and $\mathbf{t}$ are placeholders for a formula and a term, respectively.

Note that $\mathrm{IT}_t^{\prec}(G)$ is equivalent to G if $\prec$ is the always false relation. Therefore the extension of first-order logic with the symbol $\prec$ and the axiom scheme $\mathcal{I}nd$ is conservative.

The first induction handling inference rule of **CTC** says that by proving the induction thesis, we automatically prove the initial one. In counter-application, it means that the verifier has the right to substitute the appropriate induction thesis for the initial thesis, when verifying a proof by induction. The second induction handling rule says additionally that the induction hypothesis need not to be put explicitly in the proof, but can be silently inserted there by the verifier. However, the induction hypothesis can not appear in the proof before all the free variables in it are declared.

Case section handling is another rule where an implicit logical predecessor, namely, the case hypothesis, is added by the verifier (in counter-application). Note that the Θ operation is not applied to a case hypothesis in the conclusion of the rule. That means that the word `thesis` can not appear in a ForTheL case hypothesis sentence, or it will not be verified.

Thesis handling. In the rules of **CTC** for assumptions, selections, and affirmations, we see $\vdash$-premises that relate a current thesis G to a new thesis G' (by "new", we mean that G' is used as the thesis for subsequent ForTheL proof sequence). Such a transformation of thesis reflects our perception of a proof development when a complex formula is being demonstrated.

For instance, whenever we want to prove a conjunction $F \wedge G$ and succeed to derive one part of it, say F, and write down the affirmation of F in the text, then the thesis can be reduced just to G. Furthermore, if we prove a universal statement about sets $\forall x\,(x$ `is a set` $\supset F)$ then we can begin by an assumption declaring x a set, thus reducing the thesis to F.

When we see such a connection between the current thesis and a sentence under consideration, we call that sentence *motivated*. Motivated affirmations, selections, assumptions allow to reduce the thesis to a new formula which would be probably simpler to prove. Sometimes, the connection is evident from the syntax, e.g. when the thesis is an implication and the assumption is the antecedent formula. Sometimes, the connection depends on several reasoning steps: for example, if variables S, T have been declared as sets and the current thesis is "`S is a subset of T`", then the assumption "`let x be an element of S`" is motivated and reduces the current thesis to "`x is an element of T`".

There is nothing special in non-motivated affirmations or selections. Whenever we meet such a sentence, we simply do not change the thesis. On the contrary, in a well-written mathematical proof, assumptions should be always motivated, i.e. "suggested" by a current thesis. A non-motivated assumption is an unjustified

narrowing of the search space. Also, it may happen that our reasoning capabilities are too weak to discover the justification.

Now, how can we infer $\Gamma \triangleright_G (\texttt{assume}\ F), \Delta$, if F is in no visible relation with G? Though the calculus **CTC** admits various solutions, in our implementation, the choice of the new thesis is guided by the form of Δ. Whenever the formula images of sentences in Δ contain occurrences of `thesis` (e.g. when there are case sections), we suppose that the proof of G continues under the non-motivated assumption and leave the thesis unchanged:

$$\frac{\boldsymbol{x} = \mathcal{DV}_\Gamma(F) \qquad \Gamma \vdash \forall \boldsymbol{x}\,(F \supset G) \supset G \qquad \Gamma,\ (\texttt{assume}\ F) \triangleright_G \Delta}{\Gamma\ \triangleright_G\ (\texttt{assume}\ \Theta_G(F)),\ \Delta}$$

The premise $\Gamma \vdash \forall \boldsymbol{x}\,(F \supset G) \supset G$ is nontrivial: its is equivalent to the disjunction $G \vee (\exists \boldsymbol{x}\,F)$. Recall that the free variables of G are all declared in Γ and thus cannot be among $\boldsymbol{x}$.

If `thesis` does not occur in the formula images in Δ, we suppose that the rest of the proof is a sort of independent argument which should be considered by itself. Therefore we take for the new thesis the image of the whole rest of the proof sequence. The inference is as follows:

$$\frac{\boldsymbol{x} = \mathcal{DV}_\Gamma(F) \qquad \Gamma \vdash \forall \boldsymbol{x}\,(F \supset |\Delta|) \supset G \qquad \Gamma, (\texttt{assume}\ F) \triangleright_{|\Delta|} \Delta}{\Gamma\ \triangleright_G\ (\texttt{assume}\ \Theta_G(F)),\ \Delta}$$

Note that the new variables in Δ, not known from Γ or F, are all bound in $|\Delta|$.

Each assumption in Δ is an antecedent in the new thesis $|\Delta|$ and therefore will be considered as motivated. Each affirmation or selection in Δ will reduce the thesis, too, so that at the end of Δ the thesis will be simply $\top$. The premise $\Gamma \vdash \forall \boldsymbol{x}\,(F \supset |\Delta|) \supset G$ finishes the demonstration by deducing G from the formula image of the proof sequence $(\texttt{assume}\ F),\ \Delta$.

Altogether, the following theorem can be seen as the statement of soundness of **CTC**:

Theorem 1. *Let Γ and Δ be arbitrary sequences of ForTheL sections and G, an arbitrary formula. If $\Gamma \triangleright_G \Delta$ can be inferred in* **CTC** *then $\mathcal{I}nd, \Gamma \vdash G$.*

Proof. The claim can be proved by induction on the number of steps in the inference of $\Gamma \triangleright_G \Delta$. Let us consider the last inference step. If it is made by a rule with $\triangleright_\top$ in conclusion, then G is $\top$, and the claim is trivial. Otherwise, we have seven cases to consider. We will denote the cases by the form of the conclusion of the corresponding inference rule.

Case $\Gamma \triangleright_G$. The premise of this rule is $\Gamma \vdash G$, hence the claim.

Case $\Gamma\ \triangleright_G\ (\texttt{assume}\ \Theta_G(F)), \Delta$. By the premises of the rule, we have $\Gamma \vdash \forall \boldsymbol{x}\,(F \supset G') \supset G$ and $\Gamma, (\texttt{assume}\, F) \triangleright_{G'} \Delta$, where $\boldsymbol{x} = \mathcal{DV}_\Gamma(F) = \mathcal{FV}(F) \backslash \mathcal{FV}(\Gamma)$. By the induction hypothesis, the latter implies $\mathcal{I}nd, \Gamma, F \vdash G'$. Also, $\mathcal{FV}(G) \subseteq \mathcal{FV}(\Gamma)$ and $\mathcal{FV}(G') \subseteq \mathcal{FV}(\Gamma) \cup \mathcal{FV}(F)$. Therefore, $\mathcal{FV}(F \supset G') \backslash \mathcal{FV}(\Gamma) = \boldsymbol{x}$. Hence $\mathcal{I}nd, \Gamma \vdash \forall \boldsymbol{x}\,(F \supset G')$ and we have the claim.

Case $\Gamma\ \triangleright_G\ (\texttt{affirm}\ \Theta_G(F)\ [\Lambda]), \Delta$. This is subsumed by the next case.

Case $\Gamma \triangleright_G (\texttt{select}\, \Theta_G(F)\, [\Lambda]), \Delta$. We have $\Gamma \triangleright_{\exists \boldsymbol{x}\, F} \Lambda$ and $\Gamma \vdash \exists \boldsymbol{x}\, (F \wedge G') \supset G$, and $\Gamma, (\texttt{select}\ \ F\ \ [\Lambda]) \triangleright_{G'} \Delta$, where $\boldsymbol{x} = \mathcal{FV}(F) \backslash \mathcal{FV}(\Gamma)$. By the induction hypothesis, we have $\mathcal{I}nd, \Gamma \vdash \exists \boldsymbol{x}\, F$ and $\mathcal{I}nd, \Gamma, F \vdash G'$. Also, $\mathcal{FV}(G) \subseteq \mathcal{FV}(\Gamma)$ and $\mathcal{FV}(G') \subseteq \mathcal{FV}(\Gamma) \cup \mathcal{FV}(F)$. Hence $\mathcal{FV}(F \wedge G') \backslash \mathcal{FV}(\Gamma) = \boldsymbol{x}$ and $\mathcal{I}nd, \Gamma \vdash \forall \boldsymbol{x}\, (F \supset G')$. This implies $\mathcal{I}nd, \Gamma \vdash \exists \boldsymbol{x}\, (F \wedge G')$ and we have the claim.

Case $\Gamma \triangleright_G (\texttt{case}\ \ (F \supset \texttt{thesis})\ \ [\Lambda]), \Delta$. From the premises, we obtain $\Gamma, (\texttt{assume}\ \ F) \triangleright_G \Lambda$ and $\Gamma, (\texttt{case}\ \ (F \supset G)\ \ [\Lambda]) \triangleright_{G \vee F} \Delta$. Also, $\mathcal{DV}_\Gamma(F) = \varnothing$ which means that $\mathcal{FV}(F), \mathcal{FV}(G) \subseteq \mathcal{FV}(\Gamma)$. By the induction hypothesis, we have $\mathcal{I}nd, \Gamma, F \vdash G$ and $\mathcal{I}nd, \Gamma, (F \supset G) \vdash (G \vee F)$. The former gives $\mathcal{I}nd, \Gamma \vdash (F \supset G)$. The latter gives $\mathcal{I}nd, \Gamma, (F \supset G) \vdash G$ and we have the claim.

Cases $\Gamma \triangleright_G \Delta$ and $\Gamma \triangleright_G \Delta, \Lambda$ (induction handling rules). By the premise of the rule and the induction hypothesis, we have $\mathcal{I}nd, \Gamma \vdash \mathrm{IT}_t^{\prec}(G)$. By definition of $\mathcal{I}nd$, we have the claim. □

4 Verification Example

Let us consider an example of a well-formed ForTheL text which, while being simple, contains proofs by case analysis and by induction (the symbol `-<-` below denotes a well-founded binary relation).

```
[number/numbers]     # let the parser know it is the same word

Signature Nat.                                          # 0
  A natural number is a notion.                         # 0.0
Signature Zer.                                          # 1
  0 is a natural number.                                # 1.0
Signature Suc.                                          # 2
  Let i be a natural number.                            # 2.0
  succ i is a natural number.                           # 2.1
Signature Add.                                          # 3
  Let i,j be natural numbers.                           # 3.0
  i + j is a natural number.                            # 3.1
Signature Ord.                                          # 4
  Let i,j be natural numbers.                           # 4.0
  i -<- j is an atom.                                   # 4.1

Axiom ZerSuc.                                           # 5
  For any natural number i if i != 0 then               # 5.0
  there exists a natural number j such that succ j = i.
Axiom AddZer.                                           # 6
  For any natural number i (i + 0 = i).                 # 6.0
Axiom AddSuc.                                           # 7
  For all natural numbers i,j (i + succ j = succ (i+j)). # 7.0
Axiom OrdSuc.                                           # 8
  For any natural number i (i -<- succ i).              # 8.0

Lemma ZerAdd.                                           # 9
  For any natural number i (0 + i = i).                 # 9.0
```

```
Proof by induction.
  Let i be a natural number.                              # 9.0.0
  Case i = 0.                                             # 9.0.1
  obvious.
  Case i != 0.                                            # 9.0.2
    Take a natural number j such that succ j = i.         # 9.0.2.0
    We have j -<- i.                                      # 9.0.2.1
    Hence 0 + j = j.                                      # 9.0.2.2
    Then we have the thesis.                              # 9.0.2.3
  end.
qed.
```

Note the numerical indexes in the comments. Each index denotes a position of a particular ForTheL section in the text. For example, 9.0 is the position of the main affirmation in the lemma `ZerAdd`, 9.0.0 is the position of the starting assumption in the proof, 9.0.1 and 9.0.2 point at the case sections.

Let us reconsider this text with the formula images in place of ForTheL sentences. In what follows, $t\,\varepsilon\,\mathrm{NatNum}$ stands for "t `is a natural number`".

```
toplevel Nat
   posit  ∀x (x ε NatNum ⊃ ⊤)
toplevel Zer
   posit  ∀x (x ≈ 0 ⊃ x ε NatNum)
toplevel Suc
   assume  i ε NatNum
   posit  ∀x (x ≈ succ i ⊃ x ε NatNum)
toplevel Add
   assume  i ε NatNum ∧ j ε NatNum
   posit  ∀x (x ≈ i + j ⊃ x ε NatNum)
toplevel Ord
   assume  i ε NatNum ∧ j ε NatNum
   posit  i ≺ j ⊃ ⊤
toplevel ZerSuc
   posit  ∀i (i ε NatNum ⊃ i ≠ 0 ⊃ ∃j (j ε NatNum ∧ succ j ≈ i))
toplevel AddZer
   posit  ∀i (i ε NatNum ⊃ i + 0 ≈ i)
toplevel AddSuc
   posit  ∀i (i ε NatNum ⊃ ∀j (j ε NatNum ⊃ i + (succ j) ≈ succ (i + j)))
toplevel OrdSuc
   posit  ∀i (i ε NatNum ⊃ i ≺ succ i)
toplevel ZerAdd
   affirm  ∀i (i ε NatNum ⊃ 0 + i ≈ i)
      assume  i ε NatNum
      case  i = 0 ⊃ thesis
      case  i ≠ 0 ⊃ thesis
         select  j ε NatNum ∧ (succ j) ≈ i
         affirm  j ≺ i
         affirm  0 + j ≈ j
         affirm  thesis
```

We are going to study the inference steps which prove correctness of the main lemma (the top level of the text is pretty trivial). In order to fit into the page width we will write position indexes in parentheses in place of the corresponding sections. We proceed in a bottom-top manner, moving from the desired conclusion to axioms.

$$\dfrac{\dfrac{\dfrac{\Gamma \rhd_{\top} (9.0)}{\Gamma \rhd_{|(9.0)|} (9.0.0), (9.0.1), (9.0.2)} \qquad \Gamma \vdash (|(9.0)| \wedge \top) \supset \top}{\Gamma \rhd_G (9.0.0), \mathbb{A}, (9.0.1), (9.0.2)} \qquad \dfrac{\Gamma, (9.0) \rhd_{\top}}{\Gamma, (9.0) \vdash \top}}$$

where

$$\begin{aligned} \Gamma &= (0), \ldots, (8) \\ \mathbb{A} &= (\texttt{assume}\ H) \\ |(9.0)| &= \forall i\, (i\, \varepsilon\, \mathrm{NatNum} \supset 0 + i \approx i) \\ G = \mathrm{IT}_i^{\prec}(|(9.0)|) &= \forall i\, (i\, \varepsilon\, \mathrm{NatNum} \supset (H \supset 0 + i \approx i)) \\ H = \mathrm{IH}_i^{\prec}(|(9.0)|) &= \forall i'\, (i'\, \varepsilon\, \mathrm{NatNum} \supset (i' \prec i \supset 0 + i' \approx i')) \end{aligned}$$

Note the fragment of inference where we apply the induction rule. Instead of proving the statement of the affirmation (9.0) as is, we descend into the proof with a weakened current thesis G having the additional induction hypothesis H.

We begin by inserting that induction hypothesis H into the proof. Note that the variable i which is free in H is declared in (9.0.0) and therefore known at the position of added hypothesis. Also note how the two assumptions reduce the current thesis from G to G' and then to G''.

$$\dfrac{\dfrac{\Gamma \rhd_G (9.0.0), \mathbb{A}, (9.0.1), (9.0.2)}{\Gamma \vdash \forall i\, (i\, \varepsilon\, \mathrm{NatNum} \supset G') \supset G} \qquad \Gamma, (9.0.0) \rhd_{G'} \mathbb{A}, (9.0.1), (9.0.2)}{\Gamma, (9.0.0) \vdash (H \supset G'') \supset G'} \qquad \Gamma, (9.0.0), \mathbb{A} \rhd_{G''} (9.0.1), (9.0.2)$$

where

$$G' = (H \supset 0 + i \approx i) \qquad\qquad G'' = (0 + i \approx i)$$

The first case section is very short. Note that `thesis` in the formula image of (9.0.1) is replaced with the actual thesis in $(9.0.1)'$:

$$\dfrac{\dfrac{\dfrac{\Gamma, (9.0.0), \mathbb{A} \rhd_{G''} (9.0.1), (9.0.2)}{\Gamma, (9.0.0), \mathbb{A}, \mathbb{C}_1 \rhd_{G''}} \qquad \Gamma, (9.0.0), \mathbb{A}, (9.0.1)' \rhd_{G'''} (9.0.2)}{\Gamma, (9.0.0), \mathbb{A}, \mathbb{C}_1 \vdash G''}}{\Gamma, i\, \varepsilon\, \mathrm{NatNum}, i \approx 0 \vdash 0 + i \approx i}$$

where

$$\mathbb{C}_1 = (\texttt{assume}\ i \approx 0) \quad (9.0.1)' = (\texttt{case}\ (i \approx 0 \supset G'')) \quad G''' = G'' \vee i \approx 0$$

The second case section is longer but no more complex:

$$\dfrac{\dfrac{\Delta_0 \rhd_{G'''} (\tau)}{\Delta_0, \mathbb{C}_2 \rhd_{G'''} (\tau.0), (\tau.1), (\tau.2), (\tau.3)} \qquad \dfrac{\dfrac{\Delta_0, (\tau)' \rhd_{G''' \vee i \not\approx 0}}{\Delta_0, (\tau)' \vdash G''' \vee i \not\approx 0}}{\vdash G'' \vee i \approx 0 \vee i \not\approx 0}$$

$$\dfrac{\dfrac{\Delta_1 \rhd_{G'''} (\tau.0), (\tau.1), (\tau.2), (\tau.3)}{\dfrac{\Delta_1 \rhd_{\exists j F_1}}{\Delta_1 \vdash \exists j F_1}} \qquad \Delta_1 \vdash \exists j\, (F_1 \wedge G''') \supset G''' \qquad \Delta_1, (\tau.0) \rhd_{G'''} (\tau.1), (\tau.2), (\tau.3)}$$

$$\dfrac{\dfrac{\Delta_2 \rhd_{G'''} (\tau.1), (\tau.2), (\tau.3)}{\dfrac{\Delta_2 \rhd_{j \prec i}}{\Delta_2 \vdash j \prec i}} \qquad \Delta_2 \vdash (j \prec i \wedge G''') \supset G''' \qquad \Delta_2, (\tau.1) \rhd_{G'''} (\tau.2), (\tau.3)}$$

$$\dfrac{\dfrac{\Delta_3 \rhd_{G'''} (\tau.2), (\tau.3)}{\dfrac{\Delta_3 \rhd_{0+j \approx j}}{\Delta_3 \vdash 0 + j \approx j}} \qquad \Delta_3 \vdash (0 + j \approx j \wedge G''') \supset G''' \qquad \Delta_3, (\tau.2) \rhd_{G'''} (\tau.3)}$$

$$\dfrac{\dfrac{\Delta_4 \rhd_{G'''} (\tau.3)}{\dfrac{\Delta_4 \rhd_{G'''}}{\dfrac{\Delta_4 \vdash G'''}{\Delta_4 \vdash 0 + i \approx i}}} \qquad \Delta_4 \vdash (G''' \wedge \top) \supset G''' \qquad \dfrac{\Delta_4, (\texttt{affirm}\ G'''\ [\ ]) \rhd_{\top}}{\Delta_4, (\texttt{affirm}\ G'''\ [\ ]) \vdash \top}}$$

where

$$\begin{aligned} \tau &= 9.0.2 & \Delta_0 &= \Gamma, (9.0.0), \mathbb{A}, (9.0.1)' \\ \mathbb{C}_2 &= (\texttt{assume}\ (i \not\approx 0)) & \Delta_1 &= \Delta_0, \mathbb{C}_2 \\ \Lambda &= (\tau.0), (\tau.1), (\tau.2), (\tau.3) & \Delta_2 &= \Delta_1, (\tau.0) \\ (\tau)' &= (\texttt{case}\ (i \not\approx 0 \supset G''')\ [\Lambda]) & \Delta_3 &= \Delta_2, (\tau.1) \\ F_1 &= j \,\varepsilon\, \mathrm{NatNum} \wedge \mathrm{succ}\ j \approx i & \Delta_4 &= \Delta_3, (\tau.2) \end{aligned}$$

A few comments should be made here. First, note the right-hand branch in the first inference fragment, where the goal $G'' \vee i \approx 0 \vee i \not\approx 0$ is proved. According to the rules of our calculus, each additional case section weakens the current thesis by putting it into a disjunction with the case's hypothesis. At the end of case analysis we have to prove the formula $G \vee H_1 \vee \cdots \vee H_n$, where G is the original thesis and $H_1, \ldots, H_n$ are explored cases. Yet, it is a good style to make case analyses exhaustive so that just the disjunction $H_1 \vee \cdots \vee H_n$ would hold at the end.

Second, a selection sentence is valid whenever we can prove the existence of named objects, i.e. the non-emptiness of the classes corresponding to the listed notions. While in the ForTheL text in question the selection (9.0.2.0) does not change the current thesis, that may happen when the current thesis is a statement of existence.

Third, pay attention that the affirmation (9.0.2.2) is a direct consequence of the induction hypothesis H and the previous affirmation (9.0.2.1). If the assumption $\mathbb{A}$ (whose formula image is H) were not inserted in the proof, the affirmation (9.0.2.2) could not be proved. However, one can write a proof where no sentence requires the induction hypothesis in order to be verified. For example, the whole proof of the affirmation (9.0) could be simply omitted. Then the system would try to prove just the induction thesis G, which is not difficult and does not require any induction reasoning capabilities.

Fourth, let us consider the last inference fragment. The formula image of the affirmation (9.0.2.3) is just the atomic formula `thesis`, which stands for the current thesis, G'''. Once having this affirmation proved, we have no pending obligations so that the new thesis is simply $\top$. Recall that G''' is the formula $0 + i \approx i \vee i \approx 0$, that is, we must either prove the initial thesis (G'') or reduce the task to the previous case.

Now, assuming the validity of all the first-order leaves ($\vdash$-sequents) in our derivation, we have demonstrated the *logical correctness* of the ForTheL text under consideration.

5 Experiments

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

- Ramsey's Finite and Infinite theorems.
- Cauchy-Bouniakowsky-Schwarz inequality.
- Newman's lemma about term-rewriting systems [21].
- 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 [4] for details).
- 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). This work have taught us many important lessons. To mention some:

- Formalization style is critical: the choice of symbols to introduce in definitions, the choice of preliminary facts, and even the way a proof is structured may decide whether the text will be verified or not.

- It is very desirable to comprehend the proofs before writing them in ForTheL. The SAD system may succeed to fulfil the gaps in a well thought-out reasoning, but it will not invent one for you.
- In most cases, the background prover finds the proof in three seconds — or does not find it at all.

6 Conclusion

While working on the development of the SAD project, we always felt that we strongly need an abstract, clear and transparent sight on the whole "formalization-and-verification" process. The calculus **CTC** proposed here is the first step in this direction. The next one will be a formal description of definition expansion and other reasoner's routines which is obviously missing now. Moreover, such a description must provide users with a kind of specific language to create their own reasoning strategies. Further on, something like a guide to problem formalization in a strong mathematical manner would be extremely useful, too.

Our abstract considerations of text verification resulted in the SAD system. Certainly, we could not give here a detailed description of all nice features of SAD. 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 cannot verify the proof of Tarski fixed point theorem as it is formulated above, even when a 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, 2–22 (1960)
2. Glushkov, V.M.: Some problems of automata theory and artificial intelligence (in Russian). Kibernetika 2, 3–13 (1970)
3. Lyaletski, A., Paskevich, A., Verchinine, K.: Theorem proving and proof verification in the system SAD. In: Asperti, A., Bancerek, G., Trybulec, A. (eds.) MKM 2004. LNCS, vol. 3119, pp. 236–250. Springer, Heidelberg (2004)
4. 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), 560–591 (2006)
5. Verchinine, K., Lyaletski, A., Paskevich, A.: System for Automated Deduction (SAD): a tool for proof verification. In: Pfenning, F. (ed.) CADE 2007. LNCS (LNAI), vol. 4603, pp. 398–403. Springer, Heidelberg (2007)

6. Wiedijk, F. (ed.): The Seventeen Provers of the World. LNCS (LNAI), vol. 3600. Springer, Heidelberg (2006)
7. Bertot, Y., Castéran, P.: Interactive Theorem Proving and Program Development. Coq'Art: The Calculus of Inductive Constructions. Texts in Theoretical Computer Science (EATCS), vol. XXV. Springer, Heidelberg (2004)
8. Nipkow, T., Paulson, L.C., Wenzel, M.: Isabelle/HOL: A Proof Assistant for Higher-Order Logic. LNCS, vol. 2283. Springer, Heidelberg (2002)
9. Owre, S., Rushby, J.M., Shankar, N.: PVS: a prototype verification system. In: Kapur, D. (ed.) CADE 1992. LNCS, vol. 607, pp. 748–752. Springer, Heidelberg (1992)
10. Gordon, M.J.C., Melham, T.F.: Introduction to HOL: a theorem proving environment for higher order logic. Cambridge University Press, Cambridge (1993)
11. Nederpelt, R.P., Geuvers, J.H., de Vrijer, R.C.(eds.): Selected Papers on Automath. Studies in Logic and the Foundations of Mathematics, vol. 133. North-Holland, Amsterdam (1994)
12. Trybulec, A., Blair, H.: Computer assisted reasoning with Mizar. In: Proc. 9th International Joint Conference on Artificial Intelligence, IJCAI 1985, pp. 26–28. Morgan Kaufmann, San Francisco (1985)
13. Wenzel, M.: Isar — a generic interpretative approach to readable formal proof documents. In: Bertot, Y., Dowek, G., Hirschowitz, A., Paulin, C., Théry, L. (eds.) TPHOLs 1999. LNCS, vol. 1690, pp. 167–184. Springer, Heidelberg (1999)
14. Kamareddine, F., Nederpelt, R.P.: A Refinement of de Bruijn's Formal Language of Mathematics. Journal of Logic, Language and Information 13(3), 287–340 (2004)
15. Jojgov, G., Nederpelt, R.: A path to faithful formalizations of mathematics. In: Asperti, A., Bancerek, G., Trybulec, A. (eds.) MKM 2004. LNCS, vol. 3119, pp. 145–159. Springer, Heidelberg (2004)
16. Kamareddine, F., Maarek, M., Wells, J.B.: Flexible encoding of mathematics on the computer. In: Asperti, A., Bancerek, G., Trybulec, A. (eds.) MKM 2004. LNCS, vol. 3119, pp. 145–159. Springer, Heidelberg (2004)
17. Kaufmann, M., Manolios, P., Moore, J.S.: Computer-Aided Reasoning: An Approach. Kluwer, Dordrecht (2000)
18. Richardson, J., Smaill, A., Green, I.: System description: Proof planning in higher-order logic with Lambda-Clam. In: Kirchner, C., Kirchner, H. (eds.) CADE 1998. LNCS (LNAI), vol. 1421, pp. 129–133. Springer, Heidelberg (1998)
19. Buchberger, B., Crăciun, A., Jebelean, T., Kovács, L., Kutsia, T., Nakagawa, K., Piroi, F., Popov, N., Robu, J., Rosenkranz, M., Windsteiger, W.: Theorema: Towards computer-aided mathematical theory exploration. Journal of Applied Logic 4(4), 470–504 (2006)
20. Vershinin, K., Paskevich, A.: ForTheL — the language of formal theories. International Journal of Information Theories and Applications 7(3), 120–126 (2000)
21. Paskevich, A., Verchinine, K., Lyaletski, A., Anisimov, A.: Reasoning inside a formula and ontological correctness of a formal mathematical text. In: Calculemus/MKM 2007 — Work in Progress, University of Linz, Austria. Number 07-06 in RISC-Linz Report Series, pp. 77–91 (2007)