Text-Orientated Formal Mathematics

Steffen Frerix and Peter Koepke
University of Bonn

The System for Automated Deduction (SAD) by Andrei Paskevich proof-checks texts written in nearly natural mathematical language, without explicit system commands or prover commands. Natural language features like soft typing by noun phrases guide efficient text processing and proof checking. We have implemented practical improvements of the checking algorithm and extensions of the input language, so that we are now able to process considerably longer and more complicated texts. These are immediately readable by mathematicians, as demonstrated by subsequent examples. This suggests an approach to formal mathematics through collections or libraries of texts in (a controlled) natural language.

1 Introduction

The common language of mathematics (CLM) has evolved to record and communicate mathematical statements and arguments succinctly and unambiguously. CLM combines natural language with symbolic terms and statements. Linguistic studies of CLM have shown that the techniques of formal linguistics are applicable to the language of mathematics and that parse results may be better than for arbitrary natural language because of a strong tendency in mathematics to avoid ambiguity [3, 5, 8]. On the other hand, in many texts an unambiguous reading may only be obvious to a (human) expert who understands the ambient and implicit assumptions and intentions and can supply missing details.

Ideally, software that assists mathematicians should communicate in the common language of mathematics: automatic provers should obtain their tasks as CLM statements; proof checkers should check the correctness of proof texts written in CLM. But current proof assistants are far from that ideal. Their input rather resembles computer languages, including proof commands, which are only readable after specific training but not by mathematicians at large [9].

The linguistic investigations suggest that certain *approximations* to CLM may be parsed faithfully on the sentence and text level. A formal grammar implemented in a parser defines a *controlled natural language* of accepted sentences which are simultaneously man- and machine-readable, like e.g., *Attempto controlled English* [2]. A good language design should comprise frequent constructs of CLM and avoid ambiguity. It should also allow linguistic figures which may be valuable for guiding the further text processing.

Our work is based on the System for Automated Deduction (SAD) by Andrei Paskevich which is a highly original and impressive prototypical system, mainly for proof checking. Reviewing the code of the program revealed that the algorithms left much room for optimization. A reduction of checking times from minutes to seconds was the starting point for experiments with much longer texts than the very special short miniatures that came with the original software. Our experiments suggested several extensions to the system some of which will be discussed in this article.

We begin the paper with relevant impressions of the original SAD approach. Then we report on notations and reasoning methods for sets and functions that we have added to the core of SAD, illustrated by an example. Originally the elegant notion-system of SAD could overburden a translation to first-order logic (FOL) with an abundance of type-guards. We have shown theoretically that many type-guards can

Submitted to:
UITP 2018

be safely omitted. The implementation of that reduction greatly speeds up proof checking. The new SAD is demonstrated with several short examples from various fields (although much longer texts are now possible).

We also indicate future work that will be carried out in close consultation with Andrei Paskevich, and discuss some implications. Our research suggests that interesting textbook mathematics and some advanced mathematics may be formulated in a SAD-like controlled natural language, so that formal mathematics could be carried out naturally in a text-orientated way, using collections of interlinked texts which are readable by men and machines.

2 The SAD programme

An early linguistically informed programme for mathematical proof checking was the Evidence Algorithm (EA) initiated by Victor Glushkov [4]. Proofs were viewed as arrangements of evident facts stated in the common language of mathematics and connected by evident proof rules. Initially, invoking strong automatic theorem provers was not envisaged.

These ideas were first implemented in Kiev, and they culminated in the System for Automatic Deduction developed in the doctoral project of Andrei Paskevich [7], see also http://nevidal.org/sad.en.html. The SAD proof checker accepts and checks some short mathematical proof texts written in a semi-natural *Formula Theory Language* (ForTheL). Some of these texts are surprisingly readable and elegant due to expressive natural language constructs and an efficient first-order logic which defines the necessary notions ($\sim$ sorts), functions and relations as needed [6].

The natural language parsing of SAD uses *ad hoc* methods which unfortunately also accept ungrammatical and unreadable texts. The NaProChe system of the second author et. al. [5] has some similarities with SAD. The checking mechanism is considerably weaker, but NaProChe employs proper natural language processing to exclude ungrammatical texts. NaProChe demonstrated that mathematical languages with symbolic context are compatible with modern formal linguistics.

ForTheL and NaProChe texts use a small subset of the common language of mathematics together with a small number of computer-orientated "commands". In both systems, a naive formal semantics of an accepted text T would be a tautology stating that the conjunction of the axioms of T implies the conjunction of the theorems of T.

3 A SAD example

Basic features of the ForTheL language and the SAD approach can be indicated by an actual example text which is accepted by our version of SAD.

[number/numbers] [divide/divides]

Signature. A rational number is a notion. Let s,r,q stand for rational numbers.

Signature. $r*q$ is a rational number.

Axiom. $r*q = q*r$.

Axiom. $r*(q*s) = (r*q)*s$.

Axiom. $q*s = q*r \Rightarrow s = r$.

Signature. A natural number is a rational number. Let n,m,k denote natural numbers.

Axiom. $n*m$ is a natural number.

Definition. $n \mid m$ iff there exists k such that $k*n = m$. Let n divides m stand for $n \mid m$. Let a divisor of m stand for a natural number that divides m.

Definition. n and m are coprime iff n and m have no common divisor.

Signature. n is prime is an atom. Let p denote a prime natural number.

Axiom. $p \mid n*m \Rightarrow p \mid n \vee p \mid m$.

Axiom. There exist coprime m,n such that $m*q = n$.

Theorem. $p = q*q$ for no rational number q.

Proof. Assume the contrary. Take a rational number q such that $p = q*q$. Take coprime m,n such that $m*q = n$. Then $p*(m*m) = n*n$. Therefore p divides n. Take a natural number k such that $n = k*p$. Then

$$\begin{aligned} p*(m*m) &= p*(k*n) \\ m*m &= p*(k*k) \end{aligned}$$

Hence p divides m. Contradiction. **qed.**

Let us make some comments.

1. The text shows that $\sqrt{p}$ is irrational for primes p. The examples in this paper have been optically improved by a faithful reformulation in LaTeX. Future versions of SAD will also allow LaTeX input.
2. The problem is formulated in a restricted natural language, which is immediately understandable by mathematicians. The language is apparently translatable to first-order logic, but it is more flexible and natural than just using logical conjuncts, quantifiers, and variables. There is, e.g., some typing and use of anonymous variables like in "a natural number that divides m". Observe the *ad hoc* linguistics in the first line which identifies some singulars with their plurals.
3. The text introduces concepts by *notions* and *axioms* without worrying about definitions in some foundational theory. Rational and natural numbers are introduced abstractly as domains with some axiomatic properties. The attentive reader will have noticed that some axioms are formulated just for the example and would have to be amended if we want to truly capture the standard rational numbers. The third axiom, e.g., usually requires that $q \neq 0$, but this is not relevant for our considerations of (non-zero) numbers. Notions are also used for soft-typing all variables and constants.
4. SAD is geared towards short self-contained texts - mathematical "miniatures". The original SAD distribution contains a number of examples from various fields of mathematics, often introducing concepts in simple first-order axiomatics which are intuitively seen as notions with a complex internal structure.

5. Axioms, definitions and theorems can be given names to explicitly refer to them in a proof and guide the search. By using the E theorem prover, however, this text can be checked without such additional hints. In fact, the Theorem could be proved fully automatically by our version of SAD, but omitting the proof would leave out the part that is mathematically most interesting to human readers.

4 Notions in SAD

Of central importance to SAD is the concept of a "notion". These form the grounds on which a ForTheL text is build. A text has certain base notions from which derived notions are constructed through combination with verbs and adjectives. In the example above we have (among many others) the notions "rational number", "natural number", "prime natural number". From the vocabulary present one can construct notions such as "natural number that divides every natural number", "natural number that is not a rational number". From a logical perspective, notions correspond to certain classes and can also be empty, as the last example shows.

New base notions, symbols and expressions are introduced in signature extensions and definitions. Such an introduction starts with a (possibly empty) sequence of assumptions on the occuring variables and is concluded with an introductory statement. These assumptions may be implicit through the use of pretyped variables. E.g., the introduction of the $*$-symbol in the above text assumes that both arguments are rational numbers.

Certain well-formedness conditions are imposed. For example any variable used in the introduced symbol must be "typed" beforehand, i.e., it must be assumed to belong to some notion, and no variables are allowed that do not appear in the introductory statement. These conditions are in accordance with standard mathematical practice and help detect formalization mistakes.

Intuitively, the assumptions made when defining a mathematical expression must be met for a particular instance of that symbol to be well-defined. Most mathematicians will agree that a statement like "The set of primitive recursive functions has a prime divisor" should not have a truth value, but that it is nonsensical since the notion of prime divisor is (usually) only introduced for integers. This leads us to the criterion of ontological correctness.

A text is ontologically correct if, loosely speaking, every application of a function symbol and every use of a predicate symbol is well-defined. This offers a natural approach to deal with partial functions and relations. SAD will check ontological correctness and we can therefore assure a priori that symbols are only applied to objects within their domain. Furthermore, the ontological check plays a role similar to type checking in strongly typed languages for the detection of common errors.

5 Text correctness

What does it mean for a ForTheL text T to be "correct"? While the demand that

$$\bigwedge \text{axioms of } T \Rightarrow \bigwedge \text{theorems of } T.$$

is a tautology, provides some kind of ultimately formal semantics, it can hardly be of use as a formal specification to a proof assistant. Instead, a calculus is employed, the *Calculus of Text Correctness* (CTC). According to the rules of the calculus SAD will walk through the text, generate certain proof obligations according to the nature of the particular statement examined and try do discharge them by using inbuilt reasoning methods or an external automated theorem prover (ATP).

ForTheL features four kinds of sections at the top level of a text: Axioms, propositions, signature extensions and definitions. Any statement can be declared to be an axiom or a proposition. Axioms are accepted as they are. For propositions, however, SAD will try to show derivability from preceding propositions and axioms. To support the verfication an author can supply a proof. Within a proof four kinds of mathematical operations are distinguished: assumptions, affirmations, selections and case hypothesis.

An assumption does not need to be proved but should in a certain sense be motivated, suggested by the form of the thesis. When we desire to show an implication for example, we may assume its antecedent in a proof. The other three operations generate proof obligations. A case hypothesis must imply the thesis and if we select an element from a certain class, we must prove that class to be nonempty. To aid this verification, an author may again supply a proof. In this way a ForTheL text has a block structure in which each subblock represents another layer of justification.

6 Implementing sets and functions

Sets and functions pervade mathematics. Across the disciplines, one uses the same established notations and conventions for set and function terms, so sets and functions should be embedded in the basic layers of a proof checking system.

While the existence of sets of the typical form

$$\{x \in a \mid \varphi(x, \vec{y})\},$$

can be justified in Zermelo-Fraenkel set theory (ZF) by a dedicated proof using the axiom scheme of separation, one can instead appeal to the above syntactic pattern together with the information that a is a set to ensure set existence. Similar remarks apply to function definitions and function existence, see below.

The original SAD system did not include this kind of syntactic reasoning which shifts the emphasis away from the central argument to routine set and function existence proofs. We have therefore enriched the ForTheL language and SAD checker with syntactic justification mechanisms for sets and functions. We demonstrate this by the next example which proves Cantor's Theorem that the powerset of a set is strictly larger than the set.

> [subset/subsets] [surject/surjects]
>
> Let M denote a set. Let f denote a function. Let the value of f at x stand for $f(x)$. Let f is defined on M stand for $\mathrm{Dom}(f) = M$. Let the domain of f stand for $\mathrm{Dom}(f)$.
>
> **Axiom.** The value of f at any element of the domain of f is a set.
>
> **Definition.** A subset of M is a set N such that every element of N is an element of M.
>
> **Definition.** The powerset of M is the set of subsets of M.
>
> **Definition.** f surjects onto M iff every element of M is equal to the value of f at some element of the domain of f.
>
> **Theorem.** No function that is defined on M surjects onto the powerset of M.
>
> **Proof.** Assume the contrary. Take a function f that is defined on M and surjects onto the powerset of M. Define
>
> $$N = \{x \in M \mid x \text{ is not an element of } f(x)\}.$$
>
> Then N is not equal to the value of f at any element of M. Contradiction. **qed.**

The example provides a text-orientated version of the standard diagonal argument, where sets are introduced using combinations of symbolic and natural language.

The notions "set" and "function" are now native to the system and can be used without an explicit signature extension. When stating an *axiom* like the powerset axiom, full comprehension like $\{x \mid x \subseteq M\}$ is allowed. In a *proof*, however, we restrict to separation. A formal semantics of the text is thus given by its axioms together with the separation scheme.

We have implemented a new ForTheL proof section called a *definition*. Here we allow syntactic set definitions of the form $N = \{t(\vec{y}) \in M \mid \varphi(\vec{y})\}$, where N is an identifier, t and M are terms and φ is any ForTheL statement with parameters $\vec{y}$. This is parsed into the first-order representation

$$\mathrm{aSet}(N) \wedge (\forall x.\ x \in N \leftrightarrow x \in M \wedge \exists \vec{y}.\ x = t(\vec{y}) \wedge \tilde{\varphi}),$$

where $\tilde{\varphi}$ denotes the first-order image of φ and $\mathrm{aSet}(N)$ is a predicate expressing set-hood. The proof obligation that is generated is (only) to show that M is a set.

Many function definitions can in part be justified syntactically. However, in certain situations, e.g., when a function is defined by case distinction, well-definedness of the function object has to be derived from the context. In a proof section of type "definition" a function object can be introduced by either using notation similar to lambda abstraction or by describing the function values and its domain.

The λ-like notation $f = \backslash x \in M.\ \varphi(x,y)$ is allowed, where f is an identifier, x and y are variables, M is a term and φ is a ForTheL statement. The syntax is translated to

$$\mathrm{aFunction}(f) \wedge (\forall x.\ x \in \mathrm{Dom}(f) \leftrightarrow x \in M) \wedge (\forall x.x \in \mathrm{Dom}(f) \rightarrow (\forall y.f[x] = y \leftrightarrow \tilde{\varphi}(x,y))),$$

where as above $\tilde{\varphi}$ denotes the first-order image of φ and $\mathrm{aFunction}(f)$ expresses function-hood. Therefore a function definition in ForTheL is split into the type declaration, the domain conditions and the computation rules. The proof task generated is that all choices are made from nonempty classes and that $\tilde{\varphi}$ does then indeed describe a well-defined function.

Apart from this general form there are several alternative syntactic constructs available to make work more natural, allowing a simple description of the function values, convenient case distinctions and definitions of functions on complex terms rather than variables. For example, complex conjugation could be defined by "Define $\overline{a+(i*b)} = a-(i*b)$ on $\mathbb{C}$." (The ForTheL source would read like " Define $\backslash\mathrm{conj}\{a+(i*b)\} = a-(i*b)$.", where $\backslash$conj has a double function: as a SAD function symbol, and as a LaTeXmacro which typesets as overlining.)

A function object can take any values definable by a first-order formula together with a choice operation. Since the image of a set under a function can axiomatically be declared to be a set in a ForTheL text, we can simulate any argument that takes place in ZFC, especially using the replacement scheme or the axiom of choice.

Functions are often defined recursively, and recursion is intimately connected with induction. SAD features induction as a distinguished proof method in a pecularly efficient way. The abstract symbol $\prec\!\!\prec$ is recognized by the system as an induction relation, and it implicitly assumes an induction scheme. When an author declares an inductive proof the system will check whether the inductive relation of the intended proof embeds into $\prec\!\!\prec$. Thereafter, the corresponding induction hypothesis is generated automatically and the author has to prove the inductive step.

Similarly, we can define functions by recursion over the relation $\prec\!\!\prec$ and other relations embedded into $\prec\!\!\prec$. Then a proof task of the form

$$D \rightarrow \forall y.(y \prec\!\!\prec x \rightarrow C(y)) \rightarrow L \rightarrow t \prec\!\!\prec x$$

is generated for each recursion term t, where D is the domain part of the function defintion, C is its computational part and L are local assumptions on the recursion term t. Intuitively this means that we can assume that the function has already been constructed below some value x, and we have to prove that in the definition for x we only call upon known values.

As an example consider the following ForTheL definition of a function:

Define $f[n] =$ **Case** $n = 0 \to 0$,
Case $n \neq 0 \to f[f[n-1]]$
for $n \in \mathbb{N}$.

We assume that this definition occurs in a ForTheL text that has set up a notion of natural number. We then have the domain part

$$D = \forall x.\ x \in \mathrm{Dom}(f) \leftrightarrow x \in \mathbb{N}$$

and the computational part

$$C(y) = y \in \mathrm{Dom}(f) \to (\forall z.\ f[y] = z \leftrightarrow (y = 0 \wedge z = 0) \vee (y \neq 0 \wedge z = f[f[y-1]])).$$

The only recursion term that occurs is $f[n-1]$ and its only (relevant) local assumption in the formula is $n \neq 0$. Therefore the proof task generated is

$$D \to \forall m.(m \prec\!\!\prec n \to C(m)) \to n \neq 0 \to f[n-1] \prec\!\!\prec n.$$

Notice that it is crucial here that we assume the function to be already constructed below n, otherwise the statement $f[n-1] \prec\!\!\prec n$ could not be shown (and would not even make sense). While many simple recursive definitions, like the factorial for example, can be proved to be wellfounded without this assumption, it is necessary in general.

For reasoning with both sets and functions, we implement special support. We keep track of elementhood conditions of sets, domain conditions and evaluations of functions. This information can the be added locally to a formula to try and make proof tasks easier for the ATP. It also contributes greatly to ontological verification.

Extensionality axioms can be a problem for ATPs since they tend to extend the search space significantly. Especially superposition provers suffer from them due to the nature of their underlying calculus. Extensionality is therefore treated specially in the new version of SAD. If an equality of sets or functions is to be proved, the corresponding extensionality axiom is added *locally* to the formula. In this way, no additional axiom is added to the global context that might impede other proof tasks.

7 Improved handling of soft typing

ForTheL is a softly typed language. Variables must be declared to belong to some notion before they can be used. Moreover, we cannot quantify unboundedly, but only over notions. The translation to first-order logic is done using type guards. Depending on the number of variables, the formulas obtained can therefore become quite large, which burdens the ATP used to discharge proof obligations. We have developed a more efficient handling of ForTheL typings in order to enable the verification of much coarser proofs.

A text that is ontologically correct can only control the behaviour of its symbols inside of their respective domains. We utilize knowledge of the symbol domains of a text to determine which assumptions play the role of typings and cannot essentially contribute to any derivation.

Every predicate or function that is introduced in a ForTheL text is accompanied by certain assumptions on their arguments that we call their domain conditions. For example the predicate isPrime(n) may have aNaturalNumber(n) as a domain condition. If φ is a formula occuring in the first-order image of a ForTheL text, we can determine for every variable x occuring in φ the most general domain that x is assumed to be in. These domain assumptions are then deleted from φ. Let us demonstrate the reduction on a short example.

> [number/-s]
>
> **Signature.** A real number is a notion.
>
> let x, y denote real numbers.
>
> **Signature.** $x * y$ is a real number.
>
> **Signature.** x is nonzero is an atom.
>
> **Signature.** Assume x is nonzero. x^{-1} is a real number.
>
> **Axiom.** Assume x and y are nonzero. $x * y$ is nonzero.
>
> **Axiom.** Assume x is nonzero. x^{-1} is nonzero.

The first order images of the two axioms are

$$(\text{aRealNumber}(x) \wedge \text{aRealNumber}(y)) \to (\text{isNonzero}(x) \wedge \text{isNonzero}(y)) \to \text{isNonzero}(x * y)$$
$$\text{aRealNumber}(x) \to \text{isNonzero}(x) \to \text{isNonzero}(x^{-1}).$$

Being a real number is a domain condition for all the other symbols involved while being nonzero is only a condition for $()^{-1}$. Therefore for the first formula, the most general domain condition is $(\text{aRealNumber}(x) \wedge \text{aRealNumber}(y)$ and for the second formula it is $\text{aRealNumber}(x) \wedge \text{isNonzero}(x)$. We can thus reduce these formulas to

$$\text{isNonzero}(x) \wedge \text{isNonzero}(y) \to \text{isNonzero}(x * y)$$
$$\text{isNonzero}(x^{-1}).$$

This reduction corresponds to usual mathematical thinking. The term x^{-1} will always be nonzero when it is well-defined. We call the procedure ontological reduction.

Such a deletion of disjuncts from a first-order problem is clearly complete. One can moreover show the following theorem.

Theorem. *Let T be an ontologically correct ForTheL text. Assume that the first-order image of T has a model in which all domain conditions are non-empty. Then its ontological reduction also has a model, i.e., ontological reduction is sound.*

In the proof of the theorem, a model for the ontological reduction is constructed by suitably changing the interpretation of symbols outside of their domain, so that domain guards become superfluous. Ontological correctness ensures that after those changes we still have a model of the original text.

The assumption on the non-emptyness of all domain conditions corresponds to non-emptyness of sorts in many-sorted first-order logic. It will in general not be a problem, since usually mathematical symbols are defined on non-empty domains. In case where a text uses hypotheticals to finally show emptyness of a certain class, possibly problematic domain conditions can be excluded from the reduction process to ensure soundness.

The reduction is computed on a per formula basis without taking into account the rest of the text. While this means that we may miss out on some reduction, we only need to compute the reduction of a formula once and it is ensured that changes to the surrounding text cannot cause unsoundness. In the example above we could clearly reduce the first axiom further to

$$\text{isNonzero}(x * y).$$

But simply adding the line

Signature. 0 is a real number such that $0 * x$ is not nonzero for every real number x.

would introduce a contradiction.

Dealing with typings in a first-order setting is not a new problem. In [1], multiple encodings of (mono- and polymorphic) many-sorted FOL to pure FOL are developed for Sledgehammer in order to encode Isabelle's type system. Encoding by type guards, the approach formerly used by SAD, was among the worst performing. If one applies ontological reduction to a ForTheL text that describes some problem in (monomorphic) many-sorted FOL, then the result is very close to that obtained by the "featherweight guards" encoding described in [1]. However, our approach is more flexible and therefore better suited for our soft type system. For example we can possibly delete predicates of arity higher than one and negated predicates. Furthermore, we can more adequately deal with the complex relations between ForTheL notions.

8 Further plans and discussion

We are following a comprehensive plan for transforming the original small SAD system into a more productive formalization workbench. We want to run SAD in the prover- and LaTeX-friendly Isabelle editor and give more feedback to the user. A better understanding and documentation of the compact Haskell source code of SAD is a challenge essential for the sustainability of the project.

For the definition and implementation of a natural language grammar for SAD we shall work together with linguists like we did in the NaProChe project. The ForTheL language should be further enriched by constructs for algebraic structures and inductive data types. Standard domains like number systems should be formalized in a basic library of texts useful for many purposes. Possibly some aspects like the handling of natural numbers should be taken over into the software, to provide some computational power. Also our term rewriting system has to be improved.

The work so far indicates that the SAD approach has a much wider potential than the beautiful miniatures of the original distribution. Texts can be an order of magnitude longer, and they can check in seconds instead of minutes. These improvements lead to formalizations of comprehensive theories rather than minimal axiom systems for single theorems.

Moreover, texts about various domains can be arranged in interlinked libraries. These texts could be linked by reading-in other texts, or by more sophisticated, truth-preserving import- and export operations. This is similar to the situation in the mathematical literature where a vast collection of texts, which in themselves are locally correct, are linked in various ways by explicit and implicit references.

This suggests an approach to formal mathematics by collections of proof-checked CLM-like texts. The $\sqrt{p}$-example could be linked to an introductory text about natural and rational numbers, which again could be linked to some set-theoretic definition of natural numbers. Such combinations might need some adjustments like unifications of notations (the other text might use the word "integer" instead of "natural number"), or it might require some logical bridging between conclusions of one text and premisses of the other.

9 A few more examples

9.1 Newman's lemma

Our first example is the wellknown Newman's Lemma about the confluence of rewriting systems. The text takes a high-level perspective where a rewriting system is a basic notion with a small number of axiomatic properties. The proof is basically one of Paskevich's examples, but presented here in LaTeX-typesetting. Whereas the original SAD took 48.36 seconds[1] for the check, the new system only takes 0.31 seconds. Notice that for a relation like $\sim$, we can use $a \sim b \sim c$ to conveniently express $a \sim b \wedge b \sim c$ as well as $a, b \sim c$ and $a \sim b, c$ to express $a \sim c \wedge b \sim c$ respectively $a \sim b \wedge a \sim c$.

[element/-s] [system/-s] [reduct/-s]

Signature. An element is a notion.

Signature. A rewriting system is a notion.

Let $a, b, c, d, u, v, w, x, y, z$ denote elements. Let R denote a rewriting system.

Signature. A reduct of x in R is an element.

Let $x \to_R y$ stand for y is a reduct of x in R.

Signature. $x \to_R^+ y$ is a relation.

Definition. $x \to_R^+ y \iff x \to_R y \vee \exists z : x \to_R z \to_R^+ y$.

Axiom. $x \to_R^+ y \to_R^+ z \implies x \to_R^+ z$.

Definition. $x \to_R^* y \iff x = y \vee x \to_R^+ y$.

Lemma. $x \to_R^* y \to_R^* z \implies x \to_R^* z$.

Definition. R is confluent iff for all a, b, c such that $a \to *_R b, c$ there exists d such that $b, c \to_R^* d$.

Definition. R is locally confluent iff for all a, b, c such that $a \to b, c$ there exists d such that $b, c \to_R^* d$.

Definition. R is terminating iff for all a, b $a \to_R^+ b \implies b \prec\!\!\prec a$.

Definition. A normal form of x in R is an element y such that $x \to_R^* y$ and y has no reducts in R.

Lemma. Let R be a terminating rewriting system. Every element x has a normal form in R.
Proof by induction. Obvious.

Lemma (Newman). Any locally confluent terminating rewriting system is confluent.
Proof. Let R be locally confluent and terminating.
We can show by induction that for all a, b, c such that $a \to_R^* b, c$ there exists d such that $b, c \to_R^* d$.

Assume a $\to_R^+ b, c$.

Take u such that $a \to_R u \to_R^* b$.
Take v such that $a \to_R v \to *c$.
Take w such that $u, v \to_R^* w$.
Take a normal form d of w in R.

$b \to_R^* d$. Indeed take x such that $b, d \to_R^* x$.
$c \to_R^* d$. Indeed take y such that $c, d \to_R^* y$. end.
qed.

[1] all results were obtained on an Intel i5-7200U 2.50 GHz in single-threaded execution

9.2 The maximum principle from complex analysis

The following example proves an important theorem of complex analysis from axiomatic assumptions on "high-level" notion like complex numbers and holomorphic functions. The text is axiomatically self-contained and is checked by the improved SAD in 0.33 seconds which seems only possible by ontological reduction.

The text may be viewed as a subsection within a collection of texts which construct or define the notions, which are axiomatized in the beginning.

[number/-s]
Let the domain of f stand for $\mathrm{Dom}(f)$. Let z is in M stand for z is an element of M. Let M contains z stand for z is in M. Let $z << M$ stand for z is in M.

Let f denote a function. Let M denote a set.

Set theoretic notions

Definition. A subset of M is a set N such that every element of N is an element of M. Let $N \subseteq M$ stand for N is a subet of M.

Definition. Assume M is a subset of the domain of f.

$$f^*[M] = \{f(x) \mid x \in M\}.$$

Fix the universe of discourse

Signature. f is holomorphic is an atom.

Signature. A complex number is a notion. Let z, w denote complex numbers.

Only complex functions.
Axiom. Every element of $\mathrm{Dom}(f)$ is a complex number and for every element z of $\mathrm{Dom}(f)$ $f(z)$ is a complex number.

Only complex sets.
Axiom. Every element of M is a complex number.

Signature. A real number is a notion. Let x, y denote real numbers.

Signature. x is positive is an atom. Let ε, δ denote positive real numbers.

Signature. $x < y$ is an atom. Let $x \leq y$ stand for $x = y$ or $x < y$.

Signature. $|z|$ is a real number.

Signature. $B_\varepsilon(z)$ is a set that contains z.

Axiom. $|w| < |z|$ for some element w of $B_\varepsilon(z)$.

Definition. M is open iff for every element z of M there exists ε such that $B_\varepsilon(z) \subseteq M$.

Axiom. $B_\varepsilon(z)$ is open.

Signature. A region is an open set.

Predicates for functions

Definition. A local maximal point of f is an element z of the domain of f such that there exists ε such that $B_\varepsilon(z) \subseteq \mathrm{Dom}(f)$ and $|f(w)| \leq |f(z)|$ for every element w of $B_\varepsilon(z)$.

Definition. Assume $U \subseteq \mathrm{Dom}(f)$. f is constant on U iff there exists z such that $f(w) = z$ for every element w of U. Let f is constant stand for f is constant on the domain of f.

Fundamental theorems on holomorphic functions

Axiom (Open_Mapping_Theorem). Assume f is holomorphic and $B_\varepsilon(z)$ is a subset of the domain of f. If f is not constant on $B_\varepsilon(z)$ then $f^*[B_\varepsilon(z)]$ is open.

Axiom (Identity_Theorem). Assume f is holomorphic and the domain of f is a region. Assume $B_\varepsilon(z) \subseteq \mathrm{Dom}(f)$. If f is constant on $\mathrm{Dom}(f)$ then f is constant.

Theorem (Maximum_Principle). Assume f is holomorphic and the domain of f is a region. If f has a local maximal point, then f is constant.
Proof. Let z be a local maximal point of f. Take ε such that $B_\varepsilon(z) \subseteq \mathrm{Dom}(f)$ and $|f(w)| \leq |f(z)|$ for every element w of $B_\varepsilon(z)$. Let us show that f is constant on $B_\varepsilon(z)$. proof.
Assume the contrary. Then $f^*[B_\varepsilon(z)]$ is open. We can take δ such that $B_\delta(f(z)) \subseteq f^*[B_\varepsilon(z)]$. Therefore there exists an element w of $B_\varepsilon(z)$ such that $|f(z)| < |f(w)|$. Contradiction. end.
Hence f is constant. **qed.**

9.3 A logic puzzle

We conclude with the statement and the solution of the Dreadbury Mansion puzzle by L. Schubert which was once considered a benchmark problem for automated theorem proving. Whereas standard theorem provers require a translation into symbolic language (a standard task in logic exercise classes) SAD accepts controlled natural language statements as formalization. It would be straightforward to modify SAD so that the problem reads even more like natural text. E.g., we have a version, in which all Axioms are written consecutively as one big Axiom. Either way, the solution is being checked in 0.01 seconds, due to the power of eprover.

[live/-s][herself/themselves][person/-s]

The Signature commands introduce the language of the puzzle.

Signature. A person is a notion. Let P, Q stand for persons.

Signature. P lives in Dreadbury mansion is an atom.

Signature. P killed Q is an atom. Let P killed themselves stand for P killed P.

Signature. P hates Q is an atom. Let Q is hated by P stand for P hates Q.

Signature. Aunt Agatha is a person.

Signature. Charles is a person.

Signature. The butler is a person.

The Axioms describe the situation in Dreadbury mansion.

Axiom. Aunt Agatha, Charles and the butler live in Dreadbury mansion.

Axiom. If P lives in Dreadbury mansion then P is Aunt Agatha or P is Charles or P is the butler.

Axiom. Some person that lives in Dreadbury mansion killed Aunt Agatha.

Axiom. If P killed Q then P hates Q and P is not richer than Q.

Axiom. Charles hates no person that is hated by Aunt Agatha.

Axiom. Aunt Agatha hates every person that is not the butler.

Axiom. The butler hates every person that is not richer than Aunt Agatha.

Axiom. The butler hates every person that is hated by Aunt Agatha.

Axiom. No person that lives in Dreadbury mansion hates every person that lives in Dreadbury man-

sion.

Axiom. Aunt Agatha is not the butler.

The obvious question is: who killed Aunt Agatha?
The solution is stated in the theorem, and SAD (with eprover) is able to
automatically prove the theorem.

Theorem. Aunt Agatha killed herself.

References

[1] Jasmin Christian Blanchette, Sascha Böhme, Andrei Popescu & Nicholas Smallbone (2013): *Encoding monomorphic and polymorphic types*. In: International Conference on Tools and Algorithms for the Construction and Analysis of Systems, Springer, pp. 493–507.

[2] Norbert E Fuchs & Rolf Schwitter (1996): *Attempto controlled english (ace)*. arXiv preprint cmp-lg/9603003.

[3] Mohan Ganesalingam (2010): *The language of mathematics*. Ph.D. thesis, Springer.

[4] V Mo Glushkov (1970): *Some problems in the theories of automata and artificial intelligence*. Cybernetics 6(2), pp. 17–27.

[5] Daniel Kühlwein, Marcos Cramer, Peter Koepke & Bernhard Schröder (2009): *The naproche system*. Intelligent Computer Mathematics, Springer LNCS, ISBN 978, pp. 3–642.

[6] Alexander Lyaletski, Konstantin Verchinine & Andriy Paskevych: *SAD wep page*. Available at `http://nevidal.org/sad.en.html`. Accessed: 2018-04-08.

[7] Andriy Paskevych (2007): *Méthodes de formalisation des connaissances et des raisonnements mathématiques: aspects appliqués et théoriques*. Ph.D. thesis, Université Paris 12. In French.

[8] Aarne Ranta (1993): *Type theory and the informal language of mathematics*. In: International Workshop on Types for Proofs and Programs, Springer, pp. 352–365.

[9] Freek Wiedijk (2007): *The QED manifesto revisited*. Studies in Logic, Grammar and Rhetoric 10(23), pp. 121–133.