Formalizing Foundational Notions in Naproche-SAD

Peter Koepke, Jan Penquitt, Marcel Schütz, Erik Sturzenhecker

Mathematical Institute, University of Bonn, Germany

Abstract. We present three current formalization projects with the proof assistant Naproche-SAD. Naproche-SAD formalizations use the natural mathematical input language ForTheL and in favorable cases read like textbook material. In this paper we emphasize the encoding of basic notions and axioms on the basis of inbuilt or slightly modified mechanisms of Naproche-SAD. The formalizations concern an introduction into set theory up to substantial results in infinitary combinatorics, a general theory of structures which are all made pairwise disjoint to avoid ambiguities, and an introduction to abstract linear algebra where for efficiency reasons the type system is restricted to a single type "object", which is hard-coded into Naproche-SAD.

Formalizing mathematics requires a foundational axiomatic system and definitions of common notions like relations, functions, numbers etc. within that system. The choice of initial formalizations is decisive for the development of a theory, determining its look and feel, its applicability and also its efficiency in implementations.

The Naproche-SAD system [2] emphasizes the naturalness of accepted proof texts. Naproche-SAD is a combination of the System for Automated Deduction (SAD) [7] with the Naproche approach for Natural Proof Checking [6]. Text are written in the language ForTheL which employs natural language categories like noun, verb, or adjective to capture mathematical content. We have chapter-sized formalizations from various areas which in favorable cases read like textbook mathematics [3]. Presently, Naproche-SAD translates ForTheL into first-order logic which is checked by general purpose ATPs like Eprover. We are also experimenting with translations to type theory and Lean.

Up to now ForTheL formalizations typically encompass ad-hoc formalizations of basic notions needed in the particular text. We have started to write ForTheL libraries for basic mathematical notions. We encourage a pluralistic approach in order to gain experiences which may lead to several foundational libraries on which specific formalizations can be based.

The language ForTheL was conceived with the idea that there is a common mathematical language specific to (modern) mathematics, which is not committed to particular foundations. The language contains simple "linguistic" notions which have generally agreed mathematical properties. There are, e.g., rudimentary mechanisms for functions: if f is a function and x is a valid argument, then one can form the term for the value $f(x)$. But whether functions are atomic notions or constructed as sets of ordered pairs, or whether domains of functions are sets or proper classes is up to further specifications. In set theory all mathematical objects including functions should be sets; in category theory, functions, or morphisms, are primary objects, but they should not only have a domain, but also a codomain. Such particulars must be specified by axioms.

The Naproche-SAD system has some of those notions wired in. A function is an inbuilt notion and the application $f(x)$ of a function to an argument is an inbuilt term constructor. Functions can be introduced as λ-terms with associated mechanisms. Abstraction terms like $\{x \mid \phi(x)\}$ are also provided.

ForTheL texts then postulate further axioms about inbuilt and new notions and thus extend the "linguistic" theory to some (standard) mathematical theory. Eventually such axiomatizations will form the bases of libraries. We are in a prototypical phase with some peculiarities which will be sorted out in future releases. E.g., the formalization of set theory by Jan Penquitt employs the inbuilt notion of "set" which are rather classes from the perspective of the standard Zermelo-Fraenkel (ZF) set theory. The formalization thus introduces an ad-hoc notion of "zfset" for sets in the sense of the theory ZF.

The two other formalizations also have to deal with initial ontologies of sets and functions. They then approach the concept of first-order structures in markedly different ways. Marcel Schütz's formalization makes the domains of distinct structures pairwise disjoint, so that every element of a structure is an element of a unique structure. The addition in a term like $x + y$ is uniquely determined as the addition in that structure which has x and y as elements. On the other hand, formalizations by Erik Sturzenhecker first viewed structures in a more conventional way as sets together with the interpretations of relation and function symbols on the underlying set. This approach is very inefficient due to a large number of proof obligations to show that elements lie in the right structure so that the operations and relations of the structure can be applied. This problem could be overcome by flattening the various types down to a single type "object".

Formalizing Set Theory (Jan Penquitt)

In my master thesis I work with Naproche-SAD to formalize classic set theory. My work is separated in two parts:

The first part is developing Naproche. Naproche can only understand those words and phrases which are integrated in the system or which are introduced in the proof text. To make work easier some notions such as sets, elements or functions are already integrated and satisfy some elementary properties. These properties are for example that a function has a domain, which is a set, such that to each element of this set the function assigns a value. In most cases these notions do not cause any problems, but it some fields of mathematics these notions must satisfy certain conditions, otherwise they will lead to contradictions. The classical example is Russell's Paradox, so in set theory it is important to distinguish between sets and proper classes whereas in natural language classes and sets can often be used synonymously. Both notions indicate mathematical objects which are a collection of other objects. But set theoretically these terms denote different kinds of objects, where sets are only “small” classes.

Similar problems occur if we consider the integrated functions. A function in Naproche is just an object with a domain which applies values to elements of its domain. These values can be arbitrary objects, but if we want to have set theoretic functions it is important that these values are sets, and not proper classes. So the aim is to solve problems with the integrated notions so that it is possible to formalize and proof more complicated statements in set theory or other fields of mathematics without leading towards those kind of contradictions.

The second part of my thesis is to use Naproche to formalize Zermelo-Fraenkel set theory. First, I introduced new notions such as zfsets and zffunctions which denote the actual sets and functions in terms of set theory. Some classes are zfsets, but not all classes are. The axioms of Zermelo and Fraenkel then state rules how to get new zfsets out of given zfsets. Similarly, a zffunction denotes a function with additional properties such that we have actual set theoretic functions when talking about zffunctions. This means that the value of an element of the domain is a zfset and not an arbitrary class.

With these new notions we can now differentiate between sets (zfsets) and classes (sets):

```
Signature. A zfset is a notion.
Axiom. Let x be a zfset. Then x is a set.
Axiom. Let a be a set. Let b be an object. Let b ∈ a. Then b is a
zfset.

Definition Emptyset. The empty set is {zfset x | x ≠ x}.
Let ∅ stand for the empty set.
Definition Universe. The universe is {zfset x | x = x}.
Let 𝕍 stand for the universe.

# ZF-Axioms
# [some definitions such as ℙ(x) or ⋃x are left out for brevity]

Axiom Emptyset. ∅ ∈ 𝕍.
Axiom Pair. Let x, y ∈ 𝕍. Then {x, y} ∈ 𝕍.
Axiom Union. Let x ∈ 𝕍. Then ⋃x ∈ 𝕍.
Axiom Separation. Let x ∈ 𝕍. Let a be an object. Let a ⊂ x. Then
a ∈ 𝕍.
Axiom Powerset. Let x ∈ 𝕍. Then ℙ(x) ∈ 𝕍.
Axiom Foundation. Let A be a set. Let A ≠ ∅. Then there exists
b ∈ 𝕍 such that (b ∈ A ∧ forall c ∈ 𝕍 (c ∈ b ⇒ c ∉ A)).

Lemma. Forall x (x ∉ x).
Proof by contradiction. Assume the contrary.
    Take a zfset x such that x ∈ x.
    Then {x, x} ∈ 𝕍.
    {x, x} ≠ ∅.
    Forall $y \in \{x,x\}$ ($y = x$).
    Then forall $c \in \mathbb{V}$ ($c \in x \Rightarrow c \notin \{x,x\}$) (by Foundation).
    Contradiction.
qed.

Lemma. $\mathbb{V}$ is not a zfset.
Proof by contradiction. Assume the contrary.
    Then $\mathbb{V} \in \mathbb{V}$.
    Contradiction.
qed.
```

Starting with these axioms I can straightforwardly develop classic set theory. I started with defining ordinals, cardinals and natural numbers, and every proof is only based on the Zermelo-Fraenkel-Axioms and previously proved facts:

```

Definition transitive. Let $A$ be a set. $A$ is transitive iff
forall $x,y \in \mathbb{V}$ ($y \in A \wedge x \in y \Rightarrow x \in A$).
Let Trans($A$) stand for $A$ is transitive.

Lemma. Trans($\emptyset$).

Signature. An ordinal is a notion.
Axiom. Let $\alpha$ be an ordinal. Then $\alpha$ is a zfset.
Axiom. Let $\alpha$ be a zfset. Then $\alpha$ is an ordinal iff (Trans($\alpha$) $\wedge$
forall $y \in \alpha$ $\mathrm{Trans}(y)$).

Definition Ord. The class of ordinals is
{zfset $x$ | $x$ is an ordinal}.
Let Ord stand for the class of ordinals.

Lemma. $\mathrm{Ord} \notin \mathbb{V}$.
Proof by contradiction. Assume the contrary.
    Then $\mathrm{Ord} \in \mathbb{V}$.
    Trans(Ord).
    Proof.
        Let $\alpha \in \mathrm{Ord}$.
        Let $x \in \alpha$.
        Then $x \in \mathbb{V}$.
        Trans($x$).
        Then $x \in \mathrm{Ord}$.
    end.
    Forall $\alpha \in \mathrm{Ord}$ Trans($\alpha$).
    Then $\mathrm{Ord} \in \mathrm{Ord}$.
    Contradiction.
qed.
```

I went on by defining the cardinality of a set and proved some basic intuitive facts about cardinalities.

Also more complicated results can be proven with Naproche just as in an ordinary math lecture. With the Alef hierarchy of cardinals, the Mostowski Collapse for strongly wellfounded relations and the Gödel pairing for pairs of ordinals I proved that for infinite cardinals κ we have the identity $\kappa \cdot \kappa = \kappa$ (here the cardinal product was defined as the cardinality of the cartesian product of these two cardinals and a cardinal is identified with the corresponding ordinal, which then is regarded as the set containing all smaller ordinals).
The proof is a generalization of the proof that the set of rational numbers $\mathbb{Q}$ is countable. The following is a small extract of the proof. For a complete formal proof many notions and facts have to be defined and proven beforehand, for the sake of brevity these are black boxed; a complete formalization which is checked by Naproche-SAD can be found under [8].

```
# Goedel Ordering

Definition. Let $a_1, a_2, b_1, b_2 \in \mathsf{Ord}$. $(a_1, a_2) <_G (b_1, b_2)$ iff
   $(a_1 \cup b_1 \in a_2 \cup b_2)$
$\vee$  $(a_1 \cup b_1 = a_2 \cup b_2 \wedge a_1 \in a_2)$
$\vee$  $(a_1 \cup b_1 = a_2 \cup b_2 \wedge a_1 = a_2 \wedge b_1 \in b_2)$

Signature. goedel is a relation.

Axiom. field(goedel) = Ord × Ord ∧ forall $a_1, a_2, b_1, b_2 \in \mathsf{Ord}$
$((a_1, a_2), (b_1, b_2))$ ∈ goedel iff $(a_1, a_2) <_G (b_1, b_2)$.

Lemma. goedel is a strongly wellfounded relation and
Dom(R) = Ord × Ord.
Proof. [prove off] qed.

Definition. The Goedel-Pairing is the Mostowski Collapse of
goedel.
Let G stand for the Goedel-Pairing.


# The Proof of $\kappa \cdot \kappa = \kappa$

Lemma. Forall $\alpha \in$ Ord G^$[\aleph_\alpha \times \aleph_\alpha] = \aleph_\alpha$.
Proof.

# This is a long and tedious proof.
# The $\aleph$-Hierarchy is a numbering of all infinite cardinals.
# We define $\aleph_0 = \mathsf{Card}(\mathbb{N})$ and then order every infinite cardinal by
 size such that we have
# $\aleph_0 < \aleph_1 < \aleph_2 < \ldots < \aleph_\alpha < \ldots$ for all ordinals $\alpha$.
# Then we prove the lemma by first showing G^$[\aleph_\alpha \times \aleph_\alpha] \in \mathsf{Ord}$
# and then by induction on $\alpha$ the inequalities
# G^$[\aleph_\alpha \times \aleph_\alpha] \geq \aleph_\alpha$ and G^$[\aleph_\alpha \times \aleph_\alpha] \leq \aleph_\alpha$.
```

```
[prove off]
qed.


Theorem. For every infinite cardinal $\kappa$ ($\kappa \cdot \kappa = \kappa$).
Proof.
Let $\kappa$ be an infinite cardinal.
Take an ordinal $\alpha$ such that $\kappa = \aleph_\alpha$.
G^$[\aleph_\alpha \times \aleph_\alpha] = \aleph_\alpha$
Then G$\restriction_{\aleph_\alpha \times \aleph_\alpha}$: $\aleph_\alpha \times \aleph_\alpha \leftrightarrow \aleph_\alpha$.
Then Card($\aleph_\alpha \times \aleph_\alpha$) = $\aleph_\alpha$.
$\kappa \cdot \kappa$ = Card($\kappa \times \kappa$).
Then $\kappa \cdot \kappa = \kappa$
qed.
```

Furthermore I proved some facts about the cofinality of infinite cardinals, König's Lemma and in the end the Hausdorff Recursion Rule for cardinal exponentiation.

With the Gimel function I developed some calculation rules for cardinal exponentiation and examined cardinal exponentiation in a model of ZFC + GCH. The next task will be to examine closed unbounded and stationary subsets of uncountable regular cardinals. The end result will be the proof of Silver's Theorem about the value of the continuum function at singular cardinals depending on the values at preceding cardinals.

A natural framework for mathematical structures (Marcel Schütz)

In my bachelor thesis I examine mathematical structures and how they can be handled in ForTheL. I formalized some texts about various structures like metric and topological spaces, ordered sets and categories. My formalizations aim to give a foundation for mathematical structures in a way that we can work with these as we are used to from human-written texts.

Mathematical structures as they appear in literature come along with some problems when we try to implement them in formal languages. There are several situations in which a computer would regard a statement as ambiguous or would even draw conclusions from it which lead to inconsistencies whereas a human reader has no problem at all interpreting it. He simply knows what the author means. Let me give some examples of problems I was confronted with:

(1) We want the structures $(\mathbb{R}, +, <)$ and $(\mathbb{R}, <, +)$ to be the same, but that would contradict the definition of a tuple. Hence the naive approach of defining structures as tuples fails.

(2) Consider the structure $(\mathbb{Z}/2\mathbb{Z}, +)$ where we regard $\mathbb{Z}/2\mathbb{Z} = \{0, 1\}$ as a subset of $\mathbb{R}$. Then $(\mathbb{Z}/2\mathbb{Z}, +)$ is a group and also metric space. So what about $(\mathbb{Z}/2\mathbb{Z}, +) \times (\mathbb{Z}/2\mathbb{Z}, +)$? Is it the group or the metric space over $\mathbb{Z}/2\mathbb{Z} \times \mathbb{Z}/2\mathbb{Z}$

(provided we have defined *the* product metric)? In contrast to a machine a human reader usually knows from the context what is meant.

(3) Let $\emptyset \neq A \subsetneq X$ be sets and $U = \{\emptyset, A, X\}$. Then (X, U) is a topological space. But is U considered as a topology of open sets or of closed sets? Of course we could say that U should always be interpreted as a topology of open sets. But since there are several equivalent definitions of topological spaces in literature we want to allow a coexistence of several such definitions in our formalization, too.

(4) Let $f : G \to H$ be a group homomorphism. Then we want to state expressions like $f(x \cdot y) = f(x) \cdot f(y)$. But usually $\cdot$ is either a function $G \times G \to G$ or a function $H \times H \to H$. So to make the expression $f(x \cdot y) = f(x) \cdot f(y)$ valid, $\cdot$ must be defined on all groups. But this would lead to inconsistencies if we have two elements x, y that lie in two different groups X, Y such that the operation $x \cdot y$ has different results in X and in Y.

These problems can be solved in two steps. First I had to edit the source code of Naproche-SAD in order to allow a more liberal use of the built-in notions like the $\in$-relation. Then I was able to introduce the notion of structures in ForTheL.

I define a structure simply as an injective function of a class. Such a class can be interpreted as the underlying class of the structure. Moreover I state that an object is an element of a structure X iff it is a value of X. Furthermore I state that all structures are disjoint (i.e. no object is an element of two different structures). This matters since we want to write down expressions like "A is an open set" instead "A is a set that is open in X". If there were two structures X and Y such that A is open in X but not in Y the first formulation of the statement would lead to a contradiction.

In ForTheL my axiomatization of structures is as follows:

```
Signature FoundStr000. A structure is an injective function.

Axiom FoundStr002. Let X be a structure. The domain of X is a
class.

Axiom FoundStr005. Let X be a structure and x be an object. x is
 an element of X iff x lies in the range of X.
```

Axiom *FoundStr005* makes it clear why I had to modify the source code of Naproche-SAD. In the original version of Naproche-SAD the $\in$-relation is restricted to classes. Since one cannot derive from our axioms that a structure X is a class, we would get an error if we would write something like $y \in X$ in the original version. Thus I had to edit the source code in order to be able to state expressions like $y \in X$ for arbitrary objects X.

To see how my definition of structures can be applied let us look at my definition of topological spaces.

```
Signature TopTop100. A topological space is a small structure.

Definition TopTop105. Let X be a topological space. The topology
 of X is a class T such that T = {A | A is an open subset of X}.
```

```
Axiom TopTop115. Let X be a topological space. The topology of X
 is a topology on X.
```

First we extend our signature by the notion of topological spaces and state that they are small structures. A small structure is simply a structure whose domain is a set. Then we define the topology of a topological space as the class of its open subsets. Here ”_ is open” is just a predicate without any predefined meaning. We add such a meaning axiomatically by stating that the topology *of* a topological space is a topology *on* that space where a topology *on* an object which can contain elements is a topology in the usual sense.

This definition of topological spaces allows us now to formulate the desired expression ”A is an open set” without referring to a space in which A is open. But we must be careful. Since the empty set is a subset of every structure it is immediately open (provided we can show that there exists a topological space, but more on that later). Hence if we want to define the meaning of the predicate ”_ is open” for instance for subsets of metric spaces we must assure that the empty set is open in any metric space, too. Another disadvantage of the definition of a topological space as a structure is that we must differentiate between a structure and its range (i.e. the class of its elements). Since a structure is not a class, a topological space X is not a subset of itself, hence we cannot infer that X is open. So whenever we need the statement ”X is open” we have to replace it by ”the range of X is open”.

So far we have seen what a topological space is, but at the moment we cannot know whether there exists one. In classical textbooks where topological spaces are defined as pairs (T, U) of a set T and a topology U on T the existence of topological spaces follows directly from the ZFC axioms. In my settings of mathematical structures we have to introduce a new axiom that ensures the existence of topological spaces. The easiest way to do this would be something like the following:

```
Axiom. Let X be a set and U be a topology on X. Let V be a class
. Assume that V = {T[A] | A is an element of U}. Then there is a
 topological space T such that X is the domain of T and V is the
 topology on T.
```

But I wanted to go a step further. In the literature there are many different ways to define topological spaces: They can be defined via open sets, closed sets, neighborhoods, a closure operator, ... So I wanted to model the ability of having several definitions for the same object. I accomplished this by introducing a notion of so-called interpretations. I defined interpretations as pairs of functions (F, f) such that $dom(f) = \bigcup dom(F)$ and $f \upharpoonright X$ is a bijection between X and $F(X)$ for all $X \in dom(F)$. Thus the function F maps an object X to an object Y such that f restricts to a bijection between X and Y.

```
Signature FoundInt000. An interpretation is a notion.
```

```
Axiom FoundInt005. Let F,f be functions. (F,f) is an
interpretation iff dom(f) = \bigcup dom(F) and the restriction
of f to X is a bijection between X and F(X) for all X \in dom(F)
.
```

To illustrate how these interpretations work consider the class TOP of topological spaces. Let us define a function TOP_{open} of TOP and a function Top_{open} of $\bigcup TOP$ as follows:

```
Signature TopTop190. TOP_{open} is a function of TOP.

Axiom TopTop195. Let T be a topological space. Let X be the
domain of T and U be a class such that U = {T^{-1}[A] | A is an
open subset of T}. Then TOP_{open}(T) = (X,U).


Signature TopTop200. Top_{open} is a function of \bigcup TOP.

Axiom TopTop205. Let T be a topological space and x \in T. Top_{
open}(x) = T^{-1}(x).
```

So TOP_{open} maps a topological space to the obvious pair (X, U) of a set and a topology and f maps every element of a topological T space to the corresponding element of $TOP_{open}(T)$. These functions yield an interpretation (TOP_{open}, Top_{open}). Now we can state an axiom that guarantees the existence of topological spaces:

```
Axiom TopTop210. Let X be a set and U be a topology on X. Then
there is a topological space T such that TOP_{open}(T) = (X,U).
```

In analogy to the functions TOP_{open} and Top_{open} we can define functions TOP_{closed} and Top_{closed}:

```
Signature TopTop220. TOP_{closed} is a function of TOP.

Axiom TopTop225. Let T be a topological space. Let X be the
domain of T and U be a class. Assume that U = {T^{-1}[A] | A is
a closed subset of T}. TOP_{closed}(T) = (X,U).


Signature TopTop230. Top_{closed} is a function of \bigcup TOP.

Axiom TopTop235. Let T be a topological space and x \in T. Top_{
closed}(x) = T^{-1}(x).
```

Again $(TOP_{closed}, Top_{closed})$ is an interpretation. So we can interpret any pair (X, U) of a set X and a topology of closed sets U on X as a topological space:

```
Proposition TopTop240. Let X be a set and U be a topology of
closed sets on X. Then there is a topological space T such that
TOP_{closed}(T) = (X,U).
```

The only thing that remains is how to deal with the mentioned equivalent definitions of topological spaces. I defined equivalence of interpretations as follows:

```
Axiom FoundInt015. Let F,f,G,g be functions. Assume that (F,f)
and (G,g) are interpretations. Assume that dom(F) = dom(G). (F,f
) and (G,g) are equivalent iff for all X \in dom(F) and all x \
in X we have f(x) = g(x).
```

We can apply this definition to our two previouly defined interpretations:

```
Proposition TopTop250. (TOP_{closed},Top_{closed}) and (TOP_{
open},Top_{open}) are equivalent interpretations.
```

Let us return to the list of problems that were mentioned at the beginning of this section.

(1) The problem that we want to interpret tuples like $(\mathbb{R}, +, <)$ and $(\mathbb{R}, <, +)$ as the same structures can easily be bypassed for example by defining a type of ordered groups and showing that $(\mathbb{R}, +, <)$ and $(\mathbb{R}, <, +)$ are values of equivalent instantiations of that type.

(2) The problem whether $(\mathbb{Z}/2\mathbb{Z}, +) \times (\mathbb{Z}/2\mathbb{Z}, +)$ should be regarded as a group or a metric space can also be solved by defining a type GRP of groups and a type MET of metric spaces and showing that $(\mathbb{Z}/2\mathbb{Z}, +) \times (\mathbb{Z}/2\mathbb{Z}, +)$ is an instance of both types. So we can ”lift” $(\mathbb{Z}/2\mathbb{Z}, +) \times (\mathbb{Z}/2\mathbb{Z}, +)$ either to GRP or to MET.

(3) The coexistence of several equivalent definitions of structures, especially of topological spaces, should be made explicit enough above.

(4) In my setting statements of the form $f(x \cdot y) = f(x) \cdot f(y)$ for a homomorphism $f : G \to H$ cause no problems since G and H are both structures and thus an object cannot belong both to G and H. Hence the operation $\cdot$ yields a unique value for all pairs (x, y) of elements of some group.

The interplay of several structures or even several categories of structures is possibly the most interesting application of my definition of structures. Unfortunately I have not formalized many texts that deal with different structures yet, so it is hard to predict how well my approach can be applied to those settings.

Another kind of objects I have not examined yet are substructures. Since I regard structures as disjoint I cannot define a substructure of a structure X as a structure Y such that every element of Y lies in X. So maybe substructures should be realized as injective maps $Y \to X$ between structures in analogy to subobjects in category theory.

So there is still need of expanding my formalizations until they serve as a suitable foundation for mathematical structures. However by now my formalizations allow quite natural formulations of statements that are very close to those in textbooks. Compare for example Bredon's definition of an ε-ball in a metric space in [1] with my formalization. Bredon defines it as follows:

In a metric space X we define the "ϵ-ball", $\epsilon > 0$, about a point $x \in X$ to be

$$B_\epsilon(x) = \{y \in X \mid \operatorname{dist}(x, y) < \epsilon\}.$$

Whereas my formalization is the following:

```
Axiom MetOs000. Let X be a metric space and x \in X and epsilon
be a positive real number.
B(x,epsilon) is a class such that

  B(x,epsilon) = {y in X | dist(x,y) < epsilon}.
```

If we ignore the fact that I explicitly have to state that `B(x,epsilon)` is a class (unfortunately Naproche-SAD does not automatically regard a class term as a class) the two definitions do not really differ. With the background of my definition of metric spaces as structures we know that the ε-ball is well-defined in my formalization.

In the end of this section let us look at an example that deals with the interplay of two different structures. We can define an interpretation that maps every metric space M to the topological space whose topology is induced by the metric of M. Let us call this interpretation (MET_{top}, Met_{top}). For better readability we abbreviate $MET_{top}(M)$ with M_{TOP} and $Met_{top}(x)$ with x_{TOP}. Now we can give a short definition of metrizable topological spaces:

```
Definition TopMet055. Let X be a topological space. X is
metrizable iff there is a metric space M such that X = M_{TOP}.
```

We can for instance prove that any metrizable topological space is Hausdorff which can be used in the following proof.

```
Proposition TopMet076. Let M be a metric space and x be an
element of M. Then `{x}` is closed.

Proof.
Take a topological space T such that T = M_{TOP}. Take an
element y of T such that y = x_{TOP}. T is Hausdorff. Hence T
satisfies T1. Thus `{y}` is closed. Therefore `{x}` is closed.
qed.
```

In this proof we could easily use facts from the realm of topological spaces and apply them to metric spaces by an interpretation. Surely in hand-written proofs one would directly identify a metrizable topological space with its corresponding metric space such that there is no need for these technical functions like MET_{top}. So switching between different structures becomes a bit unhandy in my framework. Whereas when we want to work with only one type of structures, for example in "pure" group theory or "pure" topology, then the fact that structures can be regarded as disjoint in my setting allows fairly natural formalizations. Examples of such formalizations can be found at [9]. They range from order and group theory to metric and topological spaces and category theory. A modified version of Naproche-SAD that provides all changes of the source code that are necessary to make Naproche-SAD accept my formalizations can be found at [10].

Linear Algebra in Naproche-SAD vs. Lean (Erik Sturzenhecker)

In the course of my bachelor thesis I am writing a collection of ForTheL texts to be checked by Naproche-SAD. The natural language approach of ForTheL and Naproche-SAD contrasts with formalizations in theorem provers like Lean, which are more reminiscent of programming code.

While I am currently formalizing parts of representation theory of algebras, those texts require some basics from linear algebra. Thus, I am building upon the linear algebra formalizations I co-wrote in a practical project at the University of Bonn. In that project we took a Lean formalization by Kenny Lau [4] and covered the same topics, definitions and theorems in a ForTheL text that can be checked by Naproche-SAD. Our particular approach came with massive performance issues, which I was now able to fix by optimizing the formalization of algebraic structures.

We can now compare the original Lean file to the ForTheL versions regarding the style of writing, the formal approach to algebraic structures and the performance of the two systems. Building from the ground up, the topics that are covered in both formalizations comprise fields, vector spaces, subspaces, linear maps, the embedding of a vector space into its double dual space, the endomorphism ring and the automorphim group of a vector space.

The Lean file uses this definition of abelian groups from the Lean mathlib [5]:

```
class has_zero (α : Type u) := (zero : α)
class has_add  (α : Type u) := (add : α → α → α)
class has_neg  (α : Type u) := (neg : α → α)

class add_semigroup (α : Type u) extends has_add α :=
(add_assoc : ∀ a b c : α, a + b + c = a + (b + c))

class add_comm_semigroup (α : Type u) extends add_semigroup α :=
(add_comm : ∀ a b : α, a + b = b + a)

class add_monoid (α : Type u)
extends add_semigroup α, has_zero α :=
(zero_add : ∀ a : α, 0 + a = a) (add_zero : ∀ a : α, a + 0 = a)

class add_comm_monoid (α : Type u)
extends add_monoid α, add_comm_semigroup α

class add_group (α : Type u) extends add_monoid α, has_neg α :=
(add_left_neg : ∀ a : α, -a + a = 0)
```

```
class add_comm_group (α : Type u)
extends add_group α, add_comm_monoid α
```

Similarly to this, we initially formalized algebraic structures S in ForTheL by demanding them to have functions like add{S} and neg{S} with certain properties. This approach resembles the notion of a structure in first-order logic: A structure is the interpretation of (a subset of) the language lang, consisting of a set (if |S| is defined to be a set), constants and functions (if zero{S}, add{S}, etc. are defined to be suchlike). All of these are always indexed with the name of the respective structure, because multiple structures shall have the same notation, but unlike Lean, Naproche-SAD does not support implicit arguments.

```
Signature. lang is a set.
Axiom. lang = {carr,zero,one,add,mul,neg,inv,smul}.
Definition. A structure is a function S such that Dom(S) is a
subset of lang.

Let |S| stand for S[carr].
Let 0{S} stand for S[zero].
Let add{S} stand for S[add].
Let neg{S} stand for S[neg].

Let a +{S} b stand for add{S}[(a,b)].
Let ~{S} a stand for neg{S}[a].
Let a -{S} b stand for add{S}[(a, ~{S} b)].
Let a < S stand for a ∈ |S|.
Let (S has a,b,c,d) stand for (a,b,c,d ∈ Dom(S)).

Definition. Prod(A,B) = {(x,y) | x ∈ A and y ∈ B}.

Definition. An abelian group is a structure G such that
     G has carr,zero,add,neg
 and |G| is a set
 and 0{G} < G
 and add{G} is a function from Prod(|G|,|G|) to |G|
 and neg{G} is a function from |G| to |G|
 and for all a < G    :       a +{G} 0{G} = a
 and for all a < G    :          a -{G} a = 0{G}
 and for all a,b,c < G: a +{G} (b +{G} c) = (a +{G} b) +{G} c
 and for all a,b < G  :          a +{G} b = b +{G} a.
```

However, using the built-in notions of sets, functions and ordered pairs in this way slowed down the ontological text checking significantly. The proofs became very long, because the system needed many unnatural reminders like (v,w) ∈ Prod(|V|,|V|) = Dom(add{V}) to be able to verify ontological correctness of algebraic expressions.

The approach shown below makes for a way better performance of the checking process: Terms like v +{V} w are now always defined and ontological checking of any algebraic term is avoided. Everything in Naproche-SAD is an `object`, so the following four signatures are just translated as *True* and only introduce the respective terms. Algebraic structures then need to satisfy certain closure properties.

```
Signature. Let S be an object.   |S| is an object.
Signature. Let S be an object.   0{S} is an object.
Signature. Let S,a,b be objects. a +{S} b is an object.
Signature. Let S,a be objects.   ~{S} a is an object.
Let a -{S} b stand for a +{S} (~{S} b).
Let a < S stand for a ∈ |S|.

Definition. An abelian group is an object G such that
     |G| is a set
 and 0{G} < G
 and for all a,b < G  : a +{G} b < G
 and for all a < G    :   ~{G} a < G
 and for all a < G    :       a +{G} 0{G} = a
 and for all a < G    :          a -{G} a = 0{G}
 and for all a,b,c < G: a +{G} (b +{G} c) = (a +{G} b) +{G} c
 and for all a,b < G  :          a +{G} b = b +{G} a.
```

Even better performance can be achieved by completely avoiding the built-in ∈-relation which otherwise invokes a check whether the object on the right is a set. Replacing `functions` by a new notion of `maps` allows terms like `f(x)` without checking if `x` lies in the domain of `f`. Both is done in the following:

```
Signature. Let A be an object. A member of A is a notion.
Let x << A stand for x is a member of A.
Axiom. Let A be a set. Let x be an object. x << A iff x ∈ A.

Signature. Let f,x be objects. f(x) is an object.
Signature. Domain is an object.
Signature. A map is a notion.
Axiom MapExt. Let f,g be maps. If Domain(f) = Domain(g)
 and (for all x << Domain (f) : f(x) = g(x)) then f = g.
```

This approach somehow diverges from the philosophy of Naproche-SAD and ForTheL, which encourage more strictly typed expressions. However, we don't expect this avoidance of type checking to lead to logical contradictons: As long as every axiom, definition and theorem makes the necessary assumptions about the occuring objects (”`Let V and W be vector spaces over K`”), the ATP can only use them to prove ontologically correct statements. In this sense, the task of ontological checking is outsourced from Naproche-SAD to the ATP.

While the original Lean formalization consists of about 500 lines of code, the final ForTheL version takes 850 lines for the same mathematical content. This difference is mostly due to much more algebraic detail needed in the proofs. The Lean file is checked in about 20 seconds, while Naproche-SAD takes about 4 minutes. We see that in this case the performance of Naproche-SAD comes somewhat close to that of Lean, while the ForTheL texts are much more readable for most mathematicians. The final product resembles common texts in linear algebra, but comparing the writing process itself to natural mathematical writing would be quite a stretch since it takes some experience as well as trial and error to create formalizations that can be checked by Naproche-SAD.

My entire project can be found under [11].

References

1. Bredon, G.E.: Topology and Geometry. Springer (1993)
2. Frerix, S., Wenzel, M., Koepke, P.: Isabelle/Naproche (2019), https://sketis.net/2019/isabelle-naproche-for-automatic-proof-checking-of-ordinary-mathematical-texts
3. Koepke, P.: Textbook Mathematics in the Naproche-SAD System. In: CICM Informal Proceedings (2019), http://cl-informatik.uibk.ac.at/cek/cicm-wip-tentative/FMM4.pdf
4. Lau, K.: Linear Algebra in Lean, https://github.com/kckennylau/Lean/blob/master/linear_algebra/vector_space.lean
5. Lean community: The Lean mathematical library, https://github.com/leanprover-community/mathlib
6. Naproche-SAD in Isabelle-jEdit, https://files.sketis.net/Isabelle_Naproche-20190611/
7. Paskevich, A.: Méthodes de formalisation des connaissances et des raisonnements mathématiques: aspects appliqués et théoriques. Ph.D. thesis, Université Paris 12 (2007)
8. Penquitt, J.: Set Theory in ForTheL, https://github.com/naproche-community/FLib/tree/master/SetTheory
9. Schütz, M.: Mathematical structures in ForTheL, https://github.com/naproche-community/FLib/tree/master/Structures
10. Steffen Frerix, Makarius Wenzel, P.K.M.S.: A modified version of naproche-sad, https://github.com/McEarl/Naproche-SAD
11. Sturzenhecker, E.: Representation Theory in ForTheL, https://github.com/naproche-community/FLib/tree/master/RepresentationTheory