Formal Representation of Mathematics in a Dependently Typed Set Theory

Feryal Fulya Horozal and Chad E. Brown

Universität des Saarlandes, Saarbrücken, Germany
{fulya,cebrown}@ags.uni-sb.de

Abstract. We have formalized material from an introductory real analysis textbook in the proof assistant Scunak. Scunak is a system based on set theory encoded in a dependent type theory. We use the formalized material to illustrate some interesting aspects of the relationship between informal presentations of mathematics and their formal representation. We focus especially on a representative example proved using the system.

1 Introduction

In recent decades, a large amount of mathematics has been formalized in different logical systems using various computer programs. Still, the mathematics that has been formally represented and verified using computers is a tiny percent of the mathematics that has been informally written and published as books and papers. If this informally presented mathematical knowledge is to be transformed into formal versions in mechanized systems, then we must better understand the relationships between the two versions. In order to study the gap between informal and formal representations, we have formalized some material from Bartle and Sherbert's introductory textbook on real analysis [2]. This particular textbook has been studied in the context of formalized mathematics already. In particular, a linguistic analysis of portions of [2] is given in [3]. Likewise, an example from the first chapter of [2] is considered in [1] and [8]. We formalized the material in Scunak [5,6], a system based on set theory encoded into a dependent type theory.

In the next section, we will introduce the type theory of Scunak. In Section 3, we discuss the formalization of material from the textbook [2]. In Section 4, we focus on one small example from [2] (the limit of the sequence $\frac{1}{\sqrt{n}}$ is 0). We have encoded this example in ISABELLE-HOL [12] and MIZAR [14], and we briefly compare the encodings.

2 Preliminaries

We present the mathematical proof assistant Scunak [5,6]. Scunak is based on set theory formalized in a logical framework with dependent types and proof terms. The system is relatively new and has been under development since 2005.

Scunak offers several functionalities to its users.

– It provides an environment in which one can formalize mathematics from a set-theoretical foundation.
– Like several other proof assistants such as Coq [4], Scunak allows its users to interactively construct proofs (technically "proof terms") using the Scunak Interactive Prover (Scip).
– Scunak can be used as a tutor (Scutor, for a demonstration see [7]) that gives feedback to a user on his proof attempts in an arbitrary state of the proof.
– Scunak can be used for verifying textbook proofs through a process of translating the LaTeX representation of informal proofs into a proof term that can be understood and checked by Scunak. A detailed description of the process is given in [8].

In this paper, we focus on the relationship between informal mathematical texts and the formal versions in Scunak. Functionalities of Scunak like Scutor and Scunak's verification component for textbook proofs are beyond the scope of this paper.

2.1 The Scunak Type Theory

The type theory of Scunak is dependent type theory with proof irrelevance. A general frame in which proof irrelevance is discussed can be found in [13]. We briefly give the syntax for the type theory of Scunak.

We assume a countably infinite set of variables and use x to range over this set. In addition to variables, we also assume a countably infinite set of names and use c to range over this set. Names will be used to declare constants, abbreviations and claims in a signature.

The set of terms and types are given inductively as follows:

$$\begin{array}{ll} \textbf{Terms}\ p, r, s, t, \phi, \rho \ldots := x|c|(\lambda x.s)|(s\,t)|\langle s, \rho\rangle|\pi_1(t)|\pi_2(t) \\ \textbf{Types}\ S, T, S_1, T_1 \ldots \ := \mathbf{obj}|\mathbf{prop}|(\mathbf{pf}\,p)|(\mathbf{class}\,\phi)|(\Pi x : S.\,T) \end{array}$$

We often omit parenthesis if it is clear in context where they are missing. A context Γ is an ordered list of variables associated with types. We sometimes speak of a term t having a type T (in a context Γ), and we write $t : T$ (or $\Gamma \vdash t : T$). For the definitions of these notions see [5,9].

Often, we will be discussing a particular object of mathematical discourse such as a set A or a sequence X. During such a discussion we may use a corresponding term $\mathbf{A}$ and $\mathbf{X}$. In each such case, the implicit assumption is that the term (e.g., $\mathbf{A}$) corresponds to the object of discourse (e.g., A).

We describe the types of Scunak below.

– $\mathbf{obj}$ is the type of all mathematical objects. In set theory it is very common to consider any mathematical object as a set. Scunak reflects this idea by having a synonym $\mathbf{set}$ for the basic type $\mathbf{obj}$.
– $\mathbf{prop}$ is the type of all propositions.
– Proof types: $\mathbf{pf}\,p$ is the type of (all) proofs of the proposition p. Note that $\mathbf{pf}\,p$ is empty if p is unprovable. Proof types are a form of dependent types since they depend on propositions.

- Class types: These are types that correspond to the class $\{x|\phi(x)\}$ where $\phi(x)$ is a proposition depending on a mathematical object x. Such types are called *class types* and they depend on predicates ϕ. An inhabitant of a class type $\mathbf{class}\,\phi$ is a pair term $\langle s, \rho\rangle$, where $\phi : \mathbf{obj} \to \mathbf{prop}$ is a predicate, $s : \mathbf{obj}$ is a mathematical object and $\rho : \mathbf{pf}\ (\phi\, s)$ is the proof of the proposition $(\phi\, s)$. Note that without proof irrelevance, there could be more than one proof of $(\phi\, s)$, and hence more than one representative of type $\mathbf{pf}\ (\phi\, s)$.
- Π-types: The remaining types in Scunak are the dependent Π-types, which are generalizations of simple function types. We write $S \to T$ for $\Pi x : S. T$ if x does not occur in the output type T.

One represents a formal mathematical theory in Scunak by giving a *signature* Σ which is a list of constants, abbreviations and claims. We describe each of these below.

- A *constant* is specified by a name c and a type S. A constant corresponds to a basic constructor or axiom of the theory.
- An *abbreviation* is specified by a name c, a type S and a term t. An abbreviation corresponds to a defined constructor or a proved theorem.
- A *claim* is specified by a name c and a type S. A claim corresponds to a constructor we intend to define, or a proposition we intend to prove. In essence, claims are constants which should become abbreviations in a later version of a signature.

Scunak uses type checking to ensure signatures are declared in a valid manner.

2.2 Mac Lane Set Theory in Scunak

The current version of Scunak provides a variety of set theories, including a theory of hereditarily finite sets (see [6]), forms of Mac Lane set theory (see [5]) and a form of Zermelo-Fraenkel set theory with axiom of choice ($\mathbf{ZFC}$, see [9]). Each set theory is given as a signature in the type theory which can be loaded as a "kernel." The user can choose the appropriate set theory by loading the corresponding kernel.

We have worked in a set theory that is a form of Mac Lane set theory with universes, the axiom of choice and foundation ($\mathbf{MACU}$). One of the first set theories implemented in Scunak was Mac Lane set theory with universes, but without choice or foundation ($\mathbf{MU}$). The signature corresponding to $\mathbf{MU}$ is given in [5]. Aside from the fact that $\mathbf{MACU}$ includes choice and foundation (adding two constants), the formulation of universes in the two theories are different (removing two constants). Both the signature for $\mathbf{MU}$ and $\mathbf{MACU}$ consist of 29 constants. A description of Mac Lane set theory can be found in [11].

We briefly mention the constants used to construct propositions and set theoretical concepts.

There are three constants in the signature for propositions. There is a constant for the logical connective $\neg$, which is the only logical connective represented by a constant. There are two constants for the basic relations $=$ and $\in$ in set theory.

Six constants are defined in the signature for constructors corresponding to the following axioms of $\mathbf{MACU}$.

– **Axiom of empty set:** There is a set $\emptyset$ containing no elements.
– **Axiom of separation:** Given any set A and any property ϕ, there is a set of elements x of A (a subset of A) for which $\phi(x)$ holds.
– **Axiom of power set:** For any set A, there is a set $\mathcal{P}(A)$ (the power set of A) such that the elements of $\mathcal{P}(A)$ are exactly the subsets of A.
– **Axiom of union:** For any set A, there is a set $\bigcup A$ such that if $x \in \bigcup A$, then there is an element $y \in A$ such that $x \in y$.
– **Set adjoin:** For two sets A and B, $\{A\} \cup B$ is a set.
– **Universes:** For any set A, there is a set $Univ(A)$ which contains A, is transitive, and is closed under power set. (Note that $Univ(\emptyset)$ must be an infinite set.)

The remaining constants correspond to deduction rules for the basic set theory [5], as well as choice and foundation.

The other logical connectives and set theoretical notions are given as abbreviations from the constants mentioned above (see [6] for a presentation of the derivations).

2.3 A Modular Treatment of the Real Numbers

If one is formalizing mathematics within a foundational framework, then one must face the question of how to treat the real numbers. Fundamentally, the question is whether the real numbers should be constructed or axiomatized. In Scunak, we want all our mathematical content to be reduced to the basic foundational axioms. To obtain this goal, we could construct a signature with three sections:

1. **Set Theory Intro:** Axioms of set theory and basic set theoretic constructions
2. **Constructing the Reals:** A construction of the reals
3. **Real Analysis Intro:** Results from real analysis

Such a signature would guarantee that all our results can be traced back to the original axioms of set theory.

Our primary goal, however, was to follow the textbook [2], we note carefully how the authors introduce the reals in the first paragraph of Chapter 2 of [2]:

> In this chapter we shall discuss the essential properties of the real number system R. Although it is possible to give a formal construction of this system on the basis of a more primitive set (such as the set N of natural numbers or the set Q of rational numbers), we have chosen not to do so. Instead, we exhibit a list of fundamental properties associated with the real numbers and show how further properties can be deduced from them.

Bartle and Sherbert are quite explicit that they are not constructing a set of reals. However, they also refer to “the” real number system R, indicating that they have a real number system R already. How should this be reflected in the formalized version?

We decided to construct a signature of the following form:

1. **Set Theory Intro:** Axioms of set theory and basic set theoretic constructions
2. **Claiming the Reals:** Claims corresponding to the real number system
3. **Real Analysis Intro:** Results from real analysis

The claims in the second section would behave like basic constants and axioms, but could be later given definitions. The idea was that using claims would force the real analysis section to be independent of the construction of the reals (as in the textbook). This approach enforces a level of modularity between sections. The second section could be replaced by different constructions of the reals so long as the types of the claims corresponding the the real number system are the same.

In addition to the independence of the real analysis section on the construction of the reals, we found that the real analysis section was largely independent of the underlying set theory as well. In particular, while we choose to use $\mathbf{MACU}$ as the underlying set theory, after encoding the mathematical content it became clear that the axioms of choice and foundation were never used.[1] In fact, since we are working with a claimed set of reals, we do not even need an axiom of universes (or any axiom of infinity). Without difficulty, one can change the underlying set theory to be $\mathbf{MU}$, $\mathbf{MACU}$, $\mathbf{ZFC}$ or even a theory of hereditarily finite sets. Note that if the underlying set theory is a theory of hereditarily finite sets, then there is no hope of constructing the real numbers; they must remain open claims in this case.

3 How Does the Scunak Type Theory Reflect Informal Mathematics?

Informal presentations of mathematical knowledge in textbooks are untyped, but their formal versions in most mechanized systems for mathematics correspond to typed representations. We illustrate how the informal presentation of mathematics we have taken from [2] is reflected formally in Scunak by identifying several properties we observe in the formal version as consequences of the Scunak type theory.

3.1 Syntax

We briefly mention the concrete syntax employed for the examples we present in this paper. The conrete syntax used for terms and types is PAM (Pseudo-Automath) syntax [6]. The PAM syntax provides human-readable forms of notation to denote several mathematical operators using a combination of infix notation and special binder notation. We will use the `typewriter` font to present material formalized in PAM syntax.

The symbol `::` is an infix notation for the constant **in** that represents the membership relation $\in$ of sets.

The PAM syntax for the λ-binder is `\`. For convenience we also include some special forms for binders encoded as constants or abbreviations. The PAM syntax for the proof type $\mathbf{pf}\,p$ is `|- p`.

Given a set A and a property ϕ, we have the constant

$$\mathbf{dsetconstr} : \Pi\mathbf{A} : \mathbf{obj}.\, \Pi\phi : (\mathbf{class}\,(\mathbf{in}\,\mathbf{A}) \to \mathbf{prop}).\, \mathbf{obj}$$

corresponding to the Axiom of Separation. Given a set A and a proposition $P(x)$ which depends on an element x of A, the term $(\mathbf{dsetconstr}\,\mathbf{A}\,(\lambda\mathbf{x}\,.\,\mathbf{P}))$ corresponds to the

[1] We should note, however, that some lemmas were left as open claims. It is possible, though unlikely, that some of these lemmas might require choice or foundation.

subset $\{x \in A | P(x)\}$ of A. One can write this as `(dsetconstr A (\x.P))` in PAM syntax. PAM syntax also includes the syntactic sugar `{x:A|P}` for such a term.

Quantifiers are handled in a similar way. The quantifiers derived in the kernel of $\mathbf{MACU}$ are bounded quantifiers. That is, they are bounded to certain domains (sets) and have the form $\forall x \in A. P(x)$, $\exists x \in A. P(x)$ for a set A and a property $P(x)$ depending on an element x in A. The abbreviations corresponding to the bounded universal and the existential quantifiers have the names $\mathbf{dall}$ and $\mathbf{dex}$ and have the type $\Pi\mathbf{A} : \mathbf{obj}. (\mathbf{class}\,(\mathbf{in}\,\mathbf{A}) \to \mathbf{prop}). \mathbf{prop}$.

In PAM syntax, one can write `(forall x:A . P)` and `(exists x:A . P)` as syntactic sugar for $(\mathbf{dall}\,\mathbf{A}\,(\lambda\mathbf{x}.\mathbf{P}))$ and $(\mathbf{dex}\,\mathbf{A}\,(\lambda\mathbf{x}.\mathbf{P}))$ for a term $\mathbf{P}$ with type $\mathbf{prop}$.

After claiming the set of reals and defining the ordering relation on the reals, the symbol > is given as infix notation for the formal version of the 'greater than' operator.

In order to aid readability, we will sometimes mix notations in the discussion below. Also, we sometimes mention a "type" and give PAM syntax, by which we mean the type specified by the given PAM syntax.

We add variables to a context by giving the variable name, a colon, and a PAM specification of the type, all surrounded by brackets. For example, if we want to introduce a set A into the context, we actually introduce a variable $\mathbf{A}$ of type $\mathbf{set}$ into the context using the PAM syntax `[A:set]`.

3.2 Sets as Types

In the Scunak type theory, the notion of set is represented by the basic type $\mathbf{set}$, which is a synonym for the basic type $\mathbf{obj}$ of all mathematical objects.

When we formalize mathematics in Scunak we quite often use sets as types of certain terms, in particular, when we work with elements of sets. For example, consider a set A and an element x of A. We can represent these objects in Scunak by declaring variables $\mathbf{A}$ and $\mathbf{x}$ to have certain types in a context. We declare this in PAM syntax as follows: `[A:set][x:A]`.

The type of $\mathbf{x}$ is $\mathbf{class}\,(\mathbf{in}\,\mathbf{A})$. Intuitively, this corresponds to the fact that x belongs to the class of objects that are in the set A. The class type $\mathbf{class}\,(\mathbf{in}\,\mathbf{A})$ is often written as `A` in PAM syntax leaving out $\mathbf{class}$ and $\mathbf{in}$. This allows any set to be used as the "type" of its elements.

Note that the above representation of $x \in A$ uses dependent types. The type of $\mathbf{x}$ depends on the variable $\mathbf{A}$. This representation is quite compact compared to the representations of simple type systems, since the information $x \in A$ is contained in the type of $\mathbf{x}$. In simply typed systems one is required to either assume $\mathbf{A}$ is the simple type of $\mathbf{x}$ or add the information $(x \in A)$ into the formalizations usually as the antecedent of an implication $((x \in A) \Rightarrow \ldots)$. This means, one carries $x \in A$ as an extra information in the formalizations.

We now discuss some examples that demonstrate the use of sets as types in the material we have formalized in Scunak.

Fig. 1 shows the definition of the notion of a *lower bound* of a set of real numbers taken from [2] and its corresponding formal representation in Scunak in PAM syntax.

A real number w is represented as an object that is in the set $\mathbb{R}$ of real numbers as `[w:R]`, where `R` denotes the set of real numbers we have claimed in Scunak.

A subset S of $\mathbb{R}$ is represented as an object that is an element of the power set of $\mathbb{R}$, in PAM syntax as `[S:(powerset R)]`. Here `powerset` is the PAM version of the constant named $\mathbf{powerset}$ with type $\mathbf{obj} \to \mathbf{obj}$ in the kernel of Scunak corresponding to the axiom of power set.

`realLowerBoundOf` and `realLeq` are the formal versions of a *lower bound* of a set and the relation $\leq$. They inhabit the types `(powerset R) -> R -> prop` and `R -> R -> prop`, respectively.

Note that the bound variable `s` in `(forall s:S . (realLeq w s))` has the type `S`, whereas `realLeq` expects two arguments of type `R`. Scunak uses a special type conversion mechanism to type-check the application of `realLeq w` to `s`. We discuss the mechanism in Section 3.3.

Type Refinement using the Axiom of Separation. As we mentioned earlier, the Axiom of Separation is encoded in Scunak. Here we show how one can use the encoding to give refined types. Given a set A and a property ϕ, we can form the set $\{x \in A|\phi(x)\}$ and use this as a refined version of the type corresponding to A.
$\mathbf{A} : \mathbf{set}$
$\phi : (\mathbf{class}\,(\mathbf{in}\,\mathbf{A})) \to \mathbf{prop}$
$\mathbf{x} : \mathbf{class}\,(\mathbf{in}\,\{\mathbf{x} \in \mathbf{A}|(\phi\,\mathbf{x})\})$

This style of type refinement corresponds to linguistic specifications in informal mathematical texts. For example, "a **lower bound** w" as stated in Fig. 2 taken from the definition of an infimum of a set of real numbers in [2] is a linguistic specification of a real number that has the property of being a lower bound of a set S of real numbers. The formal version in Fig. 2, uses separation to reflect the informal specification by refining the type `R` of real numbers with the relation `realLowerBoundOf`. The resulting type `{x:R|(realLowerBoundOf S x)}` is the type a variable representing a real number w that is a lower bound of a set S of real numbers.

In MIZAR, there is an alternative type refinement mechanism that uses MIZAR "attributes" (see [15]) to represent such linguistic specifications in textbooks.

In ISABELLE-HOL, one can employ a type definition mechanism rather than type refinements for the presentation of these specifications. For example, a simple type α and a closed, nonempty predicate ϕ on α can be used to define the type, say γ, of terms for which the property ϕ holds. Along with the definition, there is usually a function that serves the purpose of an explicit type conversion between α and γ.

Type refinement using the Axiom of Separation does not require a function for the explicit conversion of type $\mathbf{class}\,(\mathbf{in}\,\{\mathbf{x} \in\ \mathbf{A}|\ (\phi\,\mathbf{x})\})$ to type $\mathbf{class}\,(\mathbf{in}\,\mathbf{A})$. The conversion is performed implicitly by means of certain inference rules in the kernel of

Definition. Let S be a subset of $\mathbb{R}$. A number $w \in \mathbb{R}$ is said to be a **lower bound** of S if $w \leq s$ for all $s \in S$.

```
[S:(powerset R)]
[w:R]
(realLowerBoundOf S w):prop=(forall s:S . (realLeq w s)).
```

Fig. 1. An Example of Using Sets as Types

Scunak. By means of this implicit type conversion, the application of the infix operator `>`, which expects two arguments of type `R`, to the terms `w` and `v`, which have the refined type `{x:R|(realLowerBoundOf S x)}`, type-checks.

3.3 Type Conversions

From everyday programming languages like C/C++, we are familiar with the notion of implicit type conversions, also known as **coercions**, used for converting numeric types (like the type int of integers and float of floating numbers). The general idea of type conversions is that a variable of a certain type is forced to behave as if it has another type. This means, if a type S is coerced to another type T, then any term that expects a member of T can accept an argument that is a member of S.

Scunak does not have numeric types. Numbers are members of class types. For instance, `N` and `R` are PAM notation for the sets $\mathbb{N}$ of natural numbers and $\mathbb{R}$ of real numbers, respectively. Hence `N` and `R` can be used as the types of natural numbers and real numbers. This means, there are distinct types for numbers in Scunak. Nevertheless, one can naturally expect a natural number to behave as a real number (since $\mathbb{N} \subseteq \mathbb{R}$ and thus a natural number is a real number). In other words, one technically expects a term with type `N` to behave as if it has type `R`. Scunak has a type conversion mechanism for subsets of sets (like $\mathbb{N}$ of $\mathbb{R}$). We describe the mechanism below.

Suppose there are two sets A and B with the property $B \subseteq A$. Given two corresponding terms $\mathbf{A}$ and $\mathbf{B}$, the type $\mathbf{class}\ (\mathbf{in}\, \mathbf{B})$ can be converted to type $\mathbf{class}\ (\mathbf{in}\, \mathbf{A})$ if there is a proof of the property $B \subseteq A$ (i.e, if there is a term ρ with type $\mathbf{pf}\ (\mathbf{B} \subseteq \mathbf{A})$). The conversion requires an explicit statement in the presence of a proof of $B \subseteq A$ in the formalizations. The statement needs to be declared only once. Then, in any future formalization, any term that expects arguments corresponding to elements of A can be applied to terms corresponding to elements of B without violating type checking. One should note that the type conversion do not affect the resulting type of an application.

Arithmetical operators such as addition, subtraction, multiplication are given for real numbers and their application to elements of subsets of $\mathbb{R}$ (like naturals, integers, rationals, etc.) is handled through converting the types of elements of subsets of $\mathbb{R}$ to the type of real numbers. The definitions of arithmetical operators are not overloaded for each distinct type of numbers.

Definition. Let S be a subset of $\mathbb{R}$. If S is bounded below, then a lower bound w is said to be an **infimum** (or a **greatest lower bound**) of S if no number larger than w is a lower bound of S.

```
[S:(powerset R)]
[w:{x:R|(realLowerBoundOf S x)}]
(realInfimum S w):prop=
 (not (exists v:{x:R|(realLowerBoundOf S x)} .
   (v > w))).
```

Fig. 2. An Example of Separation in Scunak

3.4 Pair Terms

Pair terms are inhabitants of class types. In the formalizations, pair terms are frequently used to address type checking issues in the case of no available type conversion procedures. The type conversion procedure we have discussed in Section 3.3 is a special case used to convert class types induced by the predicate $(\mathbf{in}\,\mathbf{A})$ for a set A. Currently, the only general way to convert a term of a class type or of type $\mathbf{obj}$ to another class type is by using pairs as we will describe below.

Suppose a term $\mathbf{x}$ of type $\mathbf{class}\,\phi$ is expected to behave as a member of type $\mathbf{class}\,\psi$ for predicates ϕ and ψ. If one can prove that the term $\mathbf{x}$ (as an object) satisfies $(\psi\,\mathbf{x})$, then the proof can be used to construct a pair term of type $\mathbf{class}\,\psi$, which can be given as an argument to a term $\mathbf{t}$ that expects a member of the latter type.

Technically, $\mathbf{x} : \mathbf{class}\,\phi$ is (judgmentally) the same as a pair term $\langle\pi_1(\mathbf{x}), \pi_2(\mathbf{x})\rangle$ with $\pi_1(\mathbf{x}) : \mathbf{obj}$ and $\pi_2(\mathbf{x}) :\ \mathbf{pf}\,(\phi\,\pi_1(\mathbf{x}))$. The first projection $\pi_1(\mathbf{x})$ of the pair is the object representation of $\mathbf{x}$. If one can prove that $(\psi\,\pi_1(\mathbf{x}))$ holds, then the proof $\rho : \mathbf{pf}\,(\psi\,\pi_1(\mathbf{x}))$ can be paired together with $\pi_1(\mathbf{x})$ and the resulting pair term has the type expected by $\mathbf{t}$. In PAM syntax, π_1 and π_2 are not written down explicitly.

If a term $\mathbf{x}$ with type $\mathbf{obj}$ is expected to behave as a member of a class type, say $\mathbf{class}\,\phi$ for a predicate ϕ, then $\mathbf{x}$ is paired together with the proof of the proposition $(\phi\,\mathbf{x})$.

An instance of using pair terms in the formalizations in Scunak is the case, where a member of type $\mathbf{class}\,(\mathbf{in}\,\mathbf{B})$ is expected to behave as if it has type $\mathbf{class}\,(\mathbf{in}\,\mathbf{A})$, but we do not have a proof that $B \subseteq A$ holds. In this case, the explicit conversion of types in Scunak (as we have mentioned in Section 3.3) cannot be applied, since the conversion is specific to sets A and B for which $B \subseteq A$ holds. We use pair terms like in the general case above. The proof we are looking for is that the object representation of the term with type $\mathbf{class}\,(\mathbf{in}\,\mathbf{B})$ is an element of the set A.

3.5 Representation of Functions

In Scunak, functions are represented as objects that are functional binary relations on arbitrary sets. This means, an element of the relation's domain is associated with a unique element of the relation's range. The encoding of this representation is presented in [6] by introducing the kernel constants $\mathbf{func}$, $\mathbf{ap}$ and $\mathbf{lam}$ with their formal definitions. They respectively serve the purpose of declaring functions, applying functions to their arguments and specifying functions.

We briefly mention how these constants are used. A function f from the set A to the set B can be represented as a member of the class of objects that are functions from A to B. For an element a of A the function application $f(a)$ is represented as $(\mathbf{ap}\,\mathbf{A}\,\mathbf{B}\,\mathbf{f}\,\mathbf{a})$, where $\mathbf{A}$, $\mathbf{B}$ have type $\mathbf{set}$, $\mathbf{f}$ has type $\mathbf{class}\,(\mathbf{func}\,\mathbf{A}\,\mathbf{B})$ and $\mathbf{a}$ has type $\mathbf{class}\,(\mathbf{in}\,\mathbf{A})$. The type of $(\mathbf{ap}\,\mathbf{A}\,\mathbf{B}\,\mathbf{f}\,\mathbf{a})$ is $\mathbf{class}\,(\mathbf{in}\,\mathbf{B})$. If $\mathbf{t}$ is a term which has type $\mathbf{class}\,(\mathbf{in}\,\mathbf{B})$ when $\mathbf{x}$ is a declared variable with type $\mathbf{class}\,(\mathbf{in}\,\mathbf{A})$, then $(\mathbf{lam}\,\mathbf{A}\,\mathbf{B}\,(\lambda\mathbf{x}.\mathbf{t})\,)$ has type $\mathbf{class}\,(\mathbf{func}\,\mathbf{A}\,\mathbf{B})$ and represents the λ-abstraction that takes an element of the set A and returns an element of the set B.

An alternative way to work with functions in Scunak is to use the notion of a *set of functions* represented by the kernel constant $\mathbf{funcSet} : \mathbf{obj} \to \mathbf{obj} \to \mathbf{obj}$, which takes two objects (sets) A and B, and returns the set of functions from A to B.

Given two sets A and B, we can represent a function f from A to B using $\mathbf{funcSet}$ by declaring variables $\mathbf{A}$, $\mathbf{B}$ and $\mathbf{f}$ as follows:
$\mathbf{A} : \mathbf{set}$
$\mathbf{B} : \mathbf{set}$
$\mathbf{f} : \mathbf{class}\,(\mathbf{in}\,(\mathbf{funcSet}\,\mathbf{A}\,\mathbf{B}))$
where $(\mathbf{in}\,(\mathbf{funcSet}\,\mathbf{A}\,\mathbf{B}))$ is a predicate that takes a term and checks whether it is in the set of functions from A to B. In PAM syntax, we write `[f:(funcSet A B)]`.

For declared variables $\mathbf{A}$ and $\mathbf{B}$ with type $\mathbf{set}$, the semantic interpretation of both a term with type $\mathbf{class}\,(\mathbf{func}\,\mathbf{A}\,\mathbf{B})$ and a term with type $\mathbf{class}\,(\mathbf{in}\,(\mathbf{funcSet}\,\mathbf{A}\,\mathbf{B}))$ is the same: A function from the set A to the set B.

The corresponding function application and λ-abstraction operators for $\mathbf{funcSet}$ are $\mathbf{ap2}$ and $\mathbf{lam2}$ with the following types respectively.
$\Pi\mathbf{A} : \mathbf{set}.\Pi\mathbf{B} : \mathbf{set}.\mathbf{class}\,(\mathbf{in}\,(\mathbf{funcSet}\,\mathbf{A}\,\mathbf{B})) \to \mathbf{class}\,(\mathbf{in}\,\mathbf{A}) \to \mathbf{class}\,(\mathbf{in}\,\mathbf{B})$
$\Pi\mathbf{A} : \mathbf{set}.\Pi\mathbf{B} : \mathbf{set}.(\mathbf{class}\,(\mathbf{in}\,\mathbf{A}) \to \mathbf{class}\,(\mathbf{in}\,\mathbf{B})) \to \mathbf{class}\,(\mathbf{in}\,(\mathbf{funcSet}\,\mathbf{A}\,\mathbf{B}))$

The use of $\mathbf{ap2}$ and $\mathbf{lam2}$ is similar to that of $\mathbf{ap}$ and $\mathbf{lam}$. For terms $\mathbf{A} : \mathbf{set}$, $\mathbf{B} : \mathbf{set}$, $\mathbf{f} : \mathbf{class}\,(\mathbf{in}\,(\mathbf{funcSet}\,\mathbf{A}\,\mathbf{B}))$, $\mathbf{a} : \mathbf{class}\,(\mathbf{in}\,\mathbf{A})$ and $\mathbf{x} : \mathbf{class}\,(\mathbf{in}\,\mathbf{A})$,

- $(\mathbf{ap2}\,\mathbf{A}\,\mathbf{B}\,\mathbf{f}\,\mathbf{a})$ represents an element $f(a) \in B$ for $a \in A$,
- $(\mathbf{lam2}\,\mathbf{A}\,\mathbf{B}\,(\lambda\mathbf{x}.\mathbf{t}))$ represents a function f determined by $f(x) = t$ for $x \in A$.

Sequences As a special case of working with functions in Scunak, we present the formalization of the notion of sequences. Fig. 3 shows the informal definition of a sequence of real numbers taken from [2] and its formal representation in Scunak.

We first formalize a general notion of sequences. Given an arbitrary set A, a sequence in the set A is a function from the set $\mathbb{N}$ of natural numbers to A. We define a set constructor called `sequenceIn` that takes a term corresponding to a set A and returns the set of functions from $\mathbb{N}$ to A using the constant $\mathbf{funcSet}$.

Definition. A **sequence of real numbers** (or a **sequence in** $\mathbb{R}$) is a function on the set $\mathbb{N}$ of natural numbers whose range is contained in the set $\mathbb{R}$ of real numbers.

```
[A:set]
(sequenceIn A):set=(funcSet N A).
notation RSeq (sequenceIn R).
[X:RSeq]
```

Fig. 3. Sequences

We instantiate `sequenceIn` with `R` to yield the set of functions from $\mathbb{N}$ to $\mathbb{R}$, which we denote as `RSeq`. Since a sequence in $\mathbb{R}$ is a member of the set represented by `RSeq`, we can use `RSeq` as the type of a sequence in $\mathbb{R}$ as `[X:RSeq]` in PAM syntax.

We define the value of a sequence at index $n \in \mathbb{N}$ using $\mathbf{ap2}$. The value of a sequence at index n is the value obtained when the sequence, as a function, is applied to n.

```
[X:(sequenceIn A)]
[n:N]
notation XinfuncSetNA (sequenceIn#U A (\x.(X::x)) X).
(valueAt A X n):A=(ap2 N A <X,XinfuncSetNA> n).
notation subA (valueAt A).
notation sub (valueAt R).
```

The term `valueAt` takes terms representing a set A, a sequence in A and a natural number n, and returns a term representing the value of the sequence at index n. The pair term in the definition is to ensure that `(ap2 N A)` is applied to an argument with the expected type `(funcSet N A)`. The type of a sequence in $\mathbb{R}$ is in PAM syntax `(sequenceIn R)`, which is not the type expected by `(ap2 N A)`. The PAM term `(sequenceIn#U A (\x.(X::x)) X)` is the proof that the context variable `X` is in the set of functions from $\mathbb{N}$ to A. We obtain the proof by unfolding the definition of `sequenceIn`. For readability, we declare `XinfuncSetNA` as notation for this proof.

The last component we need in order to be able to work with sequences is a sequence constructor and we define it using $\mathbf{lam2}$.

```
[A:set]
[f:N -> A]
notation lam2NAf
          (sequenceIn#F A (\x.((lam2 N A f)::x)) (lam2 N A f)).
(sequenceconstr A f):(sequenceIn A)=<(lam2 N A f),lam2NAf>.
```

The abbreviation `sequenceconstr` takes a term representing a set A and a meta-level function with type `N -> A` and gives back a term corresponding to a sequence in A that is determined by the meta-level function. Note `(lam2 N A f)` has the type `(funcSet N A)`. `lam2NAf` is a notation that stands for the proof that the object-level λ-abstraction `(lam2 N A f)` is a sequence in A. We use `lam2NAf` to obtain a term of type `(sequenceIn A)`.

4 A Case Study

After presenting the Monotone Convergence Theorem, Bartle and Sherbert give a number of examples which use the Monotone Convergence Theorem. We present the formal version of the first of these examples: $\lim\left(\frac{1}{\sqrt{n}}\right) = 0$. The proof in [2] essentially consists of one sentence. The statement and short proof from [2] are shown in Fig. 4.

Example. $lim\ \left(\frac{1}{\sqrt{n}}\right) = 0$.
Proof. Clearly, 0 is a lower bound for the set $\{\frac{1}{\sqrt{n}} : n \in N\}$, and it is not difficult to show that 0 is the infimum of the set $\{\frac{1}{\sqrt{n}} : n \in N\}$; hence $0 = lim\ \left(\frac{1}{\sqrt{n}}\right)$.

Fig. 4. An Example on Sequences

The formalization of the example is divided into the following parts:

– Formalization of necessary notions and theorems the example uses in its statement and proof in a PAM document
– An analysis of the informal proof to generate underspecified lemmata and their formalization
– Formalization of the mathematical statement of the example in a PAM file
– Formalization of the proof interactively in Scip

The underlying notions used in the example are the notions of a lower bound and an infimum of a set (of real numbers), sequences, the limit of a sequence, decreasing sequences, the square root function, the underlying set of a sequence, and the Monotone Convergence Theorem. Once these preliminaries are given, we give claims corresponding to the steps of the proof and then the final result.

The notions of a lower bound and an infimum of a set (of real numbers), and sequences are introduced in Sections 3.2 and 3.5. We have formalized the notion of the limit of a sequence and decreasing sequences as a term `lim:RSeq -> R -> prop` that takes a sequence of real numbers and a real number, and checks whether the proposed number is the limit of the given sequence, and `decreasing:RSeq -> prop` that takes a sequence of real numbers and checks whether the given sequence is decreasing. Whenever `X` is of type `RSeq`, then `(RSeqSet X)` (of type `set`) is defined to be the underlying set of the sequence `X`. The term `RSeqSetSubsetReals` abbreviates a proof that for any `X` of type `RSeq`, the underlying set `(RSeqSet X)` is in the power set of the reals.

Given the notions of limit and decreasing, we can represent the Monotone Convergence Theorem in PAM syntax as shown in Fig. 5. We explain Fig. 5 by giving the same information in natural language:

`X`**:** Let X be a sequence of reals.
`v`**:** Assume X is decreasing.
`a`**:** Let a be a real number.
`apf`**:** Assume a is a lower bound of the underlying set of X.
`w`**:** Assume a is an infimum of the underlying set of X. (Note that we cannot assert that a is an infimum unless we know it is a lower bound.)
`monotoneConvTheo-b-2`**:** The Monotone Convergence Theorem implies the limit of X is a.

The example we will consider is shown in PAM syntax in Fig. 6. We begin by explaining the notation. The symbol `X` is declared as notation for the sequence $\frac{1}{\sqrt{n}}$. Note that n is bound in this expression. This is reflected by the fact that `n` is λ-bound in the term `(\n.(1/ <(sqrt n),(sqrtNatInR-0 n)>))` which has type `N -> R` (in PAM syntax). `seqconstr` takes this term of function types and creates a term of type `RSeq`. We next declare `S` to be notation for the underlying set of `X`. We declare `SPR` as notation for a term proving `S` is in the power set of the reals. Finally, we declare notation `LB` for the set of lower bounds of the set `S` of reals.

Using this notation, we can represent the facts asserted in the proof of the example. Two facts are stated explicitly in the proof: 0 is a lower bound and 0 is an infimum. These two facts are represented as the claims `bs-example-3-3-3a-1` and

```
[X:RSeq]
[v:|- (decreasing X)]
[a:R]
[apf:|- (a::{x:R|(realLowerBoundOf
                    <(RSeqSet X),(RSeqSetSubsetReals X)> x)})]
[w:|- (realInfimum
        <(RSeqSet X),(RSeqSetSubsetReals X)>
         <a,apf>)]
(monotoneConvTheo-b-2 X v a apf w):|- (lim X a)?
```

Fig. 5. Formalization of Monotone Convergence in Scunak

`bs-example-3-3-3a-2` in Fig. 6. These two claims are essentially lemmas we commit to proving at some later time. Note that since the definition of infimum requires knowing that the element is a lower bound, the fact that 0 is a lower bound (as witnessed by the claim `bs-example-3-3-3a-1`) is used in the type of `bs-example-3-3-3a-2`. One of the premisses of the Monotone Convergence Theorem is that the sequence is monotone (in this case, decreasing). While the text does not explicitly say the sequence $\frac{1}{\sqrt{n}}$ is decreasing, we include this as a third claimed lemma `bs-example-3-3-3a-3`. Finally, we declare a claim `bs-example-3-3-3a` corresponding to the main result.

```
notation X
(sequenceconstr R (\n.(1/ <(sqrt n),(sqrtNatInR-0 n)>))).

notation S (RSeqSet X).
notation SPR (RSeqSetSubsetReals X).
notation LB {x:R|(realLowerBoundOf <S,SPR> x)}.

bs-example-3-3-3a-1:|- (0::LB)?
bs-example-3-3-3a-2:
 |- (realInfimum <S,SPR> <0,bs-example-3-3-3a-1>)?
bs-example-3-3-3a-3:|- (decreasing X)?
bs-example-3-3-3a:|- (lim X 0)?
```

Fig. 6. Formalization of the Example in Scunak

After Scunak has read the PAM file containing the information in Figs. 5 and 6, our goal changes to obtaining a proof term for the main result `bs-example-3-3-3a`. One way to give the proof is simply as a proof term. Since we have given names to the steps of the proof, such a proof term is small (but not enlightening):

```
(monotoneConvTheo-b-2 X bs-example-3-3-3a-3 0
  bs-example-3-3-3a-1 bs-example-3-3-3a-2)
```

Another way to give the proof is to construct it in Scip. A Scip session which constructs the proof is given in Fig. 7. This corresponds more closely to the text. We begin the Scip session with a "use" which lists the known facts we can use in the proof. In our case, we list the Monotone Convergence Theorem along with the claimed steps of the proof. Now we can construct the proof by giving three "facts." First, 0 is a lower bound of $\{\frac{1}{\sqrt{n}} : n \in N\}$. Second, 0 is an infimum of $\{\frac{1}{\sqrt{n}} : n \in N\}$. Note that these two statements correspond directly to the statements given in the textbook proof in Fig. 4 and to the claims given in Fig. 6. The third fact is that the sequence is decreasing. We end the proof by giving the Scip command `d`, indicating that the proof is done. Essentially, we have stated all the steps in the proof in the PAM file and we have then used Scip to appropriately combine them into a proof term.

```
prove bs-example-3-3-3a
use bs-example-3-3-3a-1 bs-example-3-3-3a-2
    bs-example-3-3-3a-3 monotoneConvTheo-b-2
fact (0::LB)
fact (realInfimum <S,SPR> <0,fact0>)
fact (decreasing X)
d
```

Fig. 7. Construction of the Proof in Scip

We have also experimented with this example in Isabelle-HOL [12] and Mizar [14]. We mention two interesting points.

The first point regards the use of dependent types to state definitions and theorems in a manner as close as possible to the text. In particular, we used the (dependent) type of lower bounds of S in the definition of infimum. In Isabelle-HOL, the restriction to simple types prevented us from using types. Instead, one must ignore such restrictions on arguments when defining concepts such as infimum and include the restrictions as premisses when formulating theorems. In Mizar one can define such dependent types, but only if they are nonempty. The most satisfying way we found to define such a type in Mizar was to assume S is a "bounded below subset of reals" when defining the type of lower bounds of S.

The second point regards the binding mechanism for the n in the sequence $\frac{1}{\sqrt{n}}$. One can easily give the sequence as a λ-term in Isabelle-HOL. We had difficulty trying to find an appropriate binding mechanism in Mizar.

In Mizar's library, the notion of a sequence of reals is represented by the mode `Real_Sequence` which is defined in terms of functions from naturals to reals [10]. One can easily use Mizar's `func` definition mechanism to define a unary constructor named `seq333a` which expects a natural number n and returns a real number $\frac{1}{\sqrt{n}}$. However, this does not yield the desired member of `Real_Sequence`. In the end, we formulated the example in Mizar by stating that if X is a real sequence and for all n, X_n is $\frac{1}{\sqrt{n}}$, then the limit of X is 0. Essentially this uses the universal quantifier as the binder, but leaves implicit the fact that the hypothesis determines a unique sequence X. Later, Krzysztof Retel pointed out that we could have used the `func` definition mechanism to define a nullary constructor named `seq333a` which has type `Real_Sequence`.

5 Conclusion

We have demonstrated that mathematical content informally represented in a textbook can be given a precise formal representation in Scunak. Especially useful aspects of Scunak include using sets as types, type conversions for subsets, and the handling of binding constructors (e.g., for binding n in the sequence $\frac{1}{\sqrt{n}}$). However, some aspects of the formal versions in Scunak were problematic. First, sometimes we needed to explicitly include proof objects in terms (as the second part of a pair of class type) for the purposes of type checking. A mechanism allowing users to leave out such proof objects (by looking them up somehow, not by performing proof search) would be helpful. Second, writing proofs as proof terms does not give a very human-readable (or "natural") representation of proofs. A MIZAR-style of proof presentation would be preferable. Essentially one would need a "compiler" which translates MIZAR-style proofs into Scunak proof terms. We leave such improvements as future work.

References

1. Autexier, S., Fiedler, A.: Textbook proofs meet formal logic - the problem of underspecification and granularity. In: Kohlhase, M. (ed.) MKM 2005. LNCS (LNAI), vol. 3863, pp. 96–110. Springer, Heidelberg (2006)
2. Bartle, R.G., Sherbert, D.R.: Introduction to Real Analysis. John Wiley and Sons, New York (1982)
3. Baur, J.: Syntax und semantik mathematischer texte. Diploma thesis, Saarland University, Saarbrücken, Germany (1999)
4. Bertot, Y., Castéran, P.: Interactive Theorem Proving and Program Development. Coq'Art: The Calculus of Inductive Constructions. In: Texts in Theoretical Computer Science. An EATCS Series, Springer, Heidelberg (2004)
5. Chad, E.: Combining Type Theory and Untyped Set Theory. In: Furbach, U., Shankar, N. (eds.) IJCAR 2006. LNCS (LNAI), vol. 4130, pp. 205–219. Springer, Heidelberg (2006)
6. Brown, C.E.: Encoding functional relations in Scunak. In: LFMTP'2006 (September 2006)
7. C.E. Brown. Scunak users manual (2006) `http://gtps.math.cmu.edu/cebrown/manual.ps`
8. Brown, C.E.: Verifying and invalidating textbook proofs using scunak. In: Borwein, J.M., Farmer, W.M. (eds.) MKM 2006. LNCS (LNAI), vol. 4108, pp. 110–123. Springer, Heidelberg (2006)
9. Brown, C.E.: Dependently Typed Set Theory. In: SEKI-Working-Paper SWP–2006–03, SEKI Publications, Saarland Univ (2006) ISSN 1860–5931
10. Kotowicz, J.: Real sequences and basic operations on them. Journal of Formalized Mathematics, 1 (1989)
11. Mac, S.: Mathematics, Form, and Function. Springer, Heidelberg (1986)
12. Nipkow, T., Paulson, L.C., Wenzel, M.: Isabelle/HOL. LNCS, vol. 2283. Springer, Heidelberg (2002)
13. Reed, J.: Proof irrelevance and strict definitions in a logical framework. Technical Report 02-153, School of Computer Science, Carnegie Mellon University (2002)
14. Rudnicki, P.: An overview of the mizar project. In: Workshop on Types for Proofs and Programs, pp. 311–332 (1992)
15. Wiedijk, F.: Mizar: An impression. `http://www.cs.kun.nl/~freek/mizar/mizarmanual.ps.gz`