Formalizing Axiomatic Theories of Truth

ppls-nd-prs

March 18, 2025

Chapter 1

Notes on notation

Lean is a type-theoretic language and some conceptualizations of common concepts are different in type theory than in set theory. Nevertheless, we want our formalization's documentation to be accessible to mathematicians without a firm background in type theory. Hence, we decided to not make explicit reference to types in the documentation. This is possible because of an intuitive correspondence between types and sets (see e.g. [2]). Hence, whenever the documentation includes a set, the reader can safely assume that set will be implemented in the LEAN code as a type of some sort.

Moreover, as Lean has additional functionalities on top of its type theoretic basis to allow for efficient programming and proof verification, even more limitations arise preventing the code from following exactly the common conceptualizations of set theoretic mathematics. The most notable of these deviations is that the symbol `0` in Lean has a reserved meaning, namely `Nat.zero`, an inhabitant of the primitive type `Nat`, describing the natural numbers. Note that hence the natural numbers in Lean include 0, the importance of which will become apparent when discussing the implementation of the concept of a language. As we are involved in proving statements about specific axiomatizations of arithmetic we have to define our own notion of the constant 0, and will therefore not use the symbol `0` for that but simply the symbol `zero`. Note that in this case the `zero` in our implementation more closely resembles the symbol 0, because it is just a symbol, and does not carry any extra meaning, which `0` does.

For our implementation of the syntax of first order formulas we make use of De Bruijn indices, where quantifiers are not accompanied by the variable they bind, but variables are natural numbers encoding how many levels higher their binding quantifier can be found. For example, the formula

$$\forall x(\exists y(P(y) \to P(x)) \land \exists z(R(z) \lor P(x)))$$

in regular first order syntax becomes

$$\forall(\exists(P(0) \to P(1)) \land \exists(R(0) \lor P(1)))$$

when making use of De Bruijn indices.

Chapter 2

Preliminaries

2.1 Syntax

The ordinary definition of a first-order language $\mathcal{L}$ proceeds from the specification of a signature $\mathcal{S}$ and then an inductive definition of the way formulas can be build with elements from $\mathcal{S}$, the set of logical connectives, variables and the quantifiers. See that for any first order language all these elements except the signature are the same. Hence, we could think of a specific first order language as being fully specified by its signature. For reasons of conceptual compactness that is the approach [3] have used for the specification of a language.

Definition 1 (first-order language)**.** Let A be a set of function symbols and B a set of predicate symbols. Then, a first-order language is a structure $\langle F, R\rangle$, where

(i) $F : \mathbb{N} \to A$ and

(ii) $R : \mathbb{N} \to B$.

Note that this defines a language as consisting only of functions and relations, whereas traditionally a language also contains a set of constants. Observe, however, that constants can be modelled as 0-ary functions, so this definition of a language does not limit expressive power. By providing concrete sets of function and relation symbols to F_i and R_i we obtain a specific language.

We will be implementing the languages $\mathcal{L}$ and $\mathcal{L}_T$ that are necessary for the proof of Halbach's [1] Theorem 7.5 of the conservativity of `TB` and `UTB` over `PA`.

Definition 2 ($\mathcal{L}$: the language of peano arithmetic with syntactic functions and predicates)**.** The language of peano arithmetic including syntactic functions and predicates is the first-order language $\mathcal{L} = \langle F, R\rangle$, where

(i) F is defined by

  (a) $F(0) = \{\texttt{null}\}$,

  (b) $F(1) = \{\texttt{succ}, \texttt{num}, \texttt{neg}, \texttt{forall}, \texttt{exists}, \texttt{denote}\}$,

  (c) $F(2) = \{\texttt{add}, \texttt{mult}, \texttt{conj}, \texttt{disj}, \texttt{cond}\}$ and

  (d) $F(3) = \{\texttt{subs}\}$ and

(ii) R is defined by

(iii) $R(1) = \{\texttt{Variable}, \texttt{Constant}, \texttt{Closed\_Term}, \texttt{Term}, \texttt{Formula}_{\mathcal{L}}, \texttt{Sentence}_{\mathcal{L}}, \texttt{Formula}_{\mathcal{L}_T}, \texttt{Sentence}_{\mathcal{L}_T}\}$.

Definition 3 ($\mathcal{L}_T$: the language of peano arithmetic including the truth predicate symbol)**.** Let $\mathcal{L} = \langle F_{\mathcal{L}}, R_{\mathcal{L}} \rangle$ be the language of peano arithmetic including syntactic functions symbols and predicates. Then, the language $\mathcal{L}_T$ is the first-order language $\mathcal{L}_T = \langle F, R \rangle$, where

(i) $F = F_{\mathcal{L}}$ and

(ii) R is defined by $R(1) = R_{\mathcal{L}}(1) \cup \{\texttt{Tr}\}$.

We can then work our way up, first to the level of terms and then that of formulas. For this, we use `mathlib`'s implementation of `Term`s and `BoundedFormula`s.

Definition 4 (Term)**.** Let $\mathcal{L} = \langle F, R \rangle$ be a first-order language and α a set used to index free variables. Then the set of terms with respect to $\mathcal{L}$ and α denoted $\mathcal{T}(\mathcal{L}, \alpha)$ is the smallest set such that

(i) the set of variables with respect to α denoted $\mathcal{V}(\alpha) = \{x_i | i \in \alpha\} \subseteq \mathcal{T}(\mathcal{L}, \alpha)$ and

(ii) for all $i \in \mathbb{N}$, if $f \in F(i)$ and $x_1, ..., x_i \in \mathcal{T}(\mathcal{L}, \alpha)$, then $f(x_1, ..., x_i) \in \mathcal{T}(\mathcal{L}, \alpha)$.

The concept of `Term` is used to define the concept `BoundedFormula`.

Definition 5 (BoundedFormula)**.** Let $\mathcal{L} = \langle F, R \rangle$ be a first-order language, α a set indexing free variables, n the intended number of variables bound by a quantifier and $\mathcal{T}(\mathcal{L}, \alpha \cup \{x_1, ..., x_n\})$ a set of terms. Then, the set of bounded formulas with respect to these variables $\mathcal{B}(\mathcal{L}, \alpha, n)$ is the smallest set such that

(i) $\bot \in \mathcal{B}(\mathcal{L}, \alpha, n)$,

(ii) if $t_1, t_2 \in \mathcal{T}(\mathcal{L}, \alpha \cup \{x_1, ..., x_n\})$, then $t_1 = t_2 \in \mathcal{B}(\mathcal{L}, \alpha, n)$,

(iii) for all $i \in \mathbb{N}$, if $P \in R(i)$ and $t_1, ..., t_i \in \mathcal{T}(\mathcal{L}, \alpha \cup \{x_1, ..., x_n\})$, then $P(t_1, ..., t_i) \in \mathcal{B}(\mathcal{L}, \alpha, n)$,

(iv) if $f_1, f_2 \in \mathcal{B}(\mathcal{L}, \alpha, n)$, then $(f_1 \Rightarrow f_2) \in \mathcal{B}(\mathcal{L}, \alpha, n)$ and

(v) if $f \in \mathcal{B}(\mathcal{L}, \alpha, n + 1)$, then $\forall f \in \mathcal{B}(\mathcal{L}, \alpha, n)$.

A `Formula` is defined as a `BoundedFormula` that has no variables that still need to be bound.

Definition 6 (Formula)**.** Let $\mathcal{L}$ be a first-order language and α a set indexing variables. Then the set of formulas with respect to $\mathcal{L}$ and α denoted $\mathcal{F}(\mathcal{L}, \alpha)$ is defined as the set of bounded formulas $\mathcal{B}(\mathcal{L}, \alpha, 0)$.

A `Sentence` is then defined as a `Formula` that has no free variables.

Definition 7 (Formula)**.** Let $\mathcal{L}$ be a first-order language. Then the set of sentences with respect to $\mathcal{L}$ denoted $\mathcal{S}(\mathcal{L})$ is defined as the set of formulas $\mathcal{F}(\mathcal{L}, \emptyset)$.

Chapter 3

Disquotation

Chapter 4

Examples

Bibliography

[1] Volker Halbach. *Axiomatic Theories of Truth*. Cambridge University Press, 2011.

[2] R.P. Nederpelt. Type systems – basic ideas and applications. In R.P. Nederpelt, J.H. Geuvers, and R.C. de Vrijer, editors, *Selected Papers on Automath*, volume 133 of *Studies in Logic and the Foundations of Mathematics*, pages 229–247. Elsevier, 1994.

[3] SnO2WMaN and Palalansoukî. FormalizedFormalLogic/Foundation (version 4.15.0-rc1). https://github.com/FormalizedFormalLogic/Foundation/tree/d5a5b566c4bbc8d91a509979ae5091e750e67059.