Saint-Petersburg State University

Software Engineering Chair

Group 21.B15-mm

Formalization of definability theory in Lean

Ilya Dudnikov

Internship report
in a «Solution» form

Scientific supervisor:
assistant of computer science department
C.Sc. in Mathematics and Physics
Mikhail R. Starchak

Saint-Petersburg
2023

Contents

Introduction

There is no doubt that mathematics is a remarkably complex field of knowledge. There is a significant number of various definitions and, built upon those definitions, theorems that need to be rigorously proved. However, people inevitably make mistakes, which sometimes leads to incorrect proofs. In a recent paper [6], the authors address some inaccuracies found in [3], a paper that was published back in 2005, which goes to show how subtle these mistakes can be and how long it sometimes takes to find them. It is important to note that the problem reviewed in the paper is not a mere arithmetic error but an issue concerning definability in arithmetics.

Luckily, there are tools, so-called proof assistants, that can help eliminate human error by shifting the task of checking the correctness of mathematical assertions from humans to software. Furthermore, proof assistants can be utilized in formally verifying correctness of software's behavior, which is essential to maximize security guarantees. A case in point is CompCert [4], a formally verified optimizing compiler for C, which is intended to be used in software where high levels of assurance are required.

One particularly interesting field of mathematics that has not been formalized in any existing proof assistant is definability theory. It provides tools for reasoning about the structure of mathematical objects and their relationships with other objects. Moreover, it is closely related to decidability theory, a fundamental tool that facilitates the development of efficient algorithms and systems.

At the time of writing, Lean's [5] community is actively working on formalizing definability theory, and the most fundamental definitions and theorems have already been implemented. However, the notions of definability of predicates are still absent. Therefore, our objective is to formalize some definitions of predicate definability in Lean.

1 Problem definition

The goal of this work is to formalize a number of notions concerning first-order definability of predicates in Lean. To achieve this goal, the following tasks were set.

1. Formalize the notion of arithmetic languages and arithmetic structures
2. Prove correctness of these definitions
3. Formalize the notion of definability of predicates in arithmetic structures
4. Prove definability preservation theorem

2 Related works

In this section we will discuss existing related solutions.

2.1 Lean

Lean [5] is a functional programming language that allows for writing correct and maintainable code. It is based on the calculus of constructions with inductive types and is often used as a theorem prover. Lean's features include:

- Type inference;
- Dependent types;
- Metaprogramming framework;
- Multithreading;
- Verification;
- and more.

Lean also has an extensive user-maintained library, mathlib [8], which contains plenty of mathematical definitions and theorems from many different areas of mathematics, including algebra, probability, model theory and more, as well as tactics that provide users with efficient tools for theorem proving.

2.2 Model theory in mathlib

At the time of writing, mathlib's community is actively working on implementing model theory concepts in Lean. The following constructs are currently available:

- first-order languages;
- structures;

- models;
- definability of sets;
- satisfiability;
- etc.

mathlib defines languages as a pair $(functions, relations)$ without constants, and structures as a pair (fun_map, rel_map) which determine interpretations of each symbol. Although definability of sets is implemented, notions of definability of predicates are not. Definability of predicates, particularly in arithmetic structures, as we will later cover, allows us to reason about their expressive power, and providing formalized tools for such reasoning is the fundamental difference of this work from existing solutions. Furthermore, it is important that it is easy to construct and reason about *different* arithmetic structures [2].

2.3 igl2020

In 2020 a group of students from University of Illinois started a project called *igl2020* [7] that aimed to formalize *model theory* in Lean. The project was not completed, but numerous definitions were implemented, including the ones that we need in our work. Within the project, the team successfully implemented the following notions:

- languages;
- structures;
- terms;
- formulas and sentences;
- etc.

Unlike mathlib, languages and structures in igl2020 are defined as triples (F, R, C) and their interpretations accordingly, including constants.

3 Implementation

In this section we discuss the implementation of definability of predicates in arbitrary structures and arithmetic structures and introduce definability preservation theorem.

3.1 Definability in arbitrary structures

Let us first introduce definitions that will be used in this section.

A first-order language L, as it is defined in igl2020, is a tuple (F, R, C) where F is a set of function symbols, R — a set of predicate symbols and C — a set of constants. An L-structure M is a tuple $(fun\_map, rel\_map, const)$ which determines interpretations of each symbol.

A predicate is a function $\mathbb{N}^n \to \{\top, \bot\}$.

Let L be a first-order language and M — an L-structure. We say that a predicate p is definable in the structure M if and only if there exists an L-formula $\phi(x_1, ..., x_n)$ such that for every tuple $\overline{a} \in \mathbb{N}^n$ we have $\phi(\overline{a})$ if and only if $p(\overline{a})$. Symbolically, this means

$$\text{Def}(p, M) \Leftrightarrow \exists(\phi : L\text{-formula})\ (\forall(va : \mathbb{N} \to M.univ)\ (va \models \phi \Leftrightarrow va \models p))$$

where $\text{Def}(p, M)$ is a notion for «p is definable in M» and va is a variable assignment function. Note that we define definability in a structure and not in a language since we rely on interpretation.

We now need to clarify what predicates are in terms of Lean and igl2020. In view of the fact that igl2020 already has definitions of relations and formulas, defining a structure `predicate` is redundant. It quickly becomes clear that defining predicates as relations introduces too many unnecessary difficulties. The reason for this is the fact that in order to interpret a given predicate we always have to require a structure in which it is interpreted. That might not seem as a crucial problem at first glance, but if we go further and try to define a set of all definable predicates in some structure, we see the first serious challenge. Mathematically, $\text{Def}(M)$ (the set of all predicates

that are definable in a given structure M) can be defined as follows:

$$\text{Def}(M) := \{p \mid \text{Def}(p, M)\}$$

Since p is a relation, it could be rewritten as $L.R\ n$, meaning an L-relation of arity n. It is cumbersome to implement in Lean for several reasons.

1. Language L is unknown at the time of constructing this set. Fixing the language prior to construction limits the set, so this is not a correct approach.

2. Even if we manage to somehow construct a set of predicates defined in this way, we obtain a set of relations from arbitrary languages, which complicates the process of using and extracting valuable information from it.

Using formulas instead we can overcome some of these difficulties. However, the set $\text{Def}(M)$ is still heterogeneous and it contains formulas of different languages. This can be fixed by limiting the class of considered structures.

3.2 Arithmetic languages and structures

Arithmetic structures are structures, where relations are defined via arithmetic formulas. This fact allows for easy reasoning about such structures. We introduce the following definitions:

1. Arithmetic language is a pair (n, ar) where n is the number of relations in a language and $ar : \{0, 1, ..., n-1\} \to \mathbb{N}+$ is a mapping which specifies the arity of each relation;

2. Arithmetic structure is a pair $(rels, ar\_proof)$ where $rels$ is a set of basic relations and ar_proof is a certificate that proves that the arity of each relation is correct according to the given signature.

Or in terms of Lean see Figure 1 and Figure 2.

```
structure arith_lang : Type 1 :=
  (n : ℕ+ )           -- number of relations
  (ar : fin n → ℕ+)   -- arity of each relation
```

Figure 1: Arithmetic language

```
structure arith_struc (L : arith_lang) :=
  (rels : vector (formula ordered_semiring_lang) L.n)
  (ar_proof : ∀ i, formula.count_free_vars_list (rels.nth i) = L.ar i)
```

Figure 2: Arithmetic structure

Relations in *rels* are defined by using formulas of `ordered_semiring_lang` language. Its signature, $\langle 0, 1, +, *, < \rangle$, specifies two constants, two binary function symbols and one binary predicate symbol.

Let us show that these definitions are correct, that is, prove that arithmetic languages are indeed languages and arithmetic structures, in turn, are structures. Indeed, arithmetic languages are languages without constants and function symbols, but with n predicate symbols of arity specified by ar mapping. Next, arithmetic structures are structures without constants and functions and with n relations. Since relations are specified using formulas, we will consider that

$$v \in (S.R\ i) \Leftrightarrow v \models \phi_i$$

where $v : \mathbb{N}^n$ is an assignment to the variables, S is an arithmetic structure, $S.R\ i$ — a relation in this structure and ϕ_i — formulas that specifies this relation. Considering the fact that our structures are arithmetic structures, we will evaluate ϕ_i on the set v using a simpler structure $\langle \mathbb{N}; 0, 1, +, *, < \rangle$.

3.3 Definability in arithmetic structures

Now that we have limited the class of structures, the definition of predicate definability does not change much (see Figure 3).

However, now we are able to define $\mathrm{Def}(M)$ (see Figure 4). The resulting set is no longer heterogeneous because it only consists of formulas of the

```
def predicate_is_definable_in_arith_struc
   {L : arith_lang} (S : arith_struc L)
   (pred : formula ordered_semiring_lang) :=
  ∃ φ : formula L, ∀  va : ℕ  → ℕ, va ⊨ φ ↔ va ⊨ pred
```

Figure 3: Predicate definability in arithmetic structures

fixed language `ordered_semiring_lang`. This also means that we know how to interpret all of those formulas (that is, with a $\langle 0, 1, +, *, < \rangle$-structure). Now every predicate is defined via this formula and the structure $\langle \mathbb{N}; 0, 1, +, *, < \rangle$.

```
def definable_predicates {L : arith_lang} (S : arith_struc L)
   : set (formula ordered_semiring_lang) :=
   { φ : formula ordered_semiring_lang |
   predicate_is_definable_in_arith_struc S φ }
```

Figure 4: Set of definable predicates in arithmetic structures

3.4 Definability preservation theorem

Now we have all necessary prerequisites to prove the following theorem.

Theorem. *Let S_1, S_2 be arithmetic structures such that every basic predicate of S_1 is definable in S_2. Then the set of all predicates definable in S_2 the set of predicates definable in S_1. Or symbolically,*

$$(\forall p \in S_1.rels\ (p \in \mathrm{Def}(S_2))) \Rightarrow \mathrm{Def}(S_1) \subseteq \mathrm{Def}(S_2)$$

Proof. Proof of this theorem on paper is quite straight-forward. However, Lean's strict type system enforces a more formal and detailed proof. Let ψ be a formula such that

$$\forall va\ (va \models \psi \Leftrightarrow va \models \phi)$$

for some basic formula ϕ from $S_1.rels$. Then the proof can be conducted by induction on ψ. Let us consider the most interesting base case: $\psi =$

$S_1.rel\ \psi_\alpha$ for some ψ_α. This is exactly when we use the hypothesis from the theorem statement that every S_1's basic relation is definable in S_2.

As for induction cases, it is easy to prove that if ψ_1 and ψ_2 are definable in some structure, then so are

- $\psi_1 \wedge \psi_2$;
- $\psi_1 \vee \psi_2$;
- $\neg\psi_1$;
- $\exists x\ \psi$;
- $\forall x\ \psi$.

A sketch of the proof in Lean can be seen in Figure 5. We start by extracting the formula ψ as discussed above and proceed to use induction. The proof of the base case mentioned above is essentially an `if-then-else` statement: if ψ is a basic predicate of S_1, then we use the theorem hypothesis, otherwise it is obvious that $\psi = \bot$. □

```
theorem subset_if_predicates_definable {L₁ L₂ : arith_lang} (S₁ :
   arith_struc L₁) (S₂ : arith_struc L₂) :
  (∀ pred ∈ S₁.rels.to_list, predicate_is_definable_in_arith_struc S₂
   pred) → definable_predicates S₁ ⊆ definable_predicates S₂ :=
begin
  intros h₁ φ h₂,
  simp only [definable_predicates, set.mem_set_of] at h₂ ⊢,

  cases h₂ with ψ ψ_h,
  induction ψ generalizing φ,
  ...
  { have em := em ψ(_n ∈ (vector.of_fn L₁.ar).to_list),
    cases em,
    {
      have g := rel_to_formula S₁ ψ_α ψ_α_1 (by {
        simp at em,
        exact em
      }),
      rcases g with ⟨g_w, g_h₁, g_h₂⟩,

      have h₂ := h₁ g_w g_h₁,
      cases h₂ with γ γ_h,
      use γ,

      intro va,
      simp only γ[_h, ← g_h₂, ψ_h] },
    { use ⊥',
      have h₂ : formula.rel ψ_α ψ_α_1 = ⊥',
      simp ←[ ψ_h, h₂] } }
end
```

Figure 5: Proof of definability preservation theorem

Conclusion

In conclusion, let us summarize the main results of this work.

1. Notion of arithmetic languages and arithmetic structures was formalized in Lean
2. Correctness of introduced definitions was proven.
3. Notion of predicate definability was implemented.
4. Definability preservation theorem was proven.

The code of the formalization is available at GitHub [1].

As for the prospects, we are planning to allow arithmetic structures to also have function symbols whose graphs are arithmetical predicates, since limiting arithmetic structures to only have relations was a simplification.

Another direction one might consider is generalizing notions of definability of predicates to arbitrary structure and/or proving more fundamental lemmas and theorems about arithmetic structures. Specifically, we hope to formally prove theorem 2 of [6].

References

[1] URL: https://github.com/airh4ck/igl2020/.

[2] Bès Alexis. A Survey of Arithmetical Definability // HAL. — 2002. — Vol. 2002, no. 0. — URL: http://dml.mathdoc.fr/item/hal-00091580.

[3] Bozga Marius, Iosif Radu. On Decidability Within the Arithmetic of Addition and Divisibility // Foundations of Software Science and Computational Structures / Ed. by Vladimiro Sassone. — Berlin, Heidelberg : Springer Berlin Heidelberg, 2005. — P. 425–439.

[4] CompCert. — URL: https://compcert.org/compcert-C.html.

[5] Lean. — URL: https://leanprover.github.io/.

[6] Pérez Guillermo A., Raha Ritam. Revisiting Parameter Synthesis for One-Counter Automata. — Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022. — URL: https://drops.dagstuhl.de/opus/volltexte/2022/15753/.

[7] igl2020. — URL: https://github.com/vaibhavkarve/igl2020.

[8] mathlib. — URL: https://github.com/leanprover-community/mathlib.