Ax-Grothendieck and Lean

Joseph Hua

December 20, 2021

Contents

1 Introduction **1**

2 Model Theory Background **1**
2.1 Languages 1

3 Model Theory of Algebraically Closed Fields **3**

4 The Lefschetz Principle **3**

5 Reducing to Locally Finite Fields **3**

6 Proving of the Locally Finite Case **3**

1 Introduction

2 Model Theory Background

For most definitions and proofs in this section we reference David Marker's book on Model Theory [1]. We introduce the formalisations of the content in `lean` alongside the theory.

2.1 Languages

> **Definition – Language**
> A language (also known as a *signature*) $\mathcal{L} = (\mathrm{functions}, \mathrm{relations})$ consists of
>
> - A sort symbol A, which we will have in the background for intuition.
> - For each natural number n we have $\mathrm{functions}\, n$ - the set of *function symbols* for the language of *arity* n. For some $f \in \mathrm{functions}\, n$ we might write $f : A^n \to A$ to denote f with its arity.
> - For each natural number n we have $\mathrm{relations}\, n$ - the set of *relation symbols* for the language of *arity* n. For some $r \in \mathrm{relations}\, n$ we might write $r \hookrightarrow A^n$ to denote r with its arity.
>
> The `flypitch` project implements the above definition as
>
> ```
> structure Language : Type (u+1) :=
>   (functions : ℕ → Type u)
> ```

```
(relations : ℕ → Type u)
```

This says that `Language` is a mathematical structure (like a group structure, or ring structure) that consists of two pieces of data, a map called `functions` and another called `relations`. Both take a natural number and spit out a *type* (think *set* for now) that consists respectively of all the function symbols and relation symbols of arity n.

In more detail: in type theory when we write `a:A` we mean `a` is a *term* of *type* `A`. We can draw an analogy with the set theoretic notion $a \in A$, but types in `lean` have slightly different personalities, which we will gradually introduce. Hence in the above definitions `Language`, `functions n` and `relations n` are terms of type `Type` (something), the latter is the type consisting of all types (with some universe considerations to avoid Russell's paradox), in other words they themselves are *types*.

For convenience we single out 0-ary (arity 0) functions and call them *constant* symbols, usually denoting them by $c : A$. We think of these as 'elements' of the sort A and write $c : A$. This is defined in `lean` by

```
def constants (L : Language) : Type u := functions 0
```

This says that `constants` takes in a language L and returns a type. Following the `:=` we have the definition of `constants L`, which is the type `functions 0`.

Example. *The language of rings will be used to define the theory of rings, the theory of integral domains, the theory of fields, and so on. In the appendix we give examples:*

- *The language with just a single binary relation can be used to define the theory of partial orders with the interpretation of the relation as $<$, to define the theory of equivalence relations with the interpretation of the relation as $\sim$, and to define the theory* ZFC *with the relation interpreted as $\in$.*
- *The language of categories can be used to define the theory of categories.*

We will only be concerned with the language of rings and will focus our examples around this.

Definition – Language of rings

Let the following be the signature of rings:

- The function symbols are the constant symbols $0, 1 : A$, the symbols for addition and multiplication $+, \times : A^2 \to A$ and taking for inverse $- : A \to A$.
- There are no relation symbols.

We can break this definition up into steps in `lean`. We first collect the constant, unary and binary symbols:

```
/-- The constant symbols in RingSignature -/
inductive ring_consts : Type u
| zero : ring_consts
| one : ring_consts

/-- The unary function symbols in RingSignature-/
inductive ring_unaries : Type u
| neg : ring_unaries

/-- The binary function symbols in RingSignature-/
inductive ring_binaries : Type u
| add : ring_binaries
| mul : ring_binaries
```

These are *inductively defined types* - types that are 'freely' generated by their constructors, listed below after each bar '|'. In these above cases they are particularly simple - the only constructors are terms in the type. In the appendix we give more examples of inductive types

- The natural numbers are defined as inductive types
- Lists are defined as inducive types
- The integers can be defined as inductive types

We now collect all the above into a single definition `ring funcs` that takes each natural `n` to the type of `n`-ary function symbols in the language of rings.

```
/-- All function symbols in RingSignature-/
def ring_funcs : ℕ → Type u
| 0 := ring_consts
| 1 := ring_unaries
| 2 := ring_binaries
| (n + 3) := pempty
```

The type `pempty` is the empty type and is meant to have no terms in it, since we wish to have no function symbols beyond arity 2. Finally we make the language of rings

```
/-- The language of rings -/
def ring_signature : Language :=
(Language.mk) (ring_funcs) (λ n, pempty)
```

3 Model Theory of Algebraically Closed Fields

4 The Lefschetz Principle

5 Reducing to Locally Finite Fields

6 Proving of the Locally Finite Case

References

[1] D. Marker. *Model Theory - an Introduction*. Springer.