Ax-Grothendieck and Lean

Joseph Hua

November 11, 2021

Contents

1 Introduction

2 Model Theory Background

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

2.1 Languages

Definition – Language
A language (also known as a *signature*) $\mathcal{L} = (\text{functions}, \text{relations})$ consists of

- A sort symbol A, which we will have in the background for intuition.
- For each natural number n we have $\text{functions}\, n$ - the set of *function symbols* for the language of *arity* n. For some $f \in \text{functions}\, n$ we might write $f : A^n \to A$ to denote f with its arity.
- For each natural number n we have $\text{relations}\, n$ - the set of *relation symbols* for the language of *arity* n. For some $r \in \text{relations}\, n$ 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 := L.functions 0
```

This says that `constants` takes in a language L and returns a type, specifically it returns `functions 0`.

Our languages are multi-sorted, meaning we can have elements and variables living in different spaces. For example groups, rings and partial orders are all 1-sorted with just one underlying set, whereas group actions and modules are 2-sorted. If we are working in the 1-sorted case we may not mention the sort at all, because there is only one.

3 Model Theory of Algebraically Closed Fields

4 The Lefschetz Principle

5 Reducing to Locally Finite Fields

6 Proving of the Locally Finite Case