笨蛋读书笔记

代数与 PL 与 PL 之恋[1]

什么是代数?

代数结构 (Algebraic Structure，有时被称为 Algebra)

代数结构: 长得像 (集合 G × 一组有限个集合上封闭的有限操作数[2]运算 × 一组有限个的公理) 的东西。

它其实是类似 $\{G : \text{Set} \mid \{... \text{ ops} : (G^N \to G) \mid ... \text{ axioms} : \text{equation on G and ops}\}\}$ 的依值积类型 (dependent product type)。

注意到蓝色的部分及 … 符号，这些操作都是由元理论展开的，而不包含在代数结构定义里。

~~或者可以说是在定义一个 Set of (_ × _ × _)? 所以它势必是在描述一堆挂着封闭运算和公理的特殊集合[3]。~~

例子：Group Algebra

所有满足以下要求的(集合 G × 集合上封闭的运算 $\cdot$)的组合称为 Group Algebra:

1. 基集合 G
2. 运算
   a. $\cdot : G \to G \to G$
   b. $e : G$
   c. $- : G \to G$
3. 满足以下公理:
   a. 单位元: $\forall a \in G, e \cdot a = a \cdot e = a$
   b. 逆元: $\forall a \in G, a \cdot -b = b \cdot -a = e$
   c. 结合律: $\forall abc \in G, (a \cdot b) \cdot c = a \cdot (b \cdot c)$

用以下伪 Coq 代码来解释:

```
Definition group: Type := { A: Type &
  { •: A → A → A &
  { id: A &
  { -: A → A &
  { law_assoc: ∀ a b c, a • (b • c) = (a • b) • c &
  { law_id: ∀ a, a • id = a ∧ id • a = a &
  (* law_inv: *) ∀ a, a • -a = id ∧ -a • a = id
  } } } } } }.
```

或者，我们采用一个更直观的定义，让 group 通过它的基集合和运算索引[4]：

```
Definition group (A: Set) (•: A → A → A), (id: A), (inv: A → A): Type :=
  { law_assoc: ∀ a b c, a • (b • c) = (a • b) • c &
  { law_id: ∀ a, a • id = a ∧ id • a = a &
  (* law_inv: *) ∀ a, a • -a = id ∧ -a • a = id }}.
```

然后举一个一个 Group Algebra 的实例的显然例子：(整数, 加法) 群

```
Lemma Z_plus_group: group Z +%Z 0%Z -%Z.
Proof.
```

[1] https://guest0x0.xyz/PL-and-universal-algebra/PL-and-universal-algebra.pdf

[2] 似乎有的时候不要求?

[3] 这里面有 Set of Set 的问题所以显然不太对，我不知道该如何正确形式化但是先如此感性地理解

[4] qlbf: 有一个非平凡的互转，参见 Arend 文档里的 anonymous extension 和 cooltt 的 extension type

```
  exists Z.add_assoc. exists Z.add_id. exact Z.add_inv.
Defined.
```

可见不是所有类型（加上一些操作）都可以满足某个特定代数结构的要求。特定代数结构其实是一种对类型的约束。

自然地，Abstract Data Type 是一种约束类型的方法，它显然可以用来定义特定代数结构（类型满足 Abstract Data Type 约束 $\longleftrightarrow$ 集合满足特定代数结构约束）。由于工程语言表达能力不足，它无法定义等式[5]。游客账户在原文中也提到了一个例子：Module Type。

```
module type Group = sig
  type G
  val comp: G → G → G
  val id: G
  val inv: G → G
end
module Z_plus: Group = struct
  type G = Z
  let comp = Z.add
  let id = Z.zero
  let inv = Z.opp
end
```

泛代数 (Universal Algebra)

泛代数不是一种代数结构，也不研究特定的代数结构，而是研究所有代数结构的领域。换句话说，它开始考虑上述所称的元理论的部分，因此开始研究如何操纵代数结构，例如定义代数结构间的态射，同态等。

TODO: 我不知道怎么在 Coq 里 formalize universal algebra，问题点在于如何 formalize variadic dependent product type。或许其实也不需要 formalize，只要意识到元理论和理论的关联我觉得我就能想明白了。

代数同态

TODO

[5] Coq 应该行，但是我懒得写 x.x