Lecture 2 Notes

Owen Lynch

December 18, 2025

1 Recap from last time

Last week we discussed:

1. The theses behind the geolog research program
2. The "doctrinal" view of categorical logic
3. Theories as functors from the 2-category of models of a doctrine into Cat
4. The doctrine of regular hyperdoctrines

2 Overview of this lecture

1. A syntactic approach to regular theories
2. Finite models of regular theories
3. A type theory for regular theories
4. What the geolog architecture could look like

3 Regular theories and finite models

```
open import Data.Unit
open import Data.Bool using (Bool)
import Data.Bool as Bool
open import Data.Fin using (Fin)
import Data.Fin as Fin
open import Data.Nat
open import Relation.Nullary.Decidable using (Dec)

data Con (S : Set) : Set where
  • : Con S
  _▷_ : Con S -> S -> Con S
```

```
data Var {S : Set} : Con S -> S -> Set where
  q : {A : S} -> {Γ : Con S} -> Var (Γ ▷ A) A
  p_ : {A B : S} -> {Γ : Con S} -> Var Γ A -> Var (Γ ▷ B) A

module RegularLogic (Sort : Set) (Rel : Con Sort -> Set) where
  variable Γ Δ : Con Sort
  variable A B : Sort

  Tm = Var

  Sub : Con Sort -> Con Sort -> Set
  Sub Γ Δ = (A : Sort) -> Tm Δ A -> Tm Γ A

  data Prop : Con Sort -> Set where
    true : Prop Γ
    _$_ : Rel Δ -> Sub Γ Δ -> Prop Γ
    _∧_ : Prop Γ -> Prop Γ -> Prop Γ
    _===_ : Tm Γ A -> Tm Γ A -> Prop Γ
    ∃ : Prop (Γ ▷ A) -> Prop Γ

  record Axiom : Set where
    field
      con : Con Sort
      ante : Prop con
      cons : Prop con

record Sig : Set where
  field
    sorts : Con ⊤
    rels : Con (Con (Var sorts tt))

Sort : (S : Sig) -> Set
Sort S = Var (Sig.sorts S) tt

Rel : (S : Sig) -> Con (Sort S) -> Set
Rel S = Var (Sig.rels S)

record Theory : Set where
  field
    sig : Sig
    axioms : Con (RegularLogic.Axiom (Sort sig) (Rel sig))

Tuple : (S : Sig) -> (Sort S -> ℕ) -> Con (Sort S) -> Set
Tuple S elts Γ = (A : Sort S) -> Var Γ A -> Fin (elts A)

record ModelData (S : Sig) : Set where
```

```
  field
    elts : Sort S -> ℕ
    sat : {Γ : Con (Sort S)} -> Rel S Γ -> Tuple S elts Γ -> Bool

module Interp (S : Sig) (M : ModelData S) where
  open RegularLogic (Sort S) (Rel S)

  t = Tuple S (ModelData.elts M)

  Elt : Sort S -> Set
  Elt A = Fin (ModelData.elts M A)

  extend : (Γ : Con (Sort S)) -> t Γ -> (A : Sort S) -> Elt A -> t (Γ ▷ A)
  extend = {!!}

  any : {n : ℕ} -> (Fin n -> Bool) -> Bool
  any = {!!}

  allTuples : (Γ : Con (Sort S)) -> (t Γ -> Bool) -> Bool
  allTuples = {!!}

  eval : {Γ : Con (Sort S)} -> {A : Sort S} -> Tm Γ A -> t Γ -> Elt A
  eval {Γ} {A} v f = f A v

  evalSub : {Γ Δ : Con (Sort S)} -> Sub Γ Δ -> t Γ -> t Δ
  evalSub s f A v = f A (s A v)

  interp : {Γ : Con (Sort S)} -> Prop Γ -> t Γ -> Bool
  interp true _ = Bool.true
  interp (r $ xs) f = ModelData.sat M r (evalSub xs f)
  interp (ϕ ∧ ψ) f = interp ϕ f Bool.∧ interp ψ f
  interp (x₁ === x₂) f = Dec.does (eval x₁ f Fin.≟ eval x₂ f)
  interp (∃ {Γ} {A} ϕ) f = any (λ i -> interp ϕ (extend Γ f A i))

  satisfies : Axiom -> Bool
  satisfies a = allTuples
    (Axiom.con a)
    (λ f -> Dec.does (interp (Axiom.ante a) f Bool.≤? interp (Axiom.cons a) f))
```

4 A type theory for regular logic

The query engine for geolog will probably accept an IR format that looks something like an extended version of the above. However, for the user we would like to have the ability to define higher-level abstractions.

These higher-level abstractions take the form of a type theory.

To develop this type theory, we will first look to denotational semantics. The premise here is that the denotational semantics will give us a first pass at figuring out which rules are admissable in the type theory; we will then have to refine the type theory in order to

1. Make elaboration feasible (which will involve switching to intensional equality)
2. Allow the user more control over operational semantics

In the denotational semantics, each context in the type theory will correspond to a *semantic regular theory*.

Definition 4.1. A **semantic regular theory** consists of a representable 2-functor from Reg, the 2-category of regular categories, to Cat.

If $\Gamma\colon \mathsf{Reg} \to \mathsf{Cat}$ is a representable 2-functor, we will denote its representing object by $S[\Gamma]$.

Substitutions correspond to 2-natural transformations.

A theory T in a context Γ corresponds to a representable functor from $\int \Gamma$ to Cat. A model $t\colon T$ is a section of that functor, e.g. $t((C, M)) \in T((C, M))$ for each $(C, M) \in \int \Gamma$.

We can thus start building up rules.

Rule 4.2 (The object classifier)**.**

$$\vdash \mathsf{Type\ theory}$$

This is the forgetful functor $\mathsf{Reg} \to \mathsf{Cat}$. It is representable by $\mathsf{Fin}^{\mathrm{op}}$.

Rule 4.3 (Theory of an element)**.**

$$\frac{\Gamma \vdash A\colon \mathsf{Type}}{\Gamma \vdash \mathsf{Elt}(A)\ \mathsf{theory}}$$

We have $A(C, M) \in C$ for all $(C, M) \in \int \Gamma$; $\mathsf{Elt}(A)(C, M) = C(1, A(C, M))$. It is representable by $S[\Gamma]/A(S[\Gamma], M[\Gamma])$.

Rule 4.4 (The proposition classifier)**.**

$$\vdash \mathsf{Prop\ theory}$$

This is the functor $\mathsf{Reg} \to \mathsf{Cat}$ that sends C to $\mathsf{Sub}(1)$. It is representable by $\mathbf{2}$, the walking arrow.

Rule 4.5 (Propositions as types)**.**

$$\frac{\Gamma \vdash P\colon \mathsf{Prop}}{\Gamma \vdash \mathsf{Proof}(P)\colon \mathsf{Type}}$$

Any subobject of 1 is also an object.

Rule 4.6 (Sigma types)**.**