A Geolog Prospectus

Owen Lynch

December 18, 2025

1 Introduction

1.1 Problem statement

The architecture for the Safeguarded AI project requires a system that

- records
  - observations from "the world",
  - proof certificates for the consequences of those observations according to world-models,
- via
  - an immutable log of observations and consequences,
- and supports
  - querying (and incremental querying),
  - updating with more observations, possibly concurrently,
  - updating with more consequences, also concurrently,
  - handling of conflicting observations,
  - migration of existing facts to new ontologies.

These requirements are neither satisfied by existing proof assistants (which do not manage the evolution of knowledge over time) or existing databases (whose relational foundations do not support the essentially *inductive* nature of proof certificates). Thus, a new system is needed, which will blend aspects of

- proof assistants
- databases
- version control systems

We refer to such a system as a "logical database."

Problem 1.1. Build a logical database.

1.2 Idea

We propose that *geometric theories* provide an appropriate *semantic foundation* for a logical database which satisfies the requirements of the above problem statement.

Definition 1.2 (Johnstone [2002], B4.2)**.** Let CAT be the 2-category of large categories, and for a fixed topos $\mathcal{S}$ let $\mathsf{BTop}/\mathcal{S}$ be the 2-category of bounded topoi over $\mathcal{S}$. A **geometric theory** is a 2-functor $T\colon (\mathsf{BTop}/\mathcal{S})^{\mathrm{op}} \to \mathsf{CAT}$ that may be built up by a sequence of product-inserter-equifier limits, starting from a functor $T_0\colon (\mathsf{BTop}/\mathcal{S})^{\mathrm{op}} \to \mathsf{CAT}$ given by $T_0(\mathcal{E}) = \mathcal{E}^n$ for some finite n.

Intuitively, a geometric theory is something that assigns to each topos a category of "models" of that geometric theory, where a model is some collection of objects, morphisms, and equalities between morphisms that can be expressed using the properties of a topos that are stable under geometric morphism (finite limits and arbitrary colimits).

There are various ways of syntactically expressing geometric theories. The classical way is via first-order logic with an infinitary disjunction; this infinitary disjunction is obviously unsuitable for computer implementation. An alternative approach would be to use a type theory which supports quotient inductive-inductive types; these quotient inductive-inductive types would express most colimits that one cares about in practice.

Geometric theories seem like a good foundation for a couple reasons.

- Intuitively, geometric theories capture the largest collection of types which are "internally serializable." That is, while one can "serialize" a closure via storing its source code and environment, this is dependent on the implementation details of the language and not just its abstract semantics. Of course, serializing an element of a quotient type is similarly suspect, but *equality proofs* for a quotient inductive-inductive type are also inductive, so it is not as much of a problem.
- There are a lot more geometric morphisms than there are logical morphisms; this means that there is significantly more freedom to update the database schema while preserving existing facts and derivations.
- The fact that proofs of geometric propositions are stable under geometric morphisms means that there are a lot of opportunities for *incrementalization*.

It will no doubt be necessary to "pare back" geometric theories in order to make concessions to implementation, but we believe that geometric theories give us a sufficiently general starting point that we will not need to rethink semantic foundations.

1.3 Architecture

The architecture for the software implementation of geolog can roughly be split into three parts:

1. An **elaborator** for a type theory; this type theory serves as the user interface for
    - Schemas
    - Queries
    - Updates
    - Migrations
2. An **operation log** which stores a (not-necessarily linear) history of all operations performed on a database instance.
3. A **query engine** which
    - Accepts some low-level language produced by the elaborator, an operation log, and a specific point of time
    - Executes that low-level language in order to
        - Answer queries, possibly in an *incremental* fashion as new operations are streamed in
        - Create new operations in the log

Such a software would perform **semi-automated** reasoning. That is, some questions about a database would be answerable via polynomial-time "querying", while other questions would only be answerable via proof search, which would be performed by a collaboration of

- humans
- classical "solvers"
- large language models

1.4 Related work

Geolog is a quite ambitious project. However, we believe that it is feasible to do *without developing a large amount of new mathematics or computer science*. That is, we believe that geolog is mainly an *engineering* challenge, and very few substially new

- theorems
- algorithms
- data structures

will be required in the path towards a functional prototype. Rather, we believe that a careful reading of the literature will provide us with most if not all of the tools we need to put together a full system.

In terms of semantics, the two main subjects that are of relevance are

1. geometric theories,
2. finite model theory, including finite model theory for least fixed point logics.

Each of these subjects has been well-studied for decades, relevant works include Johnstone [2002] for geometric theories and Libkin [2004] for finite model theory. One complication is how semantically to model sparsity of, for instance, a function $\mathsf{Nat} \to \mathsf{Prop}$, which in a finite model should be a finite set of natural numbers; we will have to find relevant literature for this.

The type theory is slightly more experimental, however it draws on a variety of pre-existing work such as

1. Logical frameworks (Pfenning and Schürmann [1999]).
2. Quotient inductive-inductive types (Kovács [2022])
3. Modulog (Atkey [2018])

Moreover, as far as we can see, the type theory will be completely structural and non-modal, and hence expressible as a SOGAT (Kaposi and Xie [2024]); this will simplify the mathematical treatment. As far as the implementation goes, a standard bidirection elaboration with normalization-by-evaluation will suffice. Equality, as always, will present a complication. One approach would be the standard intensional equality type. However, there are some significant advantages to working in the restricted setting of geometric theories comparted to full MLTT which may make an observational approach to equality feasible (Altenkirch and McBride [2006]), with attendant ergonomic benefits.

The query engine is a part that the author of this present document has less familiarity with, beyond a "computer science" understanding of algorithmic complexity. Nevertheless, the main jobs of the query engine are

1. Instantiation of the operations in the operation log into sparse tensor data structures
2. Incremental evaluation of conjunctive queries
3. Datalog-style least fixed point problems
4. Egglog-style mixtures of equality saturation and least fixed point reasoning (Zhang et al. [2023])

I believe that the first three items are fairly well-studied; the last one less so. But the "automated reasoning" capabilities of geolog are less important than its role as a "system of record," as we expect more specialized systems to be performing reasoning in any serious domain; for instance reachability in a Petri net would be performed by an external tool which would then export its chain of reasoning into geolog.

Finally, the "operation log" part of geolog is intended to be very similar to current data structures inside automerge (Automerge contributors [2025]),

and we expect that (from an operation log point of view) the kind of syncing/collaboration in geolog is not terribly different from the way automerge deals with operation logs for json documents. The main difference will be in the "instantiator" which is the job of the query engine.

2 Geolog by example

In this section, we give some "dream code" for what the geolog type theory might look like in practice. The intention here is for the reader to get a "vibe" for what geolog might eventually look like, not to fully understand at the current moment what geolog is.

2.1 Combinatorial data structures

Example 2.1 (Graphs)**.**

```
theory SimpleDirectedGraph
  V : Type
  E : V -> V -> Prop
end

instance TwoPathGraph (G : SimpleDirectedGraph) : SimpleDirectedGraph
  V = G.V
  E x z = exists (y : V) (G.E x y && G.E y z)
end
```

`TwoPathGraph` would be treated as a *migration*; from an instance of `SimpleDirectedGraph` it would produce another instance in which there is an edge from `x` to `z` if and only if there exists a length-2 path from `x` to `z` in the original graph.

Example 2.2 (Petri nets)**.**

```
theory PetriNet
  Place : Type
  Transition : Bag Place -> Bag Place -> Type
end

local theory Reachability (P : PetriNet)
  open P

  Reachable : Bag Place -> Bag Place -> Prop
  refl : (B : Bag Place) -> Reachable B B
  applyTransition : (X Y Z1 Z2 : Bag Place) -> Reachable X (Y + Z1) ->
    Transition Z1 Z2 -> Reachable X (Y + Z2)
end
```

A local theory is a theory which is syntactically restricted so that it has an initial instance; for instance there is an initial model of `Reachability P` for every petri net `P`. This initial model is clearly not finite, but the database can nonetheless store "partial knowledge" about the relation `Reachable` in this initial model; e.g. proofs of reachability for certain pairs of token assignments.

Example 2.3 (Simplicial sets)**.**

```
theory SemiSimplicialSet
  Simplex : Nat -> Type
  -- we use ?n for implicit parameters
  d : (?n : Nat) -> Fin (S ?n) -> Simplex (S ?n) -> Simplex ?n
  dd : (n : Nat) -> (s : Simplex (S (S n))) ->
    (i : Fin (S n)) -> (j : Fin (S n)) -> d i (d (j + 1) s) = d j (d i s)
end
```

This is only well-typed if we have an implicit coercion from `Fin n` to `Fin (S n)`, which we likely do not want, but it makes it easier to read in the example.

2.2 Relational databases

Example 2.4 (Parts and projects)**.**

```
type Metadata = [name : String, description : String]

theory PartsAndProjects
  Part : Type

  properties Part
    meta : Metadata
    quantity-on-hand : Nat
    quantity-on-order : Nat
  end
  -- syntactic sugar for:
  -- Part/meta : Part -> Metadata
  -- Part/quantity-on-hand : Part -> Nat
  -- Part/quantity-on-order : Part -> Nat

  Project : Type

  properties Project
    meta : Metadata
  end

  Commit : Type

  properties Commit
```

```
    part : Part
    project : Project
    quantity : Nat
  end
end
```

This is the classic example from the paper that started it all Codd [1970].

2.3 Program analysis

Many program analysis tasks are expressible in datalog. Bob Atkey's modulog Atkey [2018] contains a bunch of good examples.

Example 2.5 (Interprocedural Finite Distributive Subset)**.** This is a translation of a modulog program which is aimed at analyzing control flow across procedures.

```
theory IFDS_Program
  Procedure : Type
  Point : Type
  EntryFact : Type
  Fact : Type

  EntryState : Type = [Procedure, EntryFact]
  State : Type = [Point, Fact]

  Edge : State -> State -> Prop
  StartEdge : EntryState -> State -> Prop
  Call : State -> EntryState -> Prop
  Return : State -> State -> State -> Prop
  Initial : EntryState -> Prop
end

local theory IDFS (P : IDFS_Program)
  open P

  Intra : EntryState -> State -> Prop
  CallSite : State -> EntryState -> Prop
  Invoked : EntryState -> Prop
  CallReturn : State -> State -> Prop

  _ : Invoked ?es -> StartEdge ?es ?s -> Intra ?es ?s
  _ : Intra ?es ?s' -> Edge ?s' ?s -> Intra ?es ?s
  _ : Intra ?es ?s' -> CallReturn ?s' ?s -> Intra ?es ?s

  _ : Initial ?es -> Invoked ?es
  _ : CallSite ?s ?es -> Invoked ?es
```

```
  _ : Intra ?es ?s -> Call ?s ?es' -> CallSite ?s ?es'

  _ : CallSite ?s ?es -> Intra ?es ?s' -> Return ?s ?es ?s' -> CallReturn ?s ?s'
end
```

The input to this would be an instance `P : IFDS_Program`, and then automated reasoning would answer questions about the "initial instance" of `IDFS P`.

References

Thorsten Altenkirch and Conor McBride. Towards Observational Type Theory. 2006. URL `http://strictlypositive.org/ott.pdf`.

Bob Atkey. Modulog, 2018. URL `https://github.com/bobatkey/modulog`.

Automerge contributors. Automerge, 2025. URL `https://automerge.org/`.

E. F. Codd. A relational model of data for large shared data banks. *Communications of the ACM*, 13(6):377–387, 1970. doi: 10.1145/362384.362685.

P. T. Johnstone. *Sketches of an Elephant: A Topos Theory Compendium*. Number 43 in Oxford Logic Guides. Oxford University Press, Oxford ; New York, 2002. ISBN 978-0-19-852496-0 978-0-19-853425-9 978-0-19-851598-2.

Ambrus Kaposi and Szumi Xie. Second-Order Generalised Algebraic Theories: Signatures and First-Order Semantics. In Jakob Rehof, editor, *9th International Conference on Formal Structures for Computation and Deduction (FSCD 2024)*, volume 299 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 10:1–10:24, Dagstuhl, Germany, 2024. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. ISBN 978-3-95977-323-2. doi: 10.4230/LIPIcs.FSCD.2024.10.

András Kovács. *Type-Theoretic Signatures for Algebraic Theories and Inductive Types*. PhD thesis, Eötvös Loránd Tudományegyetem, 2022.

Leonid Libkin. *Elements of Finite Model Theory*. Springer Berlin Heidelberg, Berlin, Heidelberg, 2004. ISBN 978-3-642-05948-3 978-3-662-07003-1. doi: 10.1007/978-3-662-07003-1.

Frank Pfenning and Carsten Schürmann. *System Description: Twelf — A Meta-Logical Framework for Deductive Systems*, volume 1632, pages 202–206. Springer Berlin Heidelberg, Berlin, Heidelberg, 1999. ISBN 978-3-540-66222-8 978-3-540-48660-2. doi: 10.1007/3-540-48660-7_14.

Yihong Zhang, Yisu Remy Wang, Oliver Flatt, David Cao, Philip Zucker, Eli Rosenthal, Zachary Tatlock, and Max Willsey. Better Together: Unifying Datalog and Equality Saturation. *Proceedings of the ACM on Programming Languages*, 7(PLDI):468–492, June 2023. ISSN 2475-1421. doi: 10.1145/3591239.