Correct-by-construction programming with generative language models

PAMLTP and DG4D^3
Prague, 19 April 2023

Maximilian Doré
`maximilian.dore@cs.ox.ac.uk`

Background

- Project: showing correctness of some algorithms in topological data analysis.
- Goal is to use correct-by-construction paradigm for mathematical programs: develop theory and code in the same language.
- Combines two laborious endeavours:

  formalization (of a theory) & **verification** (of a program)

Correct-by-construction

Correct-by-construction paradigm:
Specification of a program is part of the language, and compiler ensures that program satisfies specification.

Correct-by-construction

Correct-by-construction paradigm:
Specification of a program is part of the language, and compiler ensures that program satisfies specification.

In dependent type theory, this takes the following form:

- given an input $x : X$,
- the program $p : X \to Y$ computes an output $p\ x$,
- which satisfies the specification $\mathsf{Spec}\ x\ (p\ x)$.

In summary, we want a term of type

$$\Pi X, \Sigma Y, \mathsf{Spec}\ X\ Y$$

A correct-by-construction powerset

Example: Compute the powerset in corr-by-constr fashion.

Idea for how to compute the powerset, in Haskell:

```
powerset :: [a] -> [[a]]
powerset [] = [[]]
powerset (x:xs) = powerset xs ++ map (x:) (powerset xs)
```

We want this program to coincide with the usual definition of powerset, which is our *specification*:

$$P(X) = \{Y \mid Y \subseteq X\}$$

Goal: Construct a program in Agda analogous to `powerset` which provably satisfies the above definition.

Powerset in Agda

From lists to sets: Given base type A with total ordering. Then sets are ordered lists:

$$\{1,2,3\} = 1 < 2 < 3$$

With an apt ordering on sets, we can also define families of sets:

$$\{\{2,3\},\{1,2,3\}\} = (2 < 3) \ll (1 < 2 < 3)$$

Steps:

- Program $\mathsf{powerset} : \mathsf{set}\ A \to \mathsf{set}\ (\mathsf{set}\ A)$.
  - Give functional program analogous to Haskell code.
  - Prove that powerset produces an ordered list.
- Prove that any Y computed by $\mathsf{powerset}\ X$ is subset of X.
- Prove that every subset of X is computed by $\mathsf{powerset}\ X$.

Then we have a term of type

$$\Pi(X : \mathsf{set}), \Sigma(P : \mathsf{family}), \Pi(Y : \mathsf{set}), Y \in P \simeq Y \subseteq X$$

Correctness of powerset function

```
powerset-corr : {xs : List carrier} (ds : ordered xs)
  → {ys : List carrier} (es : ordered ys)
  → (ys , es) ∈l powerset xs ds → (ys , es) ⊆ (xs , ds)
powerset-corr {[]} ds {ys} es P = subst (_⊆l []) (sym ys≡[]) (⊆l-refl []) where
  ys≡[] = toList≡ (∈l-singl-extract P)
powerset-corr {x :: xs} ds {[]} es P = []⊆l-all
powerset-corr {x :: xs} ds {y :: ys} es P
  with ++-dec discreteSet _ (powerset xs (⊏-tails ds)) (powerset-insert x xs ds) P
... | inl Q = ⊆l-weaken IH where
  IH : (y :: ys , es) ⊆ (xs , ⊏-tails ds)
  IH = powerset-corr (⊏-tails ds) es Q
... | inr Q = subst (_⊆l (x :: xs)) (cong (_:: ys) (sym headLemma)) (⊆l-insert x IH)
  where
  tailLemma : (ys , ⊏-tails es) ∈l powerset xs (⊏-tails ds)
  tailLemma = (insertL-tail es (powerset xs (⊏-tails ds)) (x⊏Lpowerset x _ ds) Q)
  IH : (ys , ⊏-tails es) ⊆ (xs , ⊏-tails ds)
  IH = powerset-corr (⊏-tails ds) (⊏-tails es) tailLemma
  headLemma : y ≡ x
  headLemma = insertL-head es (powerset xs (⊏-tails ds)) (x⊏Lpowerset x _ ds) Q
```

In Lean

```
def powerset (s : finset α) : finset (finset α) :=
s.1.powerset.pmap finset.mk $λ
 t h, nodup_of_le (mem_powerset.1 h) s.nodup,
s.nodup.powerset.pmap $ λa ha b hb, congr_arg finset. val

@[simp] theorem mem_powerset {s t : finset α} :
  s ∈ powerset t ↔s ⊆t :=
by cases s;
simp only [powerset, mem_mk, mem_pmap, mem_powerset,
           exists_prop, exists_eq_right];
rw ←val_le_iff
```

Comparing the proofs in Agda and Lean

- Lean code way shorter (3 lines vs $\sim$ 200 lines).
- Lean has tactics (+ black magic).
- Agda code is like a Haskell program, also proofs are manipulated in a functional style.

Speculations

- LLMs don't seem good at reasoning – but they are good at pattern matching and dealing with unstructured data.
- Working in Agda is tedious: have to write a lot of code. Advantage for machine learning?
  With granular enough level of detail proofs/programs, finding "templates" and gluing them together might be a more reliable method than randomly trying tactics.
- **Goal**: Integrate LLM in Agda mode.
  - Typecheck output of LLM directly, if type-checking fails add error-message to context and run LLM again.
  - Synthesize both proofs (terms) and conjectures (types).