Lessons learnt from formalizing theoretical computer science

Martin Dvorak

March 13, 2024

What is Lean

- Lean 4 is a powerful programming language
- Emphasis on formal verification
- Type system based on the Calculus of Inductive Constructions
- Large library of formally-verified mathematics (over 10^5 lemmas)

Showcase in VS Code

Showcase in VS Code

The rest is all about my mistakes

Showcase in VS Code

The rest is all about my mistakes (and what I learnt from them

Showcase in VS Code

The rest is all about my mistakes (and what I learnt from them (sometimes))

Trying to prove a statement that doesn't hold

Trying to prove a statement that doesn't hold

Yeah, duh!

Grammars are closed under concatenation

Grammars are closed under concatenation

Construction:

$$\begin{aligned}
G_1 &= (N_1, T, P_1, S_1) \\
G_2 &= (N_2, T, P_2, S_2) \\
G &= (N_1 \cup N_2 \cup \{S\},\ T,\ P_1 \cup P_2 \cup \{S \to S_1 S_2\},\ S)
\end{aligned}$$

Grammars are closed under concatenation

Construction:

$$\begin{aligned} G_1 &= (N_1, T, P_1, S_1) \\ G_2 &= (N_2, T, P_2, S_2) \\ G &= (N_1 \cup N_2 \cup \{S\},\ T,\ P_1 \cup P_2 \cup \{S \to S_1 S_2\},\ S) \end{aligned}$$

Counterexample:

$$\begin{aligned} P_1 &= \{S_1 \to S_1 a,\ S_1 \to \epsilon\} \\ P_2 &= \{S_2 \to S_2 a,\ S_2 \to \epsilon,\ aS_2 \to b\} \end{aligned}$$

Grammars are closed under concatenation

Construction:

$$\begin{aligned} G_1 &= (N_1, T, P_1, S_1) \\ G_2 &= (N_2, T, P_2, S_2) \\ G &= (N_1 \cup N_2 \cup \{S\},\ T,\ P_1 \cup P_2 \cup \{S \to S_1 S_2\},\ S) \end{aligned}$$

Counterexample:

$$\begin{aligned} P_1 &= \{S_1 \to S_1 a,\ S_1 \to \epsilon\} \\ P_2 &= \{S_2 \to S_2 a,\ S_2 \to \epsilon,\ aS_2 \to b\} \end{aligned}$$

We get:

$$L_1 = L_2 = \{a^n \mid n \in \mathbb{N}_0\} = L_1 L_2$$

Grammars are closed under concatenation

Construction:

$$
\begin{aligned}
G_1 &= (N_1, T, P_1, S_1) \\
G_2 &= (N_2, T, P_2, S_2) \\
G &= (N_1 \cup N_2 \cup \{S\},\ T,\ P_1 \cup P_2 \cup \{S \to S_1 S_2\},\ S)
\end{aligned}
$$

Counterexample:

$$
\begin{aligned}
P_1 &= \{S_1 \to S_1 a,\ S_1 \to \epsilon\} \\
P_2 &= \{S_2 \to S_2 a,\ S_2 \to \epsilon,\ aS_2 \to b\}
\end{aligned}
$$

We get:

$$L_1 = L_2 = \{a^n \mid n \in \mathbb{N}_0\} = L_1 L_2$$

However:

$$S \Rightarrow S_1 S_2 \Rightarrow S_1 a S_2 \Rightarrow S_1 b \Rightarrow b$$

Carelessly assuming union of assumptions

```
[OrderedCancelAddCommMonoid C]
lemma Function.HasMaxCutProperty.
    forbids_commutativeFractionalPolymorphism

[OrderedAddCommMonoidWithInfima C]
lemma FractionalOperation.IsFractionalPolymorphismFor.
    expressivePowerVCSP

[OrderedCancelAddCommMonoidWithInfima C]
theorem ValuedCSP.CanExpressMaxCut.
    forbids_commutativeFractionalPolymorphism
```

The two latter classes extend `[CompleteSemilatticeInf C]`.

Carelessly assuming union of assumptions

```
[OrderedCancelAddCommMonoid C]
lemma Function.HasMaxCutProperty.
    forbids_commutativeFractionalPolymorphism

[OrderedAddCommMonoidWithInfima C]
lemma FractionalOperation.IsFractionalPolymorphismFor.
    expressivePowerVCSP

[OrderedCancelAddCommMonoidWithInfima C]
theorem ValuedCSP.CanExpressMaxCut.
    forbids_commutativeFractionalPolymorphism
```

The two latter classes extend `[CompleteSemilatticeInf C]`.
The assumption is too strong!

```
(∃ x y : C, x < y) → False
```

Not taking time to develop good notation

```
Multiset.sum ((ω.tt x).map (fun a => m * I.evalMinimize a))
```

Not taking time to develop good notation

```
Multiset.sum ((ω.tt x).map (fun a => m * I.evalMinimize a))

abbrev Multiset.summap {T M : Type*} [AddCommMonoid M]
    (s : Multiset T) (f : T → M) : M :=
  (s.map f).sum

Multiset.summap (ω.tt x) (fun a => m * I.evalMinimize a)
```

Not taking time to develop good notation

```
Multiset.sum ((ω.tt x).map (fun a => m * I.evalMinimize a))

abbrev Multiset.summap {T M : Type*} [AddCommMonoid M]
    (s : Multiset T) (f : T → M) : M :=
  (s.map f).sum

Multiset.summap (ω.tt x) (fun a => m * I.evalMinimize a)

attribute [pp_dot] Multiset.summap

(ω.tt x).summap (fun a => m * I.evalMinimize a)
```

Not taking time to develop good notation

We write $A\,x$ using a function `Matrix.mulVec`

```
A.mulVec x
```

We write $x^\top A$ using a function `Matrix.vecMul`

```
Matrix.vecMul x A
```

Not taking time to develop good notation

We write $A\,x$ using a function `Matrix.mulVec`

```
A.mulVec x
```

We write $x^\top A$ using a function `Matrix.vecMul`

```
Matrix.vecMul x A
```

Now we have infix operators $*_v$ and ${}_v*$

$\mathtt{A}\ *_v\ \mathtt{x}$

$\mathtt{x}\ {}_v{*}\ \mathtt{A}$

Not factoring out useful lemmas

$$
\begin{array}{l}
\texttt{lemma } \{x_1\ x_2\ z_1\ z_2 : \texttt{List T}\}\ \{a_1\ a_2 : \texttt{T}\} \\
\quad (\texttt{notin\_x} : a_2 \notin x_1)\ (\texttt{notin\_z} : a_2 \notin z_1) : \\
\quad (x_1 \texttt{ ++ } [a_1] \texttt{ ++ } z_1) = (x_2 \texttt{ ++ } [a_2] \texttt{ ++ } z_2) \iff \\
\quad\quad (x_1 = x_2) \land (a_1 = a_2) \land (z_1 = z_2)
\end{array}
$$

Reïnventing the wheel

Writing an existing definition from scratch

`List.count`

Reïnventing the wheel

Writing an existing definition from scratch

`List.count`

Developing lemmas about it

`List.count_eq_zero`

Reïnventing the wheel

Writing an existing definition from scratch

`List.count`

Developing lemmas about it

`List.count_eq_zero`

Not knowing existing theorems

```
Classical.choose_spec
Finset.prod_erase_eq_div
```

Using too many “collection types”

```
Fin n → T
Array
List
Multiset
Finset
Fintype
```

Jumping head-first into the full version

Better, when dealing with a difficult problem, is to:

Jumping head-first into the full version

Better, when dealing with a difficult problem, is to:

1. prove a simplified version (end to end)

Jumping head-first into the full version

Better, when dealing with a difficult problem, is to:

1. prove a simplified version (end to end)
2. learn the API for the "one new thing"

Jumping head-first into the full version

Better, when dealing with a difficult problem, is to:

1. prove a simplified version (end to end)
2. learn the API for the "one new thing"
3. discover if there is missing API

Jumping head-first into the full version

Better, when dealing with a difficult problem, is to:

1. prove a simplified version (end to end)
2. learn the API for the “one new thing”
3. discover if there is missing API
    3.1 involve the community
    3.2 write the missing API
    3.3 Mathlib PR

Jumping head-first into the full version

Better, when dealing with a difficult problem, is to:

1. prove a simplified version (end to end)
2. learn the API for the “one new thing”
3. discover if there is missing API
    3.1 involve the community
    3.2 write the missing API
    3.3 Mathlib PR
4. state the theorem

Jumping head-first into the full version

Better, when dealing with a difficult problem, is to:

1. prove a simplified version (end to end)
2. learn the API for the “one new thing”
3. discover if there is missing API
    3.1 involve the community
    3.2 write the missing API
    3.3 Mathlib PR
4. state the theorem
5. break it into steps

Jumping head-first into the full version

Better, when dealing with a difficult problem, is to:

1. prove a simplified version (end to end)
2. learn the API for the "one new thing"
3. discover if there is missing API
   - 3.1 involve the community
   - 3.2 write the missing API
   - 3.3 Mathlib PR
4. state the theorem
5. break it into steps
6. state lemmata

Jumping head-first into the full version

Better, when dealing with a difficult problem, is to:

1. prove a simplified version (end to end)
2. learn the API for the "one new thing"
3. discover if there is missing API
   3.1 involve the community
   3.2 write the missing API
   3.3 Mathlib PR
4. state the theorem
5. break it into steps
6. state lemmata
7. prove the theorem

Jumping head-first into the full version

Better, when dealing with a difficult problem, is to:

1. prove a simplified version (end to end)
2. learn the API for the "one new thing"
3. discover if there is missing API
    3.1 involve the community
    3.2 write the missing API
    3.3 Mathlib PR
4. state the theorem
5. break it into steps
6. state lemmata
7. prove the theorem
8. prove the lemmata

Thanks for your attention!