Dokumentationen Lean Projekte

Eduard Bopp

July 2023

1 Aufgabenstellung des ersten Lean Project

Die Aufgabenstellung des ersten Projekts lautete:

> "Jede invertierbare Matrix ist ein Produkt von Elementarmatrizen."

2 Herangehensweise

Meine Idee war es die Matrix zu nehmen und alle Umformungen zur Identität nachzugehen und die durch Elementarmatrizen darzustellen. Das heißt jeder Umformungsschritt kann durch das Multiplizieren an der Matrix mit einer Elementarmatrix dargestellt werden. Da die Anzahl der Elementarmatrizen nicht fest ist, habe ich mich dazu entschieden eine Liste zu nehmen.

3 Code

Ich habe mir meine Aufgabe formalisiert durch

```
theorem invertible_prod_of_elementary (n: \N) (M : matrix (fin n) (fin n) \alpha)
(h : invertible M) :
    matrices : list (matrix (fin n) (fin n) \alpha),  A, A  matrices → is_elementary n A  :=
  begin

  end
```

Dabei ist der Input eine Matrix der Dimension $\mathbb{R}^{n\times n}$ die invertierbar ist. Die Präposition h stellt das sicher. Als Ausgabe/Goal soll eine Liste mit Elementarmatrizen sein. Dabei ist die Funktion **is_elementary** selbst definiert. Die Funktion überprüft ob eine übergebene Matrix eine Elementarmatrix ist. Der Rückgabewert ist **True/False**.

```
def is_elementary (n:\N) (A : matrix (fin n) (fin n) \alpha) : bool :=
  if check_Typ_3 n A then
    true
  else
    if check_Typ_1 n A then
      true
    else
      if check_Typ_2 n A then
        true
      else
        false
```

Die Funktion **is_elementary** überprüft die 3 Bedingungen die man auf
https://de.wikipedia.org/wiki/Elementarmatrix nachlesen kann. Der input ist dabei wieder eine quadratische Matrix. Die 3 Funktionen mit dem Check überprüft jeweils auf den entsprechenden Typ. Wie die Funktionen genau aussieht überlasse ich im Code als Kommentar.

4 Probleme

Offensichtlich habe ich keinen Beweis zustande gebracht. Ich habe versucht Gaußoperationen auf die eingebene Matrix zu bewerkstelligen. Dabei kam das Problem auf dass ich die Elementarmatrizen erstellen musste was ich ausschließlich mit lambda Ausdrücken gemacht habe. Zum einem war das dass die einzige Möglichkeit, da Matrizen nicht veränderbar sind, d.h ich konnte nicht wirklich die Funktionen aus der Bibliothek benutzen und zu anderen kannte ich keine weiteren Möglichkeiten. Was auch wirklich furchtbar war die Sonderfälle abzugehen falls man durch 0 teilt. Das führte dazu dass ich indizes benutzen musste, was durch die unterschiedlichen Typen von fin und $\mathbb{N}$ scheiterte. Es kann vorkommen dass eine Funktion die mit listen arbeitet entweder fin oder $\mathbb{N}$ benutzt und deswegen nicht beides gleichzeitig benutzt werden kann. Was auch sehr nervig war, ist die filter funktion von listen

```
list.countp (eq 1)
```

Man kann mit gleich etwas filtern aber ich habe nicht herausgefunden wie man nach ungleich sortiert. Was gut war, ist dass die Matrizenfunktionen ziemlich einheitlich sind.

5 Was man besser machen könnte

Mir ist noch nicht ganz klar wie variablen definiert werden z.b

```
variables { : Type*} [field ] [decidable_eq ]
```

Ich stelle mir immer vor dass α mein $\mathbb{R}$ ist und weitere Eigenschaften wie field oder decidable_eq in der Variable vorhanden sind, sodass vorgefertigte Funktionen dass besser verarbeiten. Was genau Type ist, ist mir bis jetzt nicht richtig klar. Mehr Übung mit quantoren und lambda Ausdrücke wären sinnvoll.

6 Aufgabenstellung des zweiten Lean Project

Die Aufgabenstellung des zweiten Projekts lautete:

> "Zeige dass $\mathbb{R}$ kein endlich dimensionaler $\mathbb{Q}$-Vektorraum ist."

7 Herangehensweise

So eine richtige Vorgehensweise hatte ich nicht. Von dem lieben Herr Jon habe ich eine Vorlage gekriegt. Da sind zwei Abschnitte wo jeweils ein Teil ergänzt werden musste. Im ersten Teil gehts darum ein Lemma zu ergänzen. Die Eingabe dieses Lemmas ist ein Körper, ein Vektorraum und eine Menge τ. Den Sinn dieser Menge τ ist mir auch noch nicht vollständig klar. Jedenfalls ist die Aussage dieses Lemmas, dass die Kardinalität des Vektorraums V beschränkt ist durch die Kardinalität von $\mathbb{N}$. Der andere Teil ist ein example. Was genau ein example in lean ist wäre auch hilfreich. In unserem Fall ist das Beispiel die Aufgabenstellung. Da haben wir ein Gegenbeweis geführt und dabei ein Widerspruch erhalten.

8 Code

Für das Lemma haben wir folgenden Code geschrieben:

```
-- Dieser Befehl schreibt die Kardinalität des Vektorraums V
-- durch die Kardinalität der Abbildungen von tau zum Körper K
1.
rw cardinal.mk_congr (h2.equiv_fun.to_equiv),

2.
rw <-cardinal.power_def,

3.
rw h,

4.
-- forme #tau  in fintype um
simp only [cardinal.mk_fintype,cardinal.pow_cast_right],

5.
-- Forme die Potenz in was ohne potenz um
apply cardinal.power_nat_le,

6.
-- reflexiv
refl,
```

Im Beweis passiert folgendes:

1. Die Aussage $\#V \leq \#\mathbb{N}$ wird zu $\#(\tau \to K) \leq \#\mathbb{N}$. Die Umformung ist möglich da τ eine Basis des Vektorraums V über K darstellt.
2. Im zweiten Schritt kann man cardinal.power_def verwenden um die aussage zu $\#K^{\#l} \leq \#\mathbb{N}$ umzuformen.
3. Der Schritt ist nur dafür da die Represantation zu ändern. Für die Aussage des Beweis nicht bedeutend.
4. Der schwierige Part, was ich jetzt nur mit Hilfe geschafft habe ist. Hier wird $\#\tau$ "umgewandelt" in ein fintype.card. Dies mcht man um den nächsten Schritt zu ermöglichen.
5. Damit wird die Aussage so vereinfacht, dass alles gezeigt wird.

Jetzt kommen wir zum example

```
example : ¬finite_dimensional \Q \R :=
```

Wie dabei vorher angesprochen, machen wir ein Gegenbeweis:

```
 -- Die Basis von ‘\R‘ über ‘\Q‘
set B := basis.of_vector_space \Q \R,

-- beginnt Gegenbeweis
by_contradiction,
```

Wir haben uns vorher noch eine Basis B von $\mathbb{R}$ über $\mathbb{Q}$ definiert. Wir definieren uns 3 Aussagen mit have:

```
 have h_Q : #\Q = cardinal.aleph_0:= cardinal.mk_eq_aleph_0 \Q,

-- Anwenden des Lemmas auf die Basis ‘B‘ von ‘\R‘ über ‘\Q‘
--#R <= #Q
have cardinal_ineq : #\R  cardinal.aleph_0 :=
  cardinal_eq_of_finite_basis h_Q B,

-- R überabzählbar
have h3 := cardinal.not_countable_real,
```

Die erste Aussage ist : $\#\mathbb{Q} = \#\mathbb{N}$.
Die zweite Aussage ist : $\#\mathbb{R} \leq \#\mathbb{N}$.
Die dritte Aussage ist : ¬set.univ.countable. Die 3. Aussage ist vereinfacht ausgedrückt, dass $\mathbb{R}$ überabzählbar.

Die letzten 2 Befehle führen zum Widerspruch:

```
 rw <- cardinal.le_aleph_0_iff_set_countable at h3,

-- schreibe um zu ¬#
simp only [cardinal.mk_univ] at h3,
```

Diese Befehle sind hauptsächlich eine Umschreibung der Typen die man in lean hat.

Weitere Kommentare ist im Code vorhanden.

9 Probleme

Hauptsächlich hatte ich Probleme mit Befehlen wie simp only, simp oder simpsqueeze? Auch hier ist das im Zusammenhang mit den verschieden Typen die man geeignet umcasten muss.

10 Schluss

An sich ist die Programmiersprache sehr strukturiert. Das ist gut, man hat ein Input und einen Output. Problematisch ist die Typenumformung und das Verständnis wie man vorgefertigte Methoden nutzt. Das sollte man irgendwie genauer hervorbringen. Für sehr technische Beweise mit listen ist lean absolut nicht geeignet. Ich fand auch mit der Vorlage zu arbeiten viel besser, da ich nicht komplett verloren war und nicht alles alleine rauskriegen musste. Wenn man das Seminar mit 2 Personen halten könnte, könnten viel mehr Studierenden geholfen werden.

Was ich jetzt noch weiß ist, dass lean eine wandelnde Bibliothek ist und keine festen Versionen vorweist. Die Doku ist meistens unintuitiv und die Community die damit programmiert ist klein.