Index notation in Lean 4

Joseph Tooby-Smith

Reykjavik University

October 30, 2024

Abstract

Index notation is a tool commonly used in physics to manipulate tensors. In physics, we use index notation for three different types of tensors: Einstein tensors (e.g., ordinary vectors and matrices), Lorentz tensors, and Van der Waerden tensors. In this paper, we discuss how these are implemented in Lean 4 using a general mathematical theory based on category theory, and related to the notation of an operad.

1. INTRODUCTION

A previous work by the current author, presented the first steps of digitalizing (or formalizing) results from high energy physics into the interactive theorem prover Lean 4. This project is called HepLean. Lean is a programming language whose syntax looks similar to what we pen-and-paper mathematics, and is used to write and automatically check statements of definitions and theorems, and proofs of theorems. HepLean has four main motivations: js: sorry

When writing and proving results on paper, the physicists is accustumed to using notation, and no notation abounds in physics more than index notation. Thus as part of the larger HepLean project, index notation has been implemented by the author in to Lean 4. Any such implementation must satisfy two basic requirements:

1. Must be mathematically rigourous.
2. Must be easy to use.

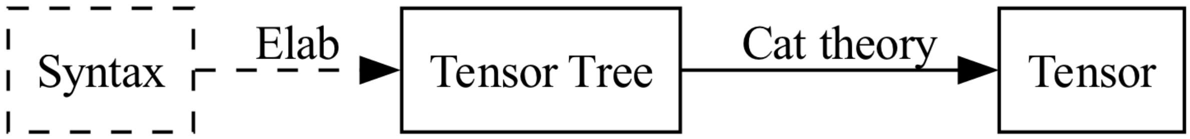
The first requirement is a consequence of Lean being a proof assistant, and not able to accept anything else but formally defined results. The second requirement is due to the fact that many other results will depend on the implementation of index notation. We beleive the implementation dicussed within this paper does satisfy both of these requirements.

- Notational conventions abound in physics, and non-more so then index notation.
- Notation, in general, is a way for the writer to compactly write down a term in a mathematical expression. The reader can implicitly unfold or elborate the compactly written term back into its full underlying meaning.
- Index notation is a compact way of writing expressions involving tensors.
- With tensors there is a notion of contraction, evaluation and permutation.

DRAFT

- All of these notions can defined independently of index notation.
- Index notation is a way of writing these operations in a compact way.
- In a previous work by the current author, the first foray of formalizing high energy physics in Lean 4 was undertaken, in a project called 'HepLean'.
- One aim of that project is to make Lean easier for the high-energy phsycisists to use.
- Motiviated by this aim, we have implemented index notation in Lean 4.
- In this paper, we will discuss how this is done.
- This is, of course, not the first paper dicussing the implmentation of index notation in a programming language. However, Lean, being a proof assistant, has a different set of requirements.
- In particular, Lean has to be provided a proof of everything.
- We also believe that the underlying mathematics used to implement index notation here is novel.

2. IMPLEMENTATION OF INDEX NOTATION INTO LEAN 4



- In this paper we will discuss each part of this diagram from the left to the right.

2.1. DEFINING TENSORS

2.1.1 Building blocks of tensors and color

- Tensors are built from a set of representations.
- For example, complex Lorentz tensors are built from six representations of the group $SL(2,\mathbb{C})$.
- To each of these building blocks we assign a label, which we call the color.
- For complex complex Lorentz tensor the colors are.
- In Lean this information is encoded as follows. We define a type C containing the colors.
- For complex Lorentz tensors this type is js: sorry
- Then we define a discrete functor from C to the category of complex representations of $SL(2,\mathbb{C})$.
- For complex Lorentz tensors this functor is defined as js: sorry.
- We will see a number of reasons shortly why we have used a functor here.

DRAFT

2.1.2 A general tensor

- A general tensor of a given species is roughly an element of a vector space built by taking the tensor product of the building block representations.
- To be more formal, let X be a type, and $f : X \to C$ a function from that type into the type of colors. Associated with f we can define a representation of $SL(2,\mathbb{C})$, by taking the tensor product of the representations $f(x)$ for $x \in X$.
- A general complex Lorentz tensor is then an element of one of these representations.
- To give an explicit example X may be something like $Fin2$, i.e. the type of numbers $(0,1)$. The function f may be $![Color.up, Color.down]$, and the corresponding representation is then (equivalent to) js: sorry.
- This can can be described categorically.
- The functions $f : X \to C$ live in a category whose objects are js: sorry and whose morphisms are js: sorry. This is equivalent the core of js: sorry.
- As such, we denote this category $\mathscr{S}^{\times}_{/C}$.
- This category has a symmetric monoidal structure, given by the disjoint union of X.
- There is a functor $C \to \text{Rep}_{SL(2,\mathbb{C})} \to \text{BraiFun}\mathscr{S}^{\times}_{/C}\text{Rep}_{SL(2,\mathbb{C})}$. from functors from $C \to \text{Rep}_{SL(2,\mathbb{C})}$ to symmetric monoidal js: sorry.
- We call this functor lift.
- A general tensor of a given species, which has discrete functor F_{dis} is then an element of a representation in the image of lift F_{dis}.
- We will be most intrested in the case when $X = Finn$ for some n.

2.1.3 Basic operations

- The formalisim we have discribed thus far gives us basic oprations on tensors for free.
- We get addition, scalar multiplication and the action of the group $SL(2,\mathbb{C})$ from the fact that things sit in representations.
- We also get the tensor product of two tensors, from the category of representations and the fact that our lift function is symmetric monoidal.
- We get permutation of indices from the fact we have a functor.
- We get negation of tensors because js: sorry
- We also get the interplay between all of these
- There somethings we do not get for free however: Contraction, the unit of the contraction, and the metrics.

DRAFT

- To define these we need introduce an involution $\tau : C \to C$. We will call the image of c under τ the dual of c.
- Colors can be self-dual, as there for Einstein tensors, for which there is only one color.
- From τ we get a functor $\tau_* : \mathscr{S}^{\times}_{/C} \to \mathscr{S}^{\times}_{/C}$.
- We let F_τ be the functor from C to $F(c) \otimes F(\tau c)$.
- To define contraction we define a natural transformation $F_\tau \to \nVdash$.
- That is for each c a morphism of representations from $F(c) \otimes F(\tau c)$ to the trivial representation.
- To give an example for complex Lorentz tensors js: sorry
- Using F_τ we can contract indices of tensors of dual colors.
- Lifiting F_τ gives us a functor which would allow us to contract all indices of a tensor at ones. This may be useful for come computations.
- In addition to contraction we want the notion of a metric.
- js: Unit
- js: Conditions

2.1.4 Tensor Species

- The data we have given so far consitutes what we have losely being calling a Tensor species.
- In Lean we make this more precise

2.2. TENSOR TREES

2.2.1 Structure

- A tensor expression consists of a series of tensors and operations performed on them.
- We can represent such an expression using a tensor tree, similar to the notion of a syntax tree.
- A tensor tree has different types of nodes either representing a tensor or a operation on or between tensors.
- Since we really only care about tensors with $X = Finn$, tensor trees in Lean are implemented only for these.
- Let us give the definition of a tensor tree and then dicuss in turn each of the nodes.
- js: Lean defn of tensor tree
- The basic node is a tensor node. The definition in Lean tells us how this is defined.
- Then for each operation addition, scalar mult, group action etc. we get a node.
- For example let us look at contraction
- js: sorry
- How we turn these tensor trees into tensors will be dicussed in the next section

DRAFT

2.2.2 To a tensor

- The notion of a tensor tree is defined without reference to the category theory we have been discussing.
- We can however turn a tensor tree into a tensor using said constructions.
- The definition of how we do this is defined recursively.

2.2.3 Using Tensor trees in proofs

- js: Discuss fact about 'tensor_eq'

2.3. ELBORATION

- We have dicussed the implementation of tensor species into Lean, and how to write tensor expressions using tensor trees.
- Really this is all one needs to effectively.
- However, we want our theorems to look like they do on paper.
- This is the role of the elaborator, which here takes a string written in lean code and turns it into a tensor tree.
- It is perhpase easist to give examples:
- The basic notation for a tensor nodes is js: sorry. Note that ... are free indices, it does not matter what we call them the expression is the same.
- The product of two tensors is written as js: sorry.
- The contraction of two tensors is written as js: sorry. Again note that the indices are free so we can call them anything without chaning how lean reads the expression.
- We also define a special notation of equality and addition. which takes account of permutation.
- This part of the Lean code is not formally verified, it is just telling lean how to read the notation. Once the tensor tree is created and we start using that, things are formally verifeid.

3. EXAMPLES

- We give two examples of theorems and proves related to index notation.
- Our first example is related to symmetric and anti-symmetric tensors.
- For this example we will go into explicit detail.
- Our second example is a more detailed one, where we prove that ...
- For this example we will only give a sketch of the prove, and discuss how things are done.

DRAFT

3.1. EXAMPLE 1: SYMMETRIC AND ANTI-SYMMETRIC TENSOR

Let $S_{\mu\nu}$ be complex Lorentz tensors where μ and ν are 4-vector indices. The corresponding

```
(S : complexLorentzTensor.F.obj (OverColor.mk ![Color.down, Color.down]))
```

To explain this notation let us work from the right to the left.

3.2. EXAMPLE 2: CONTRACTING PAULI MATRICES

4. FUTURE WORK

- Informal lemmas and definitions.
- Improvement of tactics.
- spinor-hleicity formalism.
- Tensor fields and derivatives.

REFERENCES

DRAFT