Blog Post

Nuiok and Paul

Way too late...

1 Introduction

Many well-known type theories, e.g. Martin-Löf dependent type theory or linear type theory, have as their starting points particular forms of reasoning that we wish to treat syntactically. In a sense, it is only a happy accident that these type theories then provide means to reason *about* locally cartesian closed categories or $\star$-autonomous categories.

Mike Shulman's

> "A Practical Type Theory for Symmetric Monoidal Categories"

revererses this; he starts with symmetric monoidal categories as the intended semantics and then (reverse)-engineers a syntax in which in which it is *practical* to reason about such categories.

But which properties of type theory make it practical? Shulman, sythesizing various observations settles on a few basic tenets to guide the invention of the syntax; we reduce thus further to the conditions that the type theory be: (1) intuitive, (2) economical, and (3) effective

intuitive First, a practical type theory should permit us to leverage our intuition with reasoning about 'sets with elements'. What this means in practice is that we require our type

theory, and the desired internal-language/term-model relationship with symmetric monoidal categories, to provide one-to-one correspondences:

$$\begin{aligned} \text{objects} &\longleftrightarrow \text{contexts} \\ \text{morphisms} &\longleftrightarrow \text{typing judgements} \\ \text{commuting polygons} &\longleftrightarrow \text{equality judgements} \end{aligned}$$

It is by doing so that we'll be free to think of types as sets, and to think of typed terms as elements in a way that applies to all symmetric monoidal categories.

economical Second, when working with a specific category, one usually works, whether we acknowledge it or not, with a presentation in terms of generators in relations, e.g. $\triangle$ as generated by face and degeneracy maps subject to the simplicial identities. As such, our type theory should not be directly specified in terms of a symmetric monoidal category, but rather in terms of a economical presentation of it.

effective Lastly, we ask that, given a presentation for a symmetric monoidal category, the type theory we get therefrom should be complete for that symmetric monoidal category. By this we mean that something should hold in a particular symmetric monoidal category if and only if it is derivable as a judgement in the associated type theory.

2 Symmetric Monoidal Categories and Presentations of PROPs

Shulman builds a practical type theory for each symmetric monoidal from a particularly economical description (of it); Shulman builds a practical type theory for each symmetic monoidal category from *a presentation by free* PROPs.

While it is 'well known' that every symmetric monoidal category is equivalent to a symmetric *strict* monoidal category, it is probably less well known that every symmetric strict monoidal category is equivalent to a PROP.

Definition 1. A PROP, $\mathcal{P}$, consists of a set P of generating objects and a symmetric strict monoidal category whose underlying monoid of objects is the free monoid on the set P.

Example. Given a set X, the **free permutative category on** X, Σ_X, whose monoid of objects is the monoid of lists drawn from X and whose morphisms are given by the expression

$$\Sigma_X\left(\overrightarrow{x}, \overrightarrow{y}\right) = \{\, \sigma \in S_n \mid \overrightarrow{x_\sigma} = \overrightarrow{y} \,\}$$

(where by $\overrightarrow{x_\sigma}$ we mean the reorganization of the list $\overrightarrow{x}$ according to the permutation σ) is a PROP. For a more complicated example, let X_0 be a set and let $X_1 = \{\, (\overrightarrow{x})_i \mid i \in I \,\}$ be a set of pairs of lists valued in X_0. The

While we haven't gotten there yet, it's worth mentioning in advance this has immediate benefits for our type theory - we will never need to judge any two types to be equal - they are merely the same list or they are not. This means that we can replace any symmetric monoidal category with a PROP, that is, a monoidally equivalent symmetric monoidal category in which the monoidal structure is at once as free and as strict as possible.

However, to define a PROP whose morphisms obey specific equations it is better to talk about a presentation of a prop. Intuitively, a presentation for a PROP need only be comprised of three sets:

- a set of generating objects;
- a set of generating morphisms, formal arrows between finite lists of generating objects; and
- a set of generating identities of morphisms, between formal composites and tensors generating morphisms.

Formally, this is defined using the notion of signatures. A signature $\mathcal{G}$ for a PROP consists of a set of generating objects, as well as a set of formal arrows between lists of these objects. The presentation of a PROP $\mathcal{P}$ is then defined as the following coequaliser:

$$\mathfrak{F}\mathcal{R} \rightrightarrows \mathfrak{F}\mathcal{G} \to \mathcal{P}$$

where $\mathcal{G}$ and $\mathcal{R}$ are signatures whose sets of objects are the same as that of $\mathcal{P}$, $\mathfrak{F}$ is the free PROP on a signature functor, and the two morphisms of PROPs acting as the identity on objects. We note that every PROP $\mathcal{P}$ admits such a presentation. This will be central to the method used to define a practical type theory for symmetric monoidal categories. Those familiar with the theory of computads will note that the data enumerated above exactly describes a 2-computad for a PROP.

3 From Presentation to Type Theory and Back

We now give a brief overview of the general approach needed to obtain a practical type theory for

1. *Start with the SMC for which we would like to build a PTT and consider its presentation as a (monoidally equivalent) representable (i.e has tensor and has unit) PROP $\mathcal{P}$:*

$$\mathfrak{F}\mathcal{R} \rightrightarrows \mathfrak{F}\mathcal{G} \to \mathcal{P}$$

   - Here $\mathcal{G}$ and $\mathcal{R}$ are, respectively, the signatures of generators and relations that appear in the presentation of $\mathcal{P}$ as a coequalizer of free PROPs over signatures.

2. *Given a presentation $\mathfrak{F}\mathcal{R} \rightrightarrows \mathfrak{F}\mathcal{G} \to \mathcal{P}$ of a symmetric monoidal category as a PROP $\mathcal{P}$ we define a practical type theory. This is done by specifying rules for term judgements, for typing judgements, and for equality judgements. The very rough idea is that contexts*

in the type theory are obtained from objects of the signature $\mathcal{G}$, while typing judgements are obtained from the morphisms. We present a few more details below:

- The contexts (usually denoted Γ, Δ, etc.) of the type theory are specified as being lists $x_1 : A_1, \ldots, x_n : A_n$ of variables (where the A_i's are objects in the signature $\mathcal{G}$)
- The next step is to rules for the judgements. There are three kinds of judgements defined in this type theory: term judgements, typing judgements, and equality (of typed terms) judgements.
  - Term judgements allows us to judge that some terms (for example x, or the application of f to a x) are terms (Figure 4 in [**?**])
  - The second kind of judgements are typing judgement and are obtained from the morphisms in the SMC, e.g. $x : A \vdash f(x) : B$ corresponding to a morphism $f : A \longrightarrow B$. This obviously generalizes to more complex examples.
  - Lastly equality judgements allow us to express when two functions are equal in the type theory. This is best understood by looking at the following example: take the simple example of two stating that the composite of two functions g and h is equal to the function f. This is expressed by the following equality judgement: $x : A \vdash \left(f\left(g_{(1)}(x)\right), g_{(2)}(x)\right) = \left(h_{(1)}(x), h_{(2)}(x)\right) : (B_1, B_2)$
- Much care is taken while defining these rules to ensure, among other things, that the composition and the exchange rules are admissible, therefore ensuring that any judgement has a unique derivation. We will come back to this point later.

That's it! Following these steps gives us a type theory for the prop presented by $(\mathcal{G}, \mathcal{R})$, which, as proven in the paper, allows us to reason about structures in any props, and hence also in any symmetric monoidal category.

Before we illustrate this with an example, there are two notions that are worth exploring in more details. First is Sweedler's notation and how it connects to the definition of the

judgements and second is the notion of accessibility of structural rules.

3.1 Sweedler's notation and Shulman's judgements

An important ingredient in the syntax of the type theories described in this paper is Sweedler's notation. To explain this, we first note that the practical type theory for a SMC is not cartesian. Indeed, while products enjoy a universal property characterised in terms of projection maps, this is not the case for arbitrary symmetric monoidal products. To illustrate this, think of vector spaces, where we know that any element $z \in X \times Y$ is uniquely specified by the pair of elements $x \in X$ and $y \in Y$. However, elements w of the tensor product $X \otimes Y$ do not need to be a simple tensors of the form $x \times y$ and can instead consist of linear combinations of simple tensors.

Consider the general term $\sum_{i=1}^{k} x^i \otimes y^i \in X \otimes Y$, which in Sweedler's notation is written as (x, y) with the summation, indices, and tensor symbols all dropped from the notation. We are effectively pretending that every element of $X \otimes Y$ is a simple tensor of the form $x \otimes y$, which we can do, provided that any expression involving a simple tensor $x \otimes y$ is linear in both x and y. In essence, this premise is what Shulman develops into the core of his practical type theory. To see how this plays out, suppose that we have a function f that has domain A and codomain consisting of the tensor product $(B_1, ..., B_n)$. Then, in Sweedler's notation we have the following judgement:

$$a : A \vdash \left(f_{(1)}\left(a\right), \ldots, f_{(n)}\left(a\right)\right) : (B_1, \ldots, B_n)$$

In the case of $n = 1$, Sweedler's notation then allows us to consider bare terms $f_{(k)}(M)$ corresponding to the k^{th} component of the morphism f (under the premise that a context Γ judges some M to be a term). Now, as we have already mentioned, the typing judgements are derived from rules specified by the arrows of the signature $\mathcal{G}$. For example, if in the

signature $\mathcal{G}$ we find a formal morphism

$$f : (A_1, \ldots, A_m) \longrightarrow (B_1, \ldots, B_n)$$

we will then have a rule for the typing judgement which corresponds to applying that morphism to a list $(M_1, \ldots, M_n) : (A_1, \ldots, A_n)$ of typed terms. This rule for the typing judgements will formalize, for general symmetric monoidal categories, the way in which we could pretend above that every element of a tensor product of vector spaces was a simple tensor provided we only ever considered expressions which were linear in each variable.

****This is still not a very clear explanation****

3.2 On the meaning of generation and the admissibility of structural rules

Let us now have a deeper look at exactly how a signature G generates the rules of the typing judgement. The 'obvious' way would be to define a series of rule which would respectively allow us to:

1. match the application of a morphism f to a term $(M_1, \ldots, M_n)$;
2. concatenate terms (i.e. effectively tensoring lists) to obtain terms of type

$$(M_1, \ldots, M_n, N_1, \ldots, N_m) : (A_1, \ldots, A_n, B_1, \ldots, B_m)$$

   from two distinct terms in A_i and B_j;
3. describe permutations inside the lists of terms, i.e. something that allows us to incorporate things like $\left(M_{\sigma(1)}, \ldots, M_{\sigma(n)}\right) : \left(A_{\sigma(1)}, \ldots, A_{\sigma(n)}\right)$ in the type theory.

The problem with this 'obvious' way however is that derivations for typing judgements would be non-unique. For example, we see that with this logic, the two different associations

of three composable morphisms $f : A \longrightarrow B$, $g : B \longrightarrow C$, and $h : C \longrightarrow D$ would yield distinct derivations of the same typing judgement, notably:

$$\dfrac{\dfrac{x : A \vdash f(x) : B \quad y : B \vdash g(y) : C}{x : A \vdash g(f(x)) : C} \qquad z : C \vdash h(z) : D}{x : A \vdash h(g(f(x)))}$$

and

$$\dfrac{x : A \vdash f(x) : A \qquad \dfrac{y : B \vdash g(y) : C \quad z : C \vdash h(z) : D}{y : A \vdash h(g(y)) : D}}{x : A \vdash h(g(f(x)))}$$

As induction over derivations is far more cumbersome than induction over derivable judgements Shulman opts for a more sophisticated tack by defining the rules for the typing judgements such that:

- derivations of typing judgements are unique; and
- the structural rules, i.e. the rules corresponding to composition, tensorings, and exchange are admissible which allow us to reason with them even if they are not the actual rules of the type theory.

Any user of the type theory may invoke these more naive 'structural' rules in derivations and then freely ignore the multiplicity of derivations such reasoning may bring about.

4 Example: The Free Dual Pair

Recall that for a vector space V and its dual vector space V^*, we've a bijection

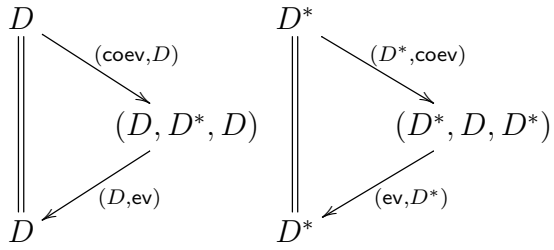
$$\mathsf{Hom}(A \otimes V, B) \xrightarrow{\sim} \mathsf{Hom}(A, V^* \otimes B)$$

natural in vector spaces A and B. This example is abstracted into the usual defintion of a dual pair in a symmetric strict monoidal category as follows.

Definition 1. A dual pair $(D, D^*, \mathsf{coev}, \mathsf{ev})$ in a symmetric monoidal category $(\mathcal{C}, (\_, \_), ())$ is comprised of:

- a pair of objects D and D^* of $\mathcal{C}$;
- a morphism $\mathsf{coev} : \mathbf{1} \longrightarrow D \otimes D^*$; and
- a morphism $\mathsf{ev} : D^* \otimes D \longrightarrow \mathbf{1}$

satisfying the triangle identities:



What's more, these data clearly suggest a presentation of the PROP generated by a dual pair. We set

$$\mathcal{G} = (\{D, D^*\}, \{\mathsf{coev} : () \longrightarrow (D, D^*), \mathsf{ev} : (D^\star, D) \longrightarrow ()\}),$$

we set

$$\mathcal{R} = (\{D, D^*\}, \{\mathsf{triangle} : D \longrightarrow D, \mathsf{triangle}^* : D^* \longrightarrow D^*\}),$$

and for the maps defining the presentation we pick the ones generated by the assignments

$$\begin{aligned} \mathsf{triangle} &\longmapsto \mathrm{id}_D \\ \mathsf{triangle}^* &\longmapsto \mathrm{id}_{D^*} \end{aligned}$$

and

$$\begin{aligned} \mathsf{triangle} &\longmapsto (D, \mathsf{ev}) \circ (\mathsf{coev}, D) \\ \mathsf{triangle}^* &\longmapsto (\mathsf{ev}, D^*) \circ (D^* \mathsf{coev}) \end{aligned}$$

The rules for the term judgment are few:

- $\dfrac{(x : A) \in \Gamma}{\Gamma \vdash x \ \mathsf{term}}$;
- $\dfrac{\mathfrak{a} \in \mathfrak{A} \quad 1 \leq k \leq 2}{\mathsf{coev}^{\mathfrak{a}}_{(k)} \ \mathsf{term}}$; and
- $\dfrac{\Gamma \vdash M \ \mathsf{term} \quad \Gamma \vdash N \ \mathsf{term}}{\Gamma \vdash \mathsf{ev}\,(M, N) \ \ \mathsf{term}}$.

FILL IN ADMISSIBLE RULES FOR TYPING JUDGEMENTS

note that we may derive the typing judgments corresponding the compositions

$$(D) \xrightarrow{(\mathsf{coev}, D)} (D, D^*, D) \xrightarrow{(D, \mathsf{ev})} (D)$$

and

$$(D^*) \xrightarrow{(D^*, \mathsf{coev})} (D, D^*, D) \xrightarrow{(D, \mathsf{ev})} (D)$$

Using the admissible rulessing the actual rules for the type theory (modulo the consideration of 'activeness') the canonical derivation for that first morphism follows.

$$\dfrac{\dfrac{\mathsf{coev} \in \mathcal{G}\,(;D, D^*) \quad \mathfrak{a} \in \mathfrak{A} \quad (132) : (D, D, D^*) \xrightarrow{\sim} (D, D^*, D)}{x : D \vdash \left(\mathsf{coev}^{\mathfrak{a}}_{(1)}, \mathsf{coev}^{\mathfrak{a}}_{(2)}, x\right) : D}}{x : D \vdash \left(\mathsf{coev}^{\mathfrak{a}}_{(1)} | \mathsf{ev}\left(\mathsf{coev}^{\mathfrak{a}}_{(2)}, x\right)\right)} \ \langle \in \rangle\, \mathcal{G}\,(D^*, D)$$

Lastly, we have the axioms

$$\dfrac{}{M : D \vdash \left(\mathsf{coev}_{(1)} | \mathsf{ev}\left(\mathsf{coev}_{(2)}, M\right)\right) = M : D}$$

and

$$\dfrac{}{N : D^* \vdash \left(\mathsf{coev}_{(2)} | \mathsf{ev}\left(N, \mathsf{coev}_{(1)}\right)\right) = N : D^*}$$

rules enough to generate a congruence.

4.1 'Elements' of dual objects

We have now developed a type theory for the free dual pair which endows the dual objects D and D^* with a universal notion of element. Since the notion of dual pair abstracted the instance of a pair of dual vector spaces, which in particular have actual elements, it behooves us to ask:

> "how much like a vector is an term of type D"

The answer is both practical and electrifying (though perhaps the authors of this blog post are too easily electrified).

It's easy enough to believe that the evaluation map

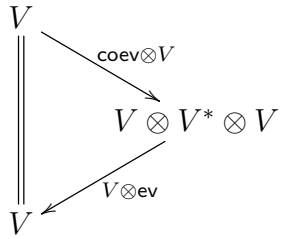
$$\mathsf{ev} : (D, D^\star) \longrightarrow ()$$

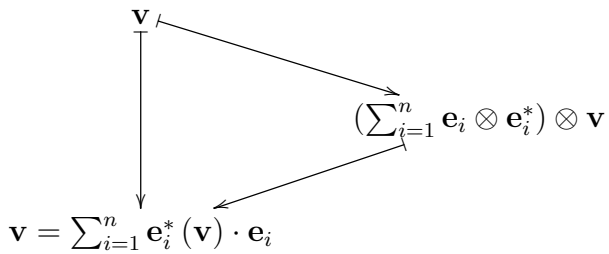
endows the terms of type D, or D^* for that matter, with structure of scalar valued functions on the other. The triangle idenities impose the unique determination of terms of type D or D^* in terms of their values as given by ev.

Consider that, for a finite dimensional vector space V over a field k, a basis $\{\mathbf{e}_i\}_{i=1}^n$ for V and a dual basis $\{\mathbf{e}_i^*\}_{i=1}^n$ for V^* give us an elegant way to write coev and the the first triangle identity. We write

$$\begin{array}{l} k \xrightarrow{\mathsf{coev}} V \otimes V^* \\ x \longmapsto \sum_{i=1}^n \mathbf{e}_i \otimes \mathbf{e}_i^* \end{array}$$

and see that





The observation, for dual vector spaces defined by way of the equality

$$V^* = \mathsf{Hom}_{\mathsf{Vect}_k} (V, k)$$

that the triangle identities hold is just the observation that a vector is precisely determined by its values: every vector $\mathbf{v}$ is equal to the un-named vector

$$\sum_{i=1}^{n} \mathbf{e}_i^* (\mathbf{v}) \cdot \mathbf{e}_i$$

which is defined by taking the values $\mathbf{e}_i^* (\mathbf{v})$ at the dual vectors $\mathbf{e}_i^*$. As part of the definition of a dual pair in an arbitrary symmetric strict monoidal category then, the triangle identities imposes this as a relationship between ev and coev. But within type theory, this sort of relationship between an un-named function and its values is familiar, indeed it is something very much like β-reduction.

To see this more clearly, let's make a pair of notational changes to bring the parallel to the fore. In place of writing

$$(x, y) : (D^*, D) \vdash \mathsf{ev} (x, y) : ()$$

we'll denote ev infix by $\_ \triangleleft \_$ and write

$$(x, y) : (D^*, D) \vdash x \triangleleft y : () .$$

Similarly, in place of writing

$$\vdash \left(\mathsf{coev}_{(1)}, \mathsf{coev}_{(2)}\right) : (D, D^*)$$

we'll denote coev by the pair $\left(u, \lambda^D u\right)$ and write

$$\vdash \left(u, \lambda^D u\right) : (D, D^*)\,.$$

With this choice of notation then, the axiom which corresponds to the first triangle identity is

$$x : D \vdash \left(u | \lambda^D u \triangleleft x\right) = x : D.$$

Then, as Shulman points out, since $=$ is a congruence with respect to substitution, if we've, for some term M, the term $\lambda^D u \triangleleft M$ appearing in the scalars of a list of terms, then we may replace all instances of u in the rest of the term with M. While a mouthful, this is a sort of 'β-reduction for duality' as Shulman identifies it. Conceptually interesting in its own right, this obeservation also yields a one line proof for a familiar theorem.

Lemma 2. (cite original result)(cyclicity of trace)Let $\left(\mathsf{C}, \left(\_, \_\right), ()\right)$ be a symmetric strict monoidal category, let

$$\left(A, A^*, \left(u, \lambda^A u\right), \_ \triangleleft \_\right)$$

and

$$\left(B, B^*, \left(v, \lambda^B v\right), \_ \triangleleft \_\right)$$

be dual pairs in C, and let $f : A \longrightarrow B$ and $g : B \longrightarrow A$ be morphisms in C. Let $\mathsf{tr}\left(f \circ g\right)$ be the composition

$$() \xrightarrow{\left(v, \lambda^B v\right)} (B, B^*) \xrightarrow{f \circ g} (B, B^*) \xrightarrow{(12)} (B^*, B) \xrightarrow{\_ \triangleleft \_} ()$$

and likewise let $\mathsf{tr}\,(g \circ f)$ be the composition.

$$() \xrightarrow{\left(u, \lambda^A u\right)} (A, A^*) \xrightarrow{g \circ f} (A, A^*) \xrightarrow{(12)} (A^*, A) \xrightarrow{-\triangleleft -} () \ .$$

Then, $\mathsf{tr}\,(f \circ g) = \mathsf{tr}\,(g \circ f)$.

Proof.

$$\begin{aligned} \mathsf{tr}\,(f \circ g) &\overset{\mathsf{def}}{=} \left(\left|\lambda_u^B \triangleleft f\,(g\,(u))\right.\right) \\ &= \left(\left|\lambda_u^B \triangleleft f\,(v)\,, \lambda_v^A \triangleleft g\,(u)\right.\right) \\ &= \left(\left|\lambda_v^A \triangleleft g\,(f\,(v))\right.\right) \\ &\overset{\mathsf{def}}{=} \mathsf{tr}\,(g \circ f) \end{aligned}$$

Where the judged equalities are application of 'β-reduction for a duality'. □