Strictly Associative Sigmas (Work In Progress)

Emmanuel (Emma) Suárez Acevedo

MURI Meeting (2024)

Martin-Löf Type Theory (MLTT)

There are the following judgements:

Contexts: $\vdash \Gamma cx$

Types: Γ ⊢ *A* type

Substitutions: $\Delta \vdash \gamma : \Gamma$

► Terms: $\Gamma \vdash a : A$

Of particular interest to this talk are the **Unit** type and Σ -types:

$$\frac{ \vdash \Gamma \operatorname{cx} }{\Gamma \vdash \operatorname{Unit} \operatorname{type} } \qquad \frac{\vdash \Gamma \operatorname{cx} }{\Gamma \vdash \operatorname{tt} : \operatorname{Unit} } \qquad \frac{\Gamma \vdash A \operatorname{type} \qquad \Gamma.A \vdash B \operatorname{type} }{\Gamma \vdash \Sigma(A,B) \operatorname{type} }$$

$$\frac{\Gamma \vdash a : A \qquad \Gamma.A \vdash B \operatorname{type} \qquad \Gamma \vdash b : B[\operatorname{id}.a]}{\Gamma \vdash \operatorname{pair}(a,b) : \Sigma(A,B)}$$

Idea

Suggested by Favonia, Carlo Angiuli, and Jon Sterling: **What if we** make Σ -types unital and associative?

$$\frac{\Gamma \vdash A \text{ type}}{\Gamma \vdash \Sigma(\textbf{Unit}, A) = A \text{ type}} \frac{\Gamma \vdash A \text{ type}}{\Gamma \vdash \Sigma(A, \textbf{Unit}) = A \text{ type}}$$

$$\frac{\Gamma \vdash A \text{ type}}{\Gamma \vdash \Sigma(A, \Sigma(B, C)) = \Sigma(\Sigma(A, B), C) \text{ type}}$$

Consequences?

- ► Consistency?
- ► Normalization?
- Elaboration? (i.e. to develop a proof assistant)

Motivation

- 1. Usability of proof assistants
- 2. Curiosity?

Motivation

- 1. Usability of proof assistants
- 2. Curiosity?

```
\begin{split} \text{Poset} : & \text{Set} \rightarrow \text{Set}_1 \\ \text{Poset} \ & \text{X} = \mathcal{\Sigma} \big[ \ \_ \leq \_ \in (\text{X} \rightarrow \text{X} \rightarrow \text{Set}) \ \big] \\ & - \text{reflexivity} \\ & (\forall \ x \rightarrow x \leq x) \times \\ & - \text{antisymmetry} \\ & (\forall \ x \ y \rightarrow x \leq y \rightarrow y \leq x \rightarrow x \equiv y) \times \\ & - \text{transitivity} \\ & (\forall \ x \ y \ z \rightarrow x \leq y \rightarrow y \leq z \rightarrow x \leq z) \end{split}
```

```
Semigroup : Set \rightarrow Set
Semigroup S = \Sigma [+ \in (S \rightarrow S \rightarrow S)]
                     - _+_ is associative
                     (\forall s_1 s_2 s_3 \rightarrow s_1 + (s_2 + s_3) \equiv (s_1 + s_2) + s_3)
Monoid : Set \rightarrow Set
Monoid M = \Sigma[(+, ) \in Semigroup M]
                  \Sigma[e \in M]
                  - e is a left and right identity
                  (\forall m \rightarrow e + m \equiv m) \times
                  (\forall m \rightarrow m + e \equiv m)
```

```
- Poset X : \_\le\_ \times \le/\text{refl} \times \le/\text{antisym} \times \le/\text{trans}

- Semigroup S : \_+\_ \times +/\text{assoc}

- Monoid M : semigrp \times e \times +/identity<sup>1</sup> \times +/identity<sup>r</sup>

PoMonoid M = \Sigma[((\_\_,\_),\_,\_,\_) \in \text{Monoid } M]

\Sigma[(\_\le\_,\_) \in \text{Poset } M]

- compatibility of \_\cdot\_ with \_\le\_

(\forall \ x \ y \ z \to x \le y \to (x \cdot z) \le (y \cdot z)) \times

(\forall \ x \ y \ z \to x \le y \to (z \cdot x) \le (z \cdot y))
```

```
- Poset X : _≤_ × ≤/refl × ≤/antisym × ≤/trans

- Semigroup S : _+_ × +/assoc

- Monoid M : semigrp × e × +/identity<sup>1</sup> × +/identity<sup>r</sup>

- PoMonoid M : monoid × poset × +/compat<sup>r</sup> × +/compat<sup>1</sup>

prop : \forall {M}

   ((((_+_ , _), e , _ , _), (_≤_ , _ , _ , _), _ , _) : PoMonoid M)

        → \forall m → (e + m) ≤ m

prop ...
```

Mental overhead to use the components of the pomonoid

```
- Poset X : \_\le\_ \times \le/\text{refl} \times \le/\text{antisym} \times \le/\text{trans}
- Semigroup S : \_+\_ \times +/\text{assoc}
- Monoid M : semigrp \times e \times +/identity^1 \times +/identity^r
- PoMonoid M : monoid \times poset \times +/compat^r \times +/compat^1

prop : \forall {M}
((((\_+\_,\_), e,\_,\_), (\_\le\_,\_,\_,\_),\_,\_) : \text{PoMonoid M})
\rightarrow \forall m \rightarrow (e + m) \le m
prop ...
```

What if we had strictly associative sigmas?

```
- Poset X : \_\le\_ \times \le/\text{refl} \times \le/\text{antisym} \times \le/\text{trans}
- Semigroup S : \_+\_ \times +/\text{assoc}
- Monoid M : semigrp \times e \times +/identity^1 \times +/identity^r
- PoMonoid M : monoid \times poset \times +/compat^r \times +/compat^1

prop : \forall {M}
((\_+\_,\_,e,\_,\_,\_,\_,\_,\_,\_,\_,\_): \text{PoMonoid M})
\to \forall m \to (e + m) \le m
prop ...
```

Motivation

- 1. Usability of proof assistants ✓
 - e.g. reduces mental overhead when dealing with nested sums
- 2. Curiosity?

Motivation

- 1. Usability of proof assistants ✓
 - e.g. reduces mental overhead when dealing with nested sums
- 2. Curiosity?

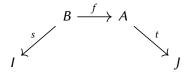
Related work

Type theory is a polynomial pseudomonad and polynomial pseudoalgebra (Awodey and Newstead 2018)

Review: polynomials

Let \mathcal{E} be a locally cartesian closed category (lccc).

A **polynomial** $p: I \rightarrow J = (s, f, t)$ in \mathcal{E} is a diagram of the form:



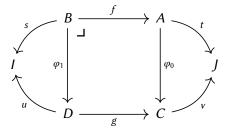
Every morphism $f: B \to A$ in \mathcal{E} is a polynomial $\mathbf{1} \to \mathbf{1}$ (taking s and t to be the unique morphisms to the terminal object $\mathbf{1}$ of \mathcal{E})

For any object *I*, the **identity polynomial** $i_I: I \rightarrow I$ is (id_I, id_I)

Review: polynomials

A morphism of polynomials $\varphi: p \Rightarrow q$ is an object D_{φ} and a triplet of morphisms $(\varphi_0, \varphi_1, \varphi_2)$

 φ is **cartesian** if φ_2 is invertible, in which case it is uniquely represented by the following diagram:

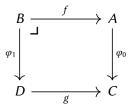


with p = (s, f, t) and g = (u, g, v)

Review: polynomials

Recall any morphism in \mathcal{E} can be considered as a polynomial $\mathbf{1} \leftrightarrow \mathbf{1}$.

For two morphisms $f: B \to A$ and $g: D \to C$, a cartesian morphism $\varphi: f \Rightarrow g$ can be further simplified to the following pullback square:



Review: natural models

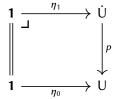
A **natural model** of type theory is a category \mathbb{C} along with:

- ▶ a terminal object ◊
- ▶ a *representable* map of presheaves $p : \dot{U} \rightarrow U$ on \mathbb{C}

- ▶ $p: \dot{U} \rightarrow U$ is a morphism in the lccc **Set**^{\mathbb{C}^{op}}
- ▶ p can be considered a polynomial $1 \leftrightarrow 1$ in **Set**^{\mathbb{C}^{op}}
- The conditions for the natural model to support unit and dependent sum types can be phrased in terms of morphisms of polynomials

Review: natural model and Unit type

The model supports unit types iff there exists a cartesian morphism $\eta: i_1 \Rightarrow p$. Diagrammatically:



Review: natural model and Σ -types

The model supports dependent sum types iff there exists a cartesian morphism $\mu : p \cdot p \Rightarrow p$. Diagrammatically:

$$\sum_{A:U} \sum_{B:U^A} \sum_{a:A} B(a) \xrightarrow{\mu_1} \dot{U}$$

$$\downarrow^{p \cdot p} \qquad \qquad \downarrow^{p}$$

$$\sum_{A:U} U^A \xrightarrow{\mu_0} \dot{U}$$

Review: polynomial monad

A **polynomial monad** is a quadruple (I, p, η, μ) consisting of:

- \triangleright an object I of \mathcal{E}
- ▶ a polynomial $p: I \rightarrow I$ in \mathcal{E}
- ► cartesian morphisms $\eta: i_l \Rightarrow p$ and $\mu: p \cdot p \Rightarrow p$ satisfying the usual monad axioms (e.g. $\mu \circ (p \cdot \eta) = \mathrm{id}_p$)

Is $(1, p, \eta, \mu)$ a polynomial monad? In particular, does it satisfy the usual monad laws?

For example:

$$\mu \circ (p \cdot \eta) = \mathrm{id}_p$$
$$\mu \circ (\eta \cdot p) = \mathrm{id}_p$$

Is $(1, p, \eta, \mu)$ a polynomial monad? In particular, does it satisfy the usual monad laws?

For example:

$$\mu \circ (p \cdot \eta) = \mathrm{id}_p$$
$$\mu \circ (\eta \cdot p) = \mathrm{id}_p$$

No — this would correspond to $\Sigma(\mathbf{Unit}, A)$ being equal to A and $\Sigma(A, \mathbf{Unit})$ being equal to A, which is not the case in MLTT.

Is $(1, p, \eta, \mu)$ a polynomial monad? In particular, does it satisfy the usual monad laws?

For example:

$$\mu \circ (p \cdot \mu) = \mu \circ (\mu \cdot p)$$

Is $(1, p, \eta, \mu)$ a polynomial monad? In particular, does it satisfy the usual monad laws?

For example:

$$\mu \circ (p \cdot \mu) = \mu \circ (\mu \cdot p)$$

No — this would correspond to $\Sigma(A, \Sigma(B, C))$ being equal to $\Sigma(\Sigma(A, B), C)$, which is not the case in MLTT.

Dependent type theories admitting a unit type and dependent sum types give rise to a polynomial *pseudo*monad. (Awodey and Newstead 2018)

• On the other hand, if (1, p, η, μ) were a polynomial monad — this model would seem to have a correspondence with MLTT with unital and associative Σ-types.

Motivation

- 1. Usability of proof assistants ✓
 - e.g. reduces mental overhead when dealing with nested sums
- 2. Curiosity? ✓
 - e.g. learning more about type theory as a polynomial monad

Σ-types are unital

$$\frac{\Gamma \vdash A \text{ type}}{\Gamma \vdash \Sigma(\text{Unit}, A) = A \text{ type}}$$

$$\frac{\Gamma \vdash A \text{ type}}{\Gamma \vdash \Sigma(A, \mathbf{Unit}) = A \text{ type}}$$

Σ-types are unital

$$\frac{\Gamma \vdash A \text{ type}}{\Gamma \vdash \Sigma(\text{Unit}, \mathbf{A}) = A \text{ type}}$$

$$\frac{\Gamma \vdash A \text{ type}}{\Gamma \vdash \Sigma(A, \mathbf{Unit}) = A \text{ type}}$$

Σ-types are unital

$$\frac{\vdash \Gamma \operatorname{cx}}{\vdash \Gamma.\mathbf{Unit} = \Gamma \operatorname{cx}}$$

$$\frac{\Gamma \vdash A \text{ type}}{\Gamma \vdash \Sigma(\mathbf{Unit}, A) = A \text{ type}}$$

$$\frac{\Gamma \vdash A \text{ type}}{\Gamma \vdash \Sigma(A, \mathbf{Unit}) = A \text{ type}}$$

Σ -types are associative

$$\frac{\Gamma \vdash A \text{ type} \qquad \Gamma.A \vdash B \text{ type} \qquad \Gamma.A.B \vdash C \text{ type}}{\Gamma \vdash \Sigma(A, \Sigma(B, C)) = \Sigma(\Sigma(A, B), C) \text{ type}}$$

Σ -types are associative

$$\frac{\Gamma \vdash A \text{ type} \qquad \Gamma.A \vdash B \text{ type} \qquad \Gamma.A.B \vdash C \text{ type}}{\Gamma \vdash \Sigma(A, \Sigma(B, C)) = \Sigma(\Sigma(A, B), \textbf{C}) \text{ type}}$$

Σ -types are associative

$$\frac{\Gamma \vdash A \text{ type} \qquad \Gamma.A \vdash B \text{ type}}{\vdash \Gamma.\Sigma(A, B) = \Gamma.A.B \text{ cx}}$$

$$\frac{\Gamma \vdash A \text{ type} \qquad \Gamma.A \vdash B \text{ type} \qquad \Gamma.A.B \vdash C \text{ type}}{\Gamma \vdash \Sigma(A, \Sigma(B, C)) = \Sigma(\Sigma(A, B), C) \text{ type}}$$

Context equations?

What does this mean for elaboration?

• e.g. synthesizing a type for a variable preterm (with the usual de Brujin index representation)

1.Nat.Nat \vdash (var 0) \Rightarrow ?? \rightsquigarrow q

Context equations?

What does this mean for elaboration?

 Synthesizing a type for a variable (with the usual de Brujin index representation)

1.Nat.Nat ⊢ (var 0)
$$\Rightarrow$$
 Nat \rightsquigarrow q

$$\mathbf{1.Nat.Nat} \vdash (\mathsf{var}\ 0) \Rightarrow \Sigma(\mathbf{Nat}, \mathbf{Nat}) \rightsquigarrow \mathbf{q}$$

No longer deterministic! This is an issue even if we change variables to be checked

Context equations?

What does this mean for normalization?

- Contexts have normal forms!
- ► An algorithm for normalization (e.g. NbE) now must first normalize the context

Simply-typed lambda calculus

What about in the simpler setting of the simply-typed lambda calculus (STLC)?

Context equations:

$$\frac{\vdash \Gamma cx}{\vdash \Gamma. Unit = \Gamma cx} \qquad \frac{\vdash \Gamma cx \qquad A \text{ type} \qquad B \text{ type}}{\vdash \Gamma. (A * B) = \Gamma. A. B \text{ cx}}$$

Simply-typed lambda calculus

What about in the simpler setting of the simply-typed lambda calculus (STLC)?

Context equations:

$$\frac{\vdash \Gamma \operatorname{cx}}{\vdash \Gamma.\operatorname{Unit} = \Gamma \operatorname{cx}} \qquad \frac{\vdash \Gamma \operatorname{cx} \quad A \operatorname{type} \quad B \operatorname{type}}{\vdash \Gamma.(A * B) = \Gamma.A.B \operatorname{cx}}$$

► The context equations have the same effect on normalization and elaboration!

Current and future work

- NbE for STLC with unital and associative product types (including context equations)
- ► Elaboration for STLC with unital and associative product types (including context equations)
- Adapt both for MLTT
- Learn more about type theory as a polynomial monad and polynomial algebra

Thank you!

Questions?

References:

- Awodey, S. (2018). Natural models of homotopy type theory. Mathematical Structures in Computer Science, 28(2), 241-286.
- Awodey, S. and Newstead, C. (2018). Polynomial pseudomonads and dependent type theory. arXiv preprint arXiv:1802.00997.