Strictly Associative Sigmas (Work In Progress)

Emmanuel (Emma) Suárez Acevedo

MURI Meeting 2024

Martin-Löf Type Theory (MLTT)

There are the following judgements:

- **Contexts**: $\vdash \Gamma\ \mathsf{cx}$
- **Substitutions:** $\Delta \vdash \gamma : \Gamma$
- **Types:** $\Gamma \vdash A\ \mathsf{type}$
- **Terms:** $\Gamma \vdash a : A$

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

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

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

Idea

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

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

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

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?

Example

```
Poset : Set → Set₁
Poset X = Σ[ _≤_ ∈ (X → X → Set) ]
          -- reflexivity
          (∀ x → x ≤ x) ×
          -- antisymmetry
          (∀ x y → x ≤ y → y ≤ x → x ≡ y) ×
          -- transitivity
          (∀ x y z → x ≤ y → y ≤ z → x ≤ z)
```

Example

```
Semigroup : Set → Set
Semigroup S = Σ[ _+_ ∈ (S → S → S) ]
                -- _+_ is associative
                (∀ s₁ s₂ s₃ → s₁ + (s₂ + s₃) ≡ (s₁ + s₂) + s₃)

Monoid : Set → Set
Monoid M = Σ[ (_+_ , _) ∈ Semigroup M ]
           Σ[ e ∈ M ]
           -- e is a left and right identity
           (∀ m → e + m ≡ m) ×
           (∀ m → m + e ≡ m)
```

Example

```
-- Poset X : _≤_ × ≤/refl × ≤/antisym × ≤/trans
-- Semigroup S : _+_ × +/assoc
-- Monoid M : semigrp × e × +/identityˡ × +/identityʳ

PoMonoid M = Σ[ ((_·_ , _) , _ , _ , _) ∈ Monoid M ]
             Σ[ (_≤_ , _) ∈ Poset M ]
             -- compatibility of _·_ with _≤_
             (∀ x y z → x ≤ y → (x · z) ≤ (y · z)) ×
             (∀ x y z → x ≤ y → (z · x) ≤ (z · y))
```

Example

```
- Poset X : _≤_ × ≤/refl × ≤/antisym × ≤/trans
- Semigroup S : _+_ × +/assoc
- Monoid M : semigrp × e × +/identityˡ × +/identityʳ
- PoMonoid M : monoid  × poset × +/compatʳ × +/compatˡ
```

prop : ∀ {M}
    ((((_+_ , _) , e , _ , _) , (_≤_ , _ , _ , _) , _ , _) : PoMonoid M)
    → ∀ m → (e + m) ≤ m

prop . . .

Example

- Mental overhead to use the components of the pomonoid

```
- Poset X : _≤_ × ≤/refl × ≤/antisym × ≤/trans
- Semigroup S : _+_ × +/assoc
- Monoid M : semigrp × e × +/identityˡ × +/identityʳ
- PoMonoid M : monoid  × poset × +/compatʳ × +/compatˡ
```

$$
\begin{aligned}
&\mathrm{prop} : \forall\ \{M\} \\
&\qquad ((((\_{+}\_ \ , \ \_) \ , \ e \ , \ \_ \ , \ \_) \ , \ (\_{\leq}\_ \ , \ \_ \ , \ \_ \ , \ \_) \ , \ \_ \ , \ \_) : \mathrm{PoMonoid}\ M) \\
&\qquad \to \forall\ m \to (e + m) \leq m \\
&\mathrm{prop}\ \ldots
\end{aligned}
$$

Example

- What if we had strictly associative sigmas?

```
- Poset X : _≤_ × ≤/refl × ≤/antisym × ≤/trans
- Semigroup S : _+_ × +/assoc
- Monoid M : semigrp × e × +/identityˡ × +/identityʳ
- PoMonoid M : monoid  × poset × +/compatʳ × +/compatˡ
```

$$
\begin{aligned}
&\mathsf{prop} : \forall\,\{M\}\\
&\qquad ((\_{+}\_\,,\,\_\,,\,\mathrm{e}\,,\,\_\,,\,\_\;,\,\_{\leq}\_\,,\,\_\,,\,\_\,,\,\_\,,\,\_\,,\,\_) : \mathsf{PoMonoid}\ M)\\
&\qquad \to \forall\ \mathrm{m} \to (\mathrm{e} + \mathrm{m}) \leq \mathrm{m}\\
&\mathsf{prop}\ \ldots
\end{aligned}
$$

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

$$\begin{tikzcd} & B \arrow[dl, "s"'] \arrow[r, "f"] & A \arrow[dr, "t"] & \\ I & & & J \end{tikzcd}$$

Every morphism $f : B \to A$ in $\mathcal{E}$ is a polynomial $\mathbf{1} \nrightarrow \mathbf{1}$ (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** $i_I : I \nrightarrow I$ is $(\mathrm{id}_I, \mathrm{id}_I, \mathrm{id}_I)$

Review: polynomials

A **morphism of polynomials** $\varphi : p \Rrightarrow q$ 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:

$$\begin{tikzcd}
 & B \arrow[r, "f"] \arrow[dl, bend right, "s"'] \arrow[dd, "\varphi_1"'] \arrow[ddr, phantom, "\lrcorner", very near start] & A \arrow[dr, bend left, "t"] \arrow[dd, "\varphi_0"] & \\
I & & & J \\
 & D \arrow[r, "g"'] \arrow[ul, bend left, "u"] & C \arrow[ur, bend right, "v"'] &
\end{tikzcd}$$

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

Review: polynomials

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

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

$$\begin{tikzcd} B \arrow[r, "f"] \arrow[d, "\varphi_1"'] \arrow[dr, phantom, very near start, "\lrcorner"] & A \arrow[d, "\varphi_0"] \\ D \arrow[r, "g"'] & C \end{tikzcd}$$

Review: natural models

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

- a terminal object $\diamond$
- a *representable* map of presheaves $p : \dot{\mathrm{U}} \to \mathrm{U}$ on $\mathbb{C}$

(

(Awodey 2018)

Review: natural model and polynomials

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

(

(Awodey and Newstead 2018)

Review: natural model and **Unit** type

The model supports unit types iff there exists a cartesian morphism $\eta : i_{\mathbf{1}} \Rightarrow p$. Diagrammatically:

$$\begin{tikzcd}
\mathbf{1} \arrow[r, "\eta_1"] \arrow[d, equal] \arrow[dr, phantom, "\lrcorner", very near start] & \dot{\mathsf{U}} \arrow[d, "p"] \\
\mathbf{1} \arrow[r, "\eta_0"'] & \mathsf{U}
\end{tikzcd}$$

(Awodey and Newstead 2018)

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:

$$
\begin{tikzcd}
\sum\limits_{A:\mathsf{U}} \sum\limits_{B:\mathsf{U}^A} \sum\limits_{a:A} B(a) \arrow[r, "\mu_1"] \arrow[d, "p\cdot p"'] \arrow[dr, phantom, very near start, "\lrcorner"] & \dot{\mathsf{U}} \arrow[d, "p"] \\
\sum\limits_{A:\mathsf{U}} \mathsf{U}^A \arrow[r, "\mu_0"'] & \mathsf{U}
\end{tikzcd}
$$

(Awodey and Newstead 2018)

Review: polynomial monad

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

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

(

(Awodey and Newstead 2018)

Review: natural model and polynomial monads

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

For example:

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

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

((Awodey and Newstead 2018)

Review: natural model and polynomial monads

Is $(\mathbf{1}, p, \eta, \mu)$ 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.

(

(Awodey and Newstead 2018)

Review: natural model and polynomial monads

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

For example:

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

(Awodey and Newstead 2018)

Review: natural model and polynomial monads

Is $(\mathbf{1}, p, \eta, \mu)$ 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.

((Awodey and Newstead 2018)

Review: natural model and polynomial monads

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 $(\mathbf{1}, p, \eta, \mu)$ *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

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

Σ-types are unital

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

Σ-types are unital

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

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

Σ-types are associative

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

Σ-types are associative

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

Σ-types are associative

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

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

Context equations?

What does this mean for elaboration?

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

$$\frac{}{\mathbf{1}.\mathbf{Nat}.\mathbf{Nat} \vdash (\mathsf{var}\ 0) \Rightarrow \textcolor{red}{\mathbf{??}} \rightsquigarrow \mathbf{q}}$$

Context equations?

What does this mean for elaboration?

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

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

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

- 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:

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

Simply-typed lambda calculus

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

Context equations:

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

- 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.