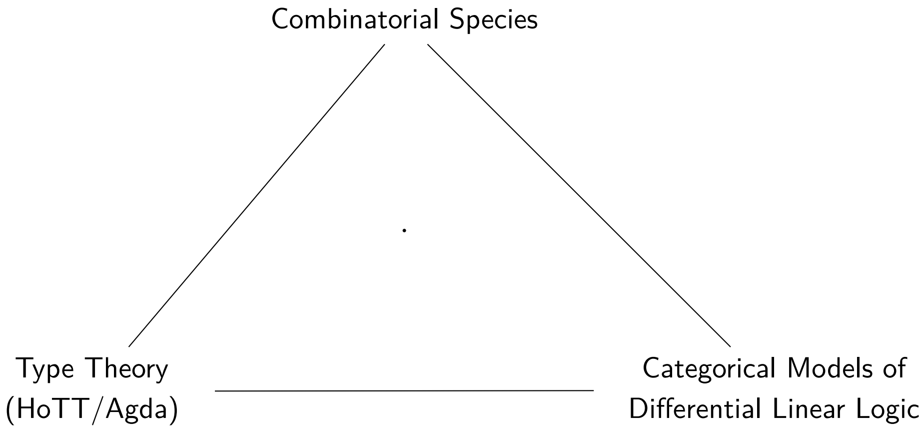
Generalised Species of Structures in Homotopy Type Theory Using Agda

13th June 2017

Introduction

Introduction

This project sits at the center of three main topics



Combinatorial Species

A combinatorial species consists of a rule, F, that associates

- with every finite set, U, a finite set, $F[U]$
- with every bijection, $\sigma : U \to V$, a bijection, $F[\sigma] : F[U] \to F[V]$

and satisifies

- $F[\mathrm{id}_U] = \mathrm{id}_{F[U]}$
- $F[\tau \circ \sigma] = F[\tau] \circ F[\sigma]$

A k-sorted species, F, acts on finite multisets, associating

- with every finite multiset, $U = (U_1, \dots, U_k)$, a finite set, $F[U_1, \dots, U_k]$
- with every bijective multifunction,

$$\sigma : (U_1, \dots, U_k) \to (V_1, \dots, V_k),$$

a bijection,

$$F[\sigma] : F[U_1, \dots, U_k] \to F[V_1, \dots, V_k]$$

Again this satisifies functoriality conditions

This notion of sorted species is linked to relational models of linear logic

Here we have operations on relations as the connectives of linear logic

$$A \otimes B :\equiv A \times B \qquad A \& B :\equiv A \uplus B$$

$$I :\equiv \{\star\} \qquad 1 :\equiv \varnothing$$

$$A \multimap B :\equiv A \otimes B$$

The exponential modality of linear logic is modelled by SM, the finite-multiset construction

- SM $A :\equiv \mathcal{M}_{fin}(A)$
- SM $f :\equiv \{([a_1, \dots, a_n], [b_1, \dots, b_n]) \mid \forall i.(a_i, b_i) \in f\}$

We can generalise the notion of relation between categories $\mathbb{C}$ and $\mathbb{D}$ as

$$\mathbb{C} \to \mathbb{D} \to \mathbf{Set}$$

We can also generalise the SM construction to categories

The category SM $\mathbb{C}$ has

- objects, finite sequences of objects of $\mathbb{C}$, $\llparenthesis c_i \rrparenthesis_{i=1,\ldots,n}$
- morphisms, pairs of bijections, $\sigma \in \boldsymbol{\sigma}_n$, and sequences of maps $\llparenthesis f_i : c_i \to c'_{\sigma i} \rrparenthesis_{i=1,\ldots,n}$

Generalised species of structures are defined as

$$\mathbb{C} \rightsquigarrow \mathbb{D} :\equiv \mathrm{SM}\,\mathbb{C} \to \mathbb{D} \to \mathbf{Set}$$

Implementation

The basis for the project is a categorical interpretation of homotopy type theory

Interpet

- Types as groupoids with morphisms given by the path space
- Type formers as categorical constructions, e.g. products
- The universe as the category $\mathbf{Set}$
- Therefore, functions as both functions and functors

Sequences of elements of a type C given by $\mathsf{List}\, C$

Now quotient by the relation of $\mathsf{ListPerm}\, C$

Quotient achieved using Higher Inductive Type (HIT) given by

$$
\begin{aligned}
&\texttt{HIT}\ \ \mathrm{Quot}_C(R) :\equiv \\
&\quad \mathrm{q} : C \to \mathrm{Quot}_C(R) \\
&\quad \mathrm{rel} : \Pi_{(x,y:C)}\ R\ x\ y \to \mathrm{q}\, x = \mathrm{q}\, y
\end{aligned}
$$

NB: This is actually quotienting by R^*

A more abstract formalisation is given by

$$\mathsf{SM}\,C \;:\equiv \Sigma_{(I\,:\,\mathcal{U})}\,(I \to C) \times \Sigma_{(n\,:\,\mathbb{N})}\,\|I \simeq \mathsf{Fin}\,n\|$$

The sequence of elements of C is indexed by the type I

This is forced to be finite by the proof of equivalence to $\mathsf{Fin}\,n$

The path space consists of bijections between finitely-indexed sets

The partial derivative of the species $P : A \rightsquigarrow B$ by $a : A$ is defined as

$$\partial_a \, P \, m \, b \;:\equiv P \, (m \cup [a]) \, b$$

Intuitively we view ${}^{P_n}\!/_{n!}$ as the coefficients of an exponential power series

$$p(x) :\equiv \sum_{n \geq 0} P_n \frac{x^n}{n!}$$

where differentiation shifts by 1

$$p'(x) :\equiv \sum_{i \geq 0} P_{i+1} \frac{x^i}{i!}$$

We can prove Leibniz Rule

$$\partial_a (P \boxtimes Q) = (\partial_a P \boxtimes Q) \boxplus (P \boxtimes \partial_a Q)$$

$$
\begin{aligned}
& \partial\, c\, (P \boxtimes Q)\, d\, m \\
\equiv\ & (P \boxtimes Q)\, d\, (m \cup [c]) \\
\equiv\ & \Sigma_{(m_1, m_2 : \mathsf{SMC})}\ P\, d\, m_1 \times Q\, d\, m_2 \times (m \cup [c] = m_1 \cup m_2) \\
=\ & \Sigma_{(m_1, m_2 : \mathsf{SMC})}\ P\, d\, m_1 \times Q\, d\, m_2 \\
& \qquad\qquad \times ((\Sigma_{(m' : \mathsf{SMC})}\ (m = m' \cup m_2) \times (m' \cup [c] = m_1)) \\
& \qquad\qquad\quad \sqcup \\
& \qquad\qquad\quad (\Sigma_{(m' : \mathsf{SMC})}\ (m = m_1 \cup m') \times (m' \cup [c] = m_2))) && \text{(combinatorial lemma)} \\
=\ & (\Sigma_{(m_1, m_2 : \mathsf{SMC})}\ P\, d\, m_1 \times Q\, d\, m_2 \\
& \qquad\qquad \times \Sigma_{(m' : \mathsf{SMC})}\ (m = m' \cup m_2) \times (m' \cup [c] = m_1)) \\
& \sqcup \\
& (\Sigma_{(m_1, m_2 : \mathsf{SMC})}\ P\, d\, m_1 \times Q\, d\, m_2 \\
& \qquad\qquad \times \Sigma_{(m' : \mathsf{SMC})}\ (m = m_1 \cup m') \times (m' \cup [c] = m_2)) \\
=\ & (\Sigma_{(m', m_2 : \mathsf{SMC})}\ P\, d\, (m' \cup [c]) \times Q\, d\, m_2 \times (m = m' \cup m_2)) \\
& \sqcup \\
& (\Sigma_{(m_1, m' : \mathsf{SMC})}\ P\, d\, m_1 \times Q\, d\, (m' \cup [c]) \times (m = m_1 \cup m')) && \text{(density formula twice)} \\
\equiv\ & (\partial\, c\, P \boxtimes Q) \boxplus (P \boxtimes \partial\, c\, Q)
\end{aligned}
$$

An equational reasoning proof

```
leibniz' : ∀ b m
         → fst (∂ a (P ⊠ Q) b m)
           ≃
           fst (((∂ a P ⊠ Q) ⊞ (P ⊠ ∂ a Q)) b m)
leibniz' b m =

  fst (∂ a (P ⊠ Q) b m)

    ≃⟨ Trunc-emap
        (S (S ⟨-2⟩))
        (Σ-emap-r (λ m₁ →
          Σ-emap-r (λ m₂ →
            ×-emap-r
              (fst (P b m₁))
              (×-emap-r
                (fst (Q b m₂))
                (coe-equiv (combinatorial-lemma m a m₁ m₂)))))) ⟩

  Trunc
    (S (S ⟨-2⟩))
    (Σ (SM (fst A)) (λ m₁ →
      Σ (SM (fst A)) (λ m₂ →
        fst (P b m₁)
        × fst (Q b m₂)
        × (Σ (SM (fst A)) (λ m' →
            (m == m' ∪ m₂) × (m' ∪ SM-⊤ a == m₁))
          ⊔
          Σ (SM (fst A)) (λ m' →
            (m == m₁ ∪ m') × (m' ∪ SM-⊤ a == m₂))))))

    ≃⟨ Trunc-emap
        (S (S ⟨-2⟩))
        (Σ-emap-r (λ m₁ →
          Σ-emap-r (λ m₂ →
            ×-emap-r
              (fst (P b m₁))
              Σ₂-⊔-equiv-⊔))) ⟩
```

```
Trunc
  (S (S ⟨-2⟩))
  (Σ (SM (fst A)) (λ m₁ →
    Σ (SM (fst A)) (λ m₂ →
      fst (P b m₁)
      × ((fst (Q b m₂)
        ×
        Σ (SM (fst A)) (λ m' →
          (m == m' ∪ m₂) × (m' ∪ SM-⊤ a == m₁)))
        ⊔
        (fst (Q b m₂)
        ×
        Σ (SM (fst A)) (λ m' →
          (m == m₁ ∪ m') × (m' ∪ SM-⊤ a == m₂)))))))

  ≃⟨ Trunc-emap
      (S (S ⟨-2⟩))
      (Σ-emap-r (λ m₁ →
        Σ-emap-r (λ m₂ →
          Σ₂-⊔-equiv-⊔))) ⟩

Trunc
  (S (S ⟨-2⟩))
  (Σ (SM (fst A)) (λ m₁ →
    Σ (SM (fst A)) (λ m₂ →
      (fst (P b m₁)
      × fst (Q b m₂)
      × Σ (SM (fst A)) (λ m' →
          (m == m' ∪ m₂) × (m' ∪ SM-⊤ a == m₁)))
        ⊔
      (fst (P b m₁)
      × fst (Q b m₂)
      × Σ (SM (fst A)) (λ m' →
          (m == m₁ ∪ m') × (m' ∪ SM-⊤ a == m₂))))))

  ≃⟨ Trunc-emap
      (S (S ⟨-2⟩))
      (Σ-emap-r (λ m₁ →
```

```
                (snd (P b x))
                (×-level
                  (snd (Q b m₂))
                  (SM-level (snd A) _ _))))
              (m' ∪ SM-⊤ a) ⁻¹)))
        (Σ-emap-r (λ m₁ →
          Σ-emap-r (λ m' →
            density
              (λ x →
                ( fst (P b m₁)
                  × fst (Q b x)
                  × (m == m₁ ∪ m')
                , ×-level
                    (snd (P b m₁))
                    (×-level
                      (snd (Q b x))
                      (SM-level (snd A) _ _))))
              (m' ∪ SM-⊤ a) ⁻¹)))) ⟩

  Trunc
    (S (S ⟨-2⟩))
    (Σ (SM (fst A)) (λ m' →
      Σ (SM (fst A)) (λ m₂ →
        fst (P b (m' ∪ SM-⊤ a))
        × fst (Q b m₂)
        × (m == m' ∪ m₂)))
    ⊔
    Σ (SM (fst A)) (λ m₁ →
      Σ (SM (fst A)) (λ m' →
        fst (P b m₁)
        × fst (Q b (m' ∪ SM-⊤ a))
        × (m == m₁ ∪ m'))))

    ≃⟨ Trunc-⊔-econv ⟨-2⟩ _ _ ⟩

  fst (((∂ a P ⊠ Q) ⊞ (P ⊠ ∂ a Q)) b m)

    ≃∎
```

Questions?