Lambda-Calculi for Logics — Valeria De Paiva

Lecture 3 - *June 25, 2025*

1 Linear Functional Programming

- It took us twenty years to combine categorical combinators and linear logic to create categorical abstract machines for linear functional programming. This brought us linear types and showed how category theory is baked into the language's syntax
- Although Rust and Haskell have linear type systems, they are not yet mainstream programming languages[1] (and linear type systems in general are not yet mainstream, but have shown much theoretical promise)

2 Explicit substitution

- Instead of implicit substitution via β-reduction, we want to have explicit substitution
- We have the syntax, but we don't yet have a semantic-level model for it
- Recall λ-calculus. We define the rules and terms, and computations are performed via β-reduction.
- In classical β-rule, the substitution mechanism usually involves replacing a variable within a term depending on the context. Traditionally, we store the bindings of terms in an collection, the language consisting of terms and environments.
- Implementing an efficient β-reduction can be difficult and error-prone. Making it explicit will make the reasoning and proofing more straightforward and easier
- The syntax of the λ-calculus should have a categorical semantics with mathematics as the source
- We create new rewrite rules that store substitutions in the environment instead of binding them. This will eliminate substitutions and work for various logic systems.
- The new contexts $z : A \times B$ and $x : A, y : B$ are isomorphic

3 $\lambda\sigma$-calculus

- λ-calculus + application of substitutions to a term: $f * t$
- Every $\lambda\sigma$-term is equivalent to some λ-term
- Contexts $z : A \times B$ and $x : A, y : B$ are related, but not equal
- Substitutions are judgments now, and can be paralleled and composed
- (Part of) rules for substitution evaluation:

$$\frac{\Gamma' \subseteq \Gamma}{\Gamma \vdash \langle\rangle : \Gamma'} \qquad \frac{\Gamma \vdash f : \Delta \qquad \Gamma \vdash t : A}{\Gamma \vdash \langle f, x \mapsto t\rangle : \Delta, x : A} \qquad \frac{\Gamma \vdash f : \Delta \qquad \Delta \vdash t : A}{\Gamma \vdash f * t : A}$$

$$\frac{\Gamma \vdash f : \Delta \qquad \Delta \vdash g : \Psi}{\Gamma \vdash f; g : \Psi}$$

- Composition of substitutions allows more interactions
- We bake substitution operators into language using a set of rewriting rules, but some issues...
  - Many design choices need to be made to get the best properties possible: which explicit substitutions? What methodology?
  - Ideally, contexts $z : A \times B$ and $x : A, y : B$ should be isomorphic
  - Counterexample idea: Cyclic substitution leads to non-termination during strong normalization
- We need a categorical, semantic way to express these

4 Categorical Semantics

- We want to extend the Curry-Howard correspondence to a categorical semantics
- Internal language gives term constructs and equational theory
- Indexed category theory as syntax debugging
- In an indexed category $D : C^{op} \to Cat$, contexts as objects, morphisms as substitutions
- Empty context as identity, assoc rewriting can be done in assoc compositions
- For every context Γ, we have a category $D(\Gamma)$
  - Objects are variable-type pairs $(x : A)$
  - Morphisms $(x : A) \to (t : B)$ are judgments of the form $\Gamma, x : A \vdash t : B$
- When re-indexing, we use $\Gamma \vdash f : \Delta$ to denote a functor $D(f) : D(\Delta) \to D(\Gamma)$

- Has some issues:
  - No syntax for forming substitutions from terms
  - Non-linear nature of indexed categories due to products of types (judgments of the shape $\Gamma, x : A \vdash x : A$)

5 Context-Handling Categories

- Solutions: Convert D^{op} into set $\mathcal{T}$; Add two new natural transformations
- Define a Symmetric Monoidal Category $\mathcal{B}$ with $\mathcal{T} \subseteq |\mathcal{B}|$
- Define a linear catersian context-handling category $L : \mathcal{B}^{op} \to Sets^{\mathcal{T}}$ with the following natural transformations:

$$\text{Sub}_A : L(-)_A \Rightarrow \mathcal{B}(-, A) \quad \text{Term}_A : \mathcal{B}(-, A) \Rightarrow L(-)_A$$

- We still keep the products, but SMCC for linear fibres, otherwise CCC
- We now have two categories E and L
- **E-Category**
  - Type constructors require extra structures on the category
  - Given $A, B \in \mathcal{T}$, we have $A \to B, A \times B \in \mathcal{T}$ with isomorphisms in Γ:

$$\frac{E((\Gamma, A)_B)}{E(\Gamma)_{A \to B}} \qquad \frac{E(\Gamma)_A \times E(\Gamma)_B}{E(\Gamma)_{A \times B}}$$

  - Naturality distributes explicit substitutions over the terms

$$f * \lambda x.t = \lambda x.f * t \quad f * (tu) = (f * t)(f * u)$$

- **L-Category**
  - Given $A, B \in \mathcal{T}$, we have $A \multimap B \in \mathcal{T}$ with isomorphisms in Γ:

$$\frac{E((\Gamma, A)_B)}{E(\Gamma)_{A \multimap B}}$$

  - Tensor product however is not isomorphic

$$z : A \otimes B \cong x : A, y : B$$

- **!-type constructor**
  - ! as the comonad of a monoidal adjunction between CCCs and SMCCs[2]
  - !L-category adjunct E and L cats together, and isomorphisms

  - Models of explicit substitutions
    - E-cat $\rightarrow, \times, \mathbb{1}$, $\lambda\sigma$-calculus
    - L-cat $\multimap, \otimes, I$

- E-categories and underlying CCCs are isomorphic
- Preserves the soundness and completeness for both E and L categories

6 Conclusions

- High-order category theory maps one category to a collection of categories and can be used to model explicit substitution - mathematical foundation has been shown
- Without explicit substitution, we wouldn't end up with a type theory with all of the properties we want/expect
- Contexts are helpful, but we can obtain equivalent systems without them. (Contexts make proofs more human-legible, but contextless systems are slightly more efficient for computers)
- Extension of Curry-Howard, easier than dependent type and modality
- Can be done in multiple potential modalities/substructural logics

References

[1] Jean-Philippe Bernardy et al. "Linear Haskell: practical linearity in a higher-order polymorphic language". In: *Proceedings of the ACM on Programming Languages* 2.POPL (Dec. 2017), pp. 1–29. ISSN: 2475-1421. DOI: 10.1145/3158093. URL: http://dx.doi.org/10.1145/3158093.

[2] P. N. Benton. "A Mixed Linear and Non-Linear Logic: Proofs, Terms and Models (Extended Abstract)". In: *Selected Papers from the 8th International Workshop on Computer Science Logic*. CSL '94. Berlin, Heidelberg: Springer-Verlag, 1994, pp. 121–135. ISBN: 3540600175.