Categorical semantics of ordered linear logic for a reconstruction of ownership in programming languages

Advisor: Guillaume Munch-Maccagnoni, Inria, Nantes

21st December 2022

Topics

- Semantics and logic
- Programming languages
- Category theory
- MPRI topics: 2.2, 2.1, (2.4)

Location Inria, Gallinette team, Laboratoire LS2N (Nantes)

Advisor Guillaume.Munch-Maccagnoni@Inria.fr

General presentation of the topic Linear logic (Girard, 1987, see also Melliès, 2009) is long thought to be relevant to the design and implementation of programming languages with a better awareness of resource usage (e.g. control over memory allocation, as early as Lafont 1988). In addition, resource-related features of programming languages commonly used in Rust (and so-called "modern" C++, starting from C++11), do indeed remind of linear logic in some aspects. Their central concept is that of resource, that is values of a type with an associated *destructor*: a clean-up action (such as freeing some memory, closing a file, releasing a lock. . .) that consumes the value.

However, linear logic, and more generally the traditional approaches to denotational semantics, have not yet developed an understanding of resource-management as found in practice in programming languages. In fact, these features originating from "systems" programming languages that appeared in the industry are not even so well treated in the general field of programming language theory.[1]

[1] For instance, not modelling destructors has been an explicit limitation of the works to model the Rust type system using separation logic (Jung et al., 2018), despite their importance for memory-safety and the known rough edges of their design in Rust.

In recent works, however, we have found hints that it is possible to reconstruct the main aspects of resource management in categorical semantics.[2] We hope that it can be a starting point to establish the groundwork for a principled and coherent understanding of these features in programming languages, and notably their successful future integration with functional programming.

This internship subject has as a starting point recent approaches to the categorical semantics of logic and computation, and it can interest mathematically-inclined young researchers who are interested in advancing the theory of programming languages and its connection to logic. The topic can evolve into a PhD subject in various directions, from more theoretical to more practical, in accordance with the tastes of the student.

Resource types and ordered logic The starting point of a rational reconstruction in the language of linear logic is to consider types with destructors as objects in a category with objects of the form $(A, \delta : A \to T1)$ —a *slice category*—, where $A \in \mathscr{C}$ is the base type of values, the starting category $\mathscr{C}$ of values is a model of linear or intuitionistic logic (λ-calculus), T is a strong monad (modelling the ambient effect, e.g. global state), and thus the destructor δ expects a value of type A and performs an effect before returning nothing. This construction has good properties, for instance one can model pairs of resource types $A' = (A, \delta_A)$ and $B' = (B, \delta_B)$ with a tensor product $A' \otimes B' \stackrel{\text{def.}}{=} (A \otimes B, \delta_{A \otimes B})$ where the destructor $\delta_{A\otimes B} : A \otimes B \to T1$ is obtained by doing δ_A followed by δ_B via monadic composition.

Since δ_A and δ_B can perform side-effects, this can be different from doing δ_B followed by δ_A, and so $A' \otimes B'$ and $B' \otimes A'$ are very different types in general (just like structs in Rust/C++). Linear logics that reject the isomorphism $A' \otimes B' \ncong B' \otimes A'$ are called *ordered* or *non-commutative* (Lambek, 1958).

Ownership In order to control mutability in programming languages (Reynolds, 1978), various notions of *uniqueness* or *ownership* have been proposed as dynamic or static (type-based) restrictions on the usage of values since the 1990's. With resource types, Rust/C++11 propose a notion of ownership (expanding the so-called *RAII*[3] pattern from earlier C++) which was at odds with the literature up to that point. It is more general, dealing with resource abstractions beyond memory. From this concept nevertheless emerges notions of linear (affine) type systems, region types (borrowing), and uniqueness (control of aliasing)—other aspects much better studied in the literature.

The long-term goal is to show how notions related to ownership (linear type systems, region typing, uniqueness) follow from a rational reconstruction in the context of categorical semantics of programming languages, and how an abstract view advantageously informs the design of programming languages.

Linear Call-by-Push-Value The framework in which this development takes place is the *call-by-push-value (CBPV)* (Levy, 2004), a model of higher-order computation with side-

[2] Combette and Munch-Maccagnoni, 2018

[3] "Resource acquisition is initialization"

effects, a decomposition of call-by-value λ-calculus motivated by denotational semantics; more precisely *linear CBPV* developed by the advisor and his colleagues (Curien, Fiore, and Munch-Maccagnoni, 2016), which is the focus of the internship.

Linear call-by-push-value is a model of higher-order computation that describes the proper way for combining side-effects (Moggi, 1991) and resources (Girard, 1987). It unifies CBPV and Melliès's tensor logic (Melliès and Tabareau, 2010), and it makes the expected connexions between (linear) logic and (effectful) computation. However, the results are currently to (commutative) linear logic.

Goals We would like to extend the logic and the results from Curien, Fiore, and Munch-Maccagnoni (2016) into an "ordered call-by-push-value", to model the non-commutative phenomenon mentioned before, and to serve as a starting point for further investigating notions related to ownership in programming languages.

The internship task is to understand this paper with the help of the advisor, and, as a research in team with the advisor, to investigate the difficulties in the non-commutative case, and lastly, if successful, to contribute to the publication of the results in an international conference or journal.

Pre-requisites As a young researcher, you probably already have:

- outstanding creativity,
- taste and capacity for acquiring a bibliographic knowledge of a topic,
- teamwork,
- clear and rigorous writing,
- good oral presentation skills.

More specifically for this topic, and less importantly, one expects:

- starting knowledge and interest in categorical semantics of logic and computation,
- interest or curiosity in systems programming (e.g. Rust), or proof theory (e.g. linear logic, focusing), or old-school linguistics (where non-commutative logic originates from), etc.

Notes

- Our team Gallinette provides a nice atmosphere. It is a large and young team of researchers in Nantes, specialised in logic, programming languages, and the formalisation of mathematics. Many colleagues work on the development and application of the Coq proof assistant.

- My speciality is in logic and denotational semantics (e.g. lambda-calculus). More recently, I have found this area of application of my works to programming languages[4], and I started hacking and contributing to the OCaml language runtime. I interact both with researchers in logic/semantics and with OCaml/Rust researchers and implementers in the industry.
- Funding is available for travel (e.g. workshops and visits abroad). There are collaboration opportunities with M. Fiore (Univ. Cambridge, UK).
- In addition, a student with taste in programming will also have the opportunity, if they want, to get involved in side-projects meant to improve OCaml's support for safe resource management in a concrete manner.

References

Guillaume Combette and Guillaume Munch-Maccagnoni. 2018. *A resource modality for RAII (abstract)*. Technical Report. INRIA. `https://hal.inria.fr/hal-01806634`

Pierre-Louis Curien, Marcelo Fiore, and Guillaume Munch-Maccagnoni. 2016. A Theory of Effects and Resources: Adjunction Models and Polarised Calculi. In *Proc. POPL*. `https://doi.org/10.1145/2837614.2837652`

Jean-Yves Girard. 1987. Linear Logic. *Theoretical Computer Science* 50 (1987), 1–102.

Ralf Jung, Jacques-Henri Jourdan, Robbert Krebbers, and Derek Dreyer. 2018. RustBelt: securing the foundations of the rust programming language. *PACMPL* 2, POPL (2018), 66:1–66:34. `https://doi.org/10.1145/3158154`

Yves Lafont. 1988. The linear abstract machine. *Theoretical computer science* 59, 1-2 (1988), 157–180.

Joachim Lambek. 1958. The mathematics of sentence structure. *The American Mathematical Monthly* 65, 3 (1958), 154–170.

Paul Blain Levy. 2004. *Call-By-Push-Value: A Functional/Imperative Synthesis*. Semantic Structures in Computation, Vol. 2. Springer.

Nicholas D. Matsakis and Felix S. Klock II. 2014. The rust language. In *ACM SIGAda Ada Letters*, Vol. 34. ACM, 103–104.

Paul-André Melliès. 2009. *Categorical semantics of linear logic*. Panoramas et Synthèses, Vol. 27. Société Mathématique de France, Chapter 1, 15–215.

Paul-André Melliès. 2012. Parametric monads and enriched adjunctions. (2012). Draft.

[4]See my presentation of this topic at the Collège de France: https://www.college-de-france.fr/site/xavier-leroy/seminar-2018-12-19-11h30.htm.

Paul-André Melliès and Nicolas Tabareau. 2010. Resource modalities in tensor logic. *Ann. Pure Appl. Logic* 161, 5 (2010), 632–653.

Eugenio Moggi. 1991. Notions of computation and monads. *Inf. Comput.* 93, 1 (July 1991), 55–92. https://doi.org/10.1016/0890-5401(91)90052-4

John C. Reynolds. 1978. Syntactic Control of Interference. In *Conference Record of the Fifth Annual ACM Symposium on Principles of Programming Languages, Tucson, Arizona, USA, January 1978*, Alfred V. Aho, Stephen N. Zilles, and Thomas G. Szymanski (Eds.). ACM Press, 39–46. https://doi.org/10.1145/512760.512766

Bjarne Stroustrup, Herb Sutter, and Gabriel Dos Reis. 2015. A brief introduction to C++'s model for type- and resource-safety. (2015). http://www.stroustrup.com/resource-model.pdf