General presentation of the topic  A major evolution in programming languages in the recent years has been the advent of modern resource-management features in the systems programming languages C++11 (Stroustrup et al., 2015) and Rust (Matsakis and Klock II, 2014). In what one might initially think unrelated, many recent advances in the theory of programming languages are due to the Curry-Howard correspondence between functional programming, logic, and category theory.

In our team, we have been studying some of these systems programming features, via the Curry-Howard correspondence, from the point of view of linear logic (Girard, 1987; Melliès, 2009; Baker, 1994), and established a formal connection with its sibling ordered (or non-commutative) logic (Combette and Munch-Maccagnoni, 2018). This opened further research directions on the general theme of merging of systems programming and functional programming (Munch-Maccagnoni, 2018) (with the ultimate goal of creating the Next Best Programming Language).
Of these research directions, some can interest mathematically-inclined young researchers who are interested in advancing the theory of programming languages and its connection to logic, others can interest those who are more looking for a hands-on approach to programming language development. This internship subject has as a starting point recent approaches to the categorical semantics of logic and computation, and can evolve into a PhD subject in either direction, in accordance with the tastes of the student.

**Context** Recently, we have proposed the notion of ordered algebraic data types to model the types of resources in C++11/Rust: those types with an associated destructor, a clean-up action (such as freeing some memory, closing a file...) that has to be called reliably and predictably at the end of the value’s lifetime (Munch-Maccagnoni and Douence, 2019). Ordered algebraic data types are expressive compound types whose associated destructor is determined by the type; for instance there are two types of list: $\mu X.(1 \oplus A \otimes X)$ defines the type of lists whose elements are destroyed in list order, whereas $\mu X.(1 \otimes X \oplus A)$ defines the type of lists whose elements are destroyed in reverse order.\(^1\) ($A \otimes X$ and $X \otimes A$ are distinct types: this is in fact where the terminology ordered and non-commutative comes from.)

This proposition has deeper roots in Combette and Munch-Maccagnoni (2018) where we proposed a categorical model construction that interprets these resource types in a semantics of non-commutative logic (Lambek, 1968). This is stated in the framework of linear Call-by-Push-Value (LCBPV) (Curien et al., 2016) which is the focus of the internship.

**Goals** Linear Call-by-Push-Value is a model of computation with higher-order functions and evaluation order that describes the proper way of combining effects (Moggi, 1991) and linearity (Girard, 1987), featuring a neat correspondence between (linear) logic and (effectful) computation. However, its results are currently limited for our purposes since they only concern (commutative) linear logic. We would like to extend the results from Curien et al. (2016) into a “non-commutative Call-by-Push-Value”, to serve as a starting point for integrating into the theory of functional programming the modelling of phenomena and features of systems programming (e.g. ordered algebraic data types, borrowing à la Rust, and more).

The internship task is to understand the paper Curien et al. (2016) with the help of the advisor, and, as a research in team with the advisor, to extend the result as desired, and, if successful, to contribute to the publication of the results in an international conference or journal.

**Pre-requisites**
- Knowledge and strong interest in categorical semantics of logic and computation,

\(^1\) In Rust:

```rust
pub struct List<A> { node: Option<Box<(A, List<A>)>>, }
pub struct List2<A> { node: Option<Box<(List2<A>, A)>>, }
```
• Interest or curiosity in systems programming (e.g. Rust), proof theory, linguistics, etc.

Remarks

• There are collaboration opportunities with M. Fiore (Univ. Cambridge, UK) with funding available for travelling (whenever sanitary conditions allow).

• In addition, a student with taste in programming will also have the opportunity to get involved in side-projects meant to improve OCaml's support for safe resource management in a concrete manner.

References


