Internship proposal:
Formalisation of Scheduled Types in Coq

Advisors. Flavien Breuvart and Micaela Mayero

Keywords. functional programming, dependents types, static analysis, Coq

Potential collaborations. Dan Ghica, Damiano Mazza,

Prerequisite. Bases of functional programming, logic and proof assistants. Expertise is at least one of those more advanced topics: Coq, dependent types, static analysis.

Localisation. LIPN, University Paris 13

Principle

This internship aims at formalising, in *Coq proof assistant*, an ongoing research project by Breuvart and Ghica called *Scheduled Types*.

A schedule type is a type of the form $l_1{\cdot}A_1 \to l_2{\cdot}A_2 \to J \bullet B$ where A_1, A_2 and B are scheduled types, where l_1 and l_2 are labels, and where J is an abstract object called *schedule* that "represents" the extensional behaviour of the execution of the programme once given its arguments. A basic example of schedule is a rational expression over the alphabet $\{l_1, l_2, e\}$ where e represent an effect, in this case, if $J = l_2((e + l_2)l_1)^*$ this means that the second argument is always called at the beginning, then there is an alternation of events with the first being either an effect or a call over the second argument, and where the second is a call over the first argument. In call-by-name evaluation, these kind of information is crucial for static analysis. Notice that the schedules can be more complex, but have to respect a certain algebraic structure.

Such type systems are endowed with a complex notion of scope and composition. The associated proofs of subject reduction and realizability semantics are then extremely intricated. In such situation, making mistakes is quite frequent and the assistance of a computer is required to verify our proofs: this leads to the need of Coq. Such proofs tend to be especially smooth in Coq due to their syntactical nature.

References

No published paper, yet, but in the same spirit, you can look at graded type systems. Notice, however, that those are simpler in the sense that they do not carry dependency on labels.

CoGITARe Project and fully funded PhD

This internship can be followed by a PhD fully funded by the ANR CoGITARe on the thematic.[1]

Type systems are used to automatically check security properties of large programs. CoGITARe's goal is to extend typing methodology to a large panel of properties currently unreachable by state-of-the-art techniques, enabling in particular the analysis of quantitative properties of programs.

We will develop a way to keep track of the extensional information inside types in order to perform the whole static analysis at the level of types. For this purpose, we will combine two (re)emerging type systems, namely graded types and intersection type systems, with the well established techniques from the field of abstract interpretation such as widening.

[1] In the project, Scheduled types are called Game-like types.

Graded type systems formally embed a first order structure within types, while intersection types will help to circumvent the unconditional non-compositionality of fine grained resource analyses. This is how we plan to tackle the long running problem of applying abstract interpretation result in functional programming.