Semantics of Probabilistic Reactive Programming

2021 Internship Proposals supervised by

Guillaume Baudart  
guillaume.baudart@ens.fr

Christine Tasson  
Christine.tasson@lip6.fr

Keywords: Programming Languages ; Reactive Programming; Probabilistic Programming ; Operational and Denotational Semantics; Linear Logic

We propose an internship which will lead to a PhD funded by a research project. The subject of this internship is the design and understanding of the semantics of probabilistic reactive systems.

1 Scientific context

1.1 Semantics of Programming Languages

The famous Curry-Howard correspondence establishes a bridge between programming languages and logic through type systems. This bridge can extends to a third part, semantics, built upon two aspects:

- semantics gives an interpretation of programs independent from syntax design together with mathematical tools that allow computer scientists to study property of programs
- features of mathematical semantics often reveal programming structures that can be imported into syntax.

Linear Logic [1] illustrates this back and forth between languages, logic and semantics. Linearity has been introduced in logic after it reveals to be an important features in models of intuitionistic logic. The corresponding type system has slowly infused in the programming language community. For instance the key concept of resources in Linear Logic shares similarities with resources used in the Rust programming language.

Mathematical models for programming languages range over partially ordered sets, topological spaces, diffeological spaces, measurable spaces or categories. These frameworks have to be adapted to fit with the language they interpret. For instance, in order to model higher order programming languages, currying and evaluation have to live inside the model. That is why standard mathematical framework often have to be tuned as illustrated in probabilistic programming. Indeed, the standard model of measurables spaces does not fit with higher-order. This gives rise to new questions in mathematics and in computer science.

1.2 Probabilistic Systems

Probabilistic systems have been studied from decades and their semantics has been already explored [2].

Probabilistic programming has been intensively developed in recent years, with the final purpose of encoding in the same setting various probabilistic models appearing in artificial
intelligence and applied to statistical computing and machine learning. Numerous programming languages using higher-order functions and continuous probabilities have been developed, e.g., WebPPL, Venture, Anglican, Stan, Gen, Pyro... This universe is challenging from a semantical viewpoint and it is only recently that the meaning of these languages has been elucidated.

Two models of linear logic have been studied by Ehrhard, Pagani and Tasson: Probabilistic Coherence Spaces (PCS) [3], that model discrete probabilistic computations and Measurable Cones and Stable Functions [4], a conservative extension of PCS that model continuous probabilistic computations. Other settings have been successfully explored by the community to model higher-order probabilistic functional programming such as Quasi-Borel Spaces [5] and Ordered Banach Spaces [6].

1.3 Reactive Systems

Reactive systems are computer systems embedded within larger environments with which they are in continuous interaction. Such systems are ubiquitous, from airplane control software to robotics. They are very often designed and implemented using dedicated languages, from high-assurance languages like SCADE [7] for safety critical systems, to more free-form modeling languages like The Mathworks’ Simulink [8].

Having a precise mathematical understanding of this family of languages is of great importance since it is often used in life-critical contexts. Fortunately, it turns out that their semantics can be cast inside the same general framework as functional programming [9–12]. Indeed, this variety of reactive programming can be understood as an instance of functional programming making pervasive yet highly-disciplined use of infinite data structures, first and foremost streams (infinite sequences). Specifically, reactive programs manipulate streams according to the synchronous programming discipline [10, 13, 14]. A synchronous program executes in a succession of discrete time steps. A programmer writes high level specifications in the form of stream functions specifying variable values at each step.

Zelus is a synchronous language extended with Ordinary Differential Equations (ODEs) to program hybrid systems that mix discrete-time and continuous-time models. An example is a discrete-time model of a control software paired with a continuous-time model of the plant. Zelus combines programming constructs from the Lustre and Lucid Synchrone [15] programming languages with continuous-time models expressed by ODEs and zero-crossing events.

2 Internships

2.1 Probabilistic Semantics of Reactive systems

Reactive-language designers have been adding probabilistic constructs to their languages. For instance, Baudart and his collaborators have extended the Zelus language [16] to the probabilistic reactive language ProbZelus [17]. This extension is motivated by the rising need for statistics in the physical modeling of embedded systems. Because of the high complexity of current physical environments, deterministic models are often intractable. On the contrary, simulations combined with statistical inference can be faithful to reality yet tractable. This recent area of research has promising applications to the design of embedded systems [18].

The first objective of the internship is to design a semantics of the probabilistic reactive language ProbZelus based on an extension of probabilistic coherence spaces.
2.2 Probabilistic and Differential Semantics of Reactive systems

Probabilistic programming languages come with various inference algorithms, for instance Importance Sampling, Particle Filtering (used in ProbZelus), Metropolis-Hasting, Hamiltonian Monte Carlo and Variational Inference. The last two algorithms rely on differentiation. In order to understand their semantics, it is necessary to combine continuous probability with differentiation. This is an active area of research with many contributions on various models [19–22].

A complete solution for the framework of PCS and Measurable Stable Cones is left open. Yet, the first steps in this direction have been made by Thomas Ehrhard with the introduction of differentiation in lambda-calculus and its study in probabilistic coherence spaces [23, 24] and Coherent Differentiation.

The second objective of the internship is to understand the right setting to combine differentiation with continuous probabilities and higher order functional programming in the extension of PCS that has been designed to handle differentiation. In the meantime, inference algorithms based on differentiation will be encoded in ProbZelus. The semantics that has been designed will be used to ensure good properties of this implementation.

3 Bibliography

References


