Présentation du Projet de l'équipe ReaLiSe

Titre : **ReaLiSe** - **Rea**ctive systems, **Li**near logic and **Se**mantics, towards an application to probabilistic programming

Coordination scientifique :	Christine Tasson
Participants de l'IRIF	Thomas Ehrhard, Claudia Faggian, Adrien Guatto, Michele Pagani, Yann Régis-Gianas, Alexis Saurin
DIENS	Marc Pouzet
IBM	Guillaume Baudart, Louis Mandel

Whatever the context (robots, planes or computer assisted cars) embedded software have to react in real time and to adapt to their continuously evolving environment. Moreover, such critical softwares have to ensure strong safety properties. That is why the computer science community has been developing dedicated programming languages so called reactive and synchronous. Nowadays, reactive languages have to include modern approaches such as Artificial Intelligence. Indeed, statistical models can model physical environment and be used for training controllers in reactive systems. Over the last years, many probabilistic programming languages have been introduced in order to describe statistical models. The probabilistic paradigm has raised new challenges for the computer science community and turned to be a rich and trendy line of research. For instance, at IBM research - one of the largest industrial research organizations in the world - the AI Programming Models team is working in collaboration with MIT and DIENS on robust and reactive probabilistic modeling.

The scientific coordinator has been working on functional programming following the tradition of semantics and linear logic in IRIF. She has then turned to applications with a view to test the formal methods she has developed with her collaborators to the test of practice. She will now focus on reactive programming and its probabilistic extension. The scientific coordinator (*Christine Tasson*) has gathered three young researchers with complementary specialities needed in the project: semantics of reactive programming *(Adrien Guatto)*, proof theory and logical time *(Alexis Saurin)* and development of probabilistic reactive programming *(Guillaume Baudart)*. The four of them will be supported by five senior researchers that will bring their own expertise.

A. Scientific Context

This project starts from the observation that two communities, in the semantics of programming languages, use similar tools in quite different ways: specialists of functional programming languages on one side, and specialists of reactive and synchronous languages — as those used for developing embedded systems — on the other side. In both cases, a major issue is to describe conceptually simple mathematical models of programs, called *semantics*, in order to reason about program correctness and inform programming language design. Both communities design and study semantics using very similar frameworks developped by logicians and computer scientists. Yet, the way they use them differs by the role given to *time*.

Functional programming is a programming paradigm where programs are functions, computing output values from input values, and assembled through function composition. This paradigm has guided the design of several programming languages, including OCAML, F#, and HASKELL, and influenced many more, such as PYTHON, RUST, and R.

The theoretical background of functional programming is Alonzo Church's λ-calculus, originally proposed in the 1940s as a foundation for mathematics. The λ-calculus specifies the behavior of programs via *rewriting* and, more generally, *operational* semantics. Operational semantics, conceptually speaking, describes the step-by-step execution of programs on idealized machines. But this notion of execution step is generally not considered relevant, or intrinsic, to the meaning of programs. For this reason, it is very helpful to abstract from the very concrete nature of operational semantics: this is the role of *denotational* semantics. Denotational semantics, which stems from the seminal work of Scott and Strachey [SS71], interprets programs as functions, or more generally morphisms, between mathematical structures. The function which *denotes* a program provides an abstract invariant stable under execution, and far removed from the concrete details of syntax.

Denotational semantics allows logicians and programming language designers to tap into mathematics for inspiration. A prototypical example of this phenomenon is the analogy between certain denotational semantics and linear algebra, which led Jean-Yves Girard to linear logic [Gir87]. Intuitively, linear logic is based on a *symmetric* interaction between programs and environments in which no duplications of informations are possible: in denotational models of linear logic such interactions are represented by the mathematical concept of *linear duality*, at the core of linear algebra. As such, linear logic describes in a mathematically convincing way rather concrete aspects of program execution, aspects which were previously only captured in operational

semantics. This fresh perspective has been very fruitful in the last decade, leading for example to finer-grained syntactic accounts of resource-sensitiveness (differential lambda-calculus, differential linear logic) and to new perspectives on infinite objects (linear logic with fixpoints and infinitary proofs) and effectful computations (including probabilities, see *infra*). However, linear logic does not appear to shed light on the *temporal* aspect of program execution, which becomes crucial when trying to extend functional programming to 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 [Est19] to more free-form modeling languages like The Mathworks' Simulink [Mat19].

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 [Kah74, CPHP87, CP96, EH97]. 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 and other infinite data structures according to the *synchronous* programming discipline [CPHP87, BG92, BCE+03]. Explaining this notion requires discussing the role and nature of *time*.

In a reactive system, *time* is there from the very beginning: the system is being continuously fed an infinite stream of inputs by the environment, while simultaneously producing a stream of outputs. Each time step corresponds to an interaction point between program and environment, at which they may or may not decide to exchange data. Synchrony corresponds to the idea that program and environment agree on the amount of data exchanged per time step, and do not interact outside of these well-defined exchanges. The synchronous point of view on *time* is thus logical: time describes interactions between a program and its environment, rather than more mundane aspects of computation (e.g., physical running time on a computer). Thus, as one can guess, *time* has to be a basic ingredient in all attempts at developing mathematical models of such computations: structural operational semantics representing their executions as infinite sequences of instantaneous reactions [BG92], denotational models are based on streamlike structures, and categorical models can be based on the topos of trees [BMSS12a, Gua18a]. In such models, the order structure reflect the program-environment alternation.

In practice, functional reactive languages mostly distinguish themselves from ordinary functional programming by the presence of dedicated type systems called *clock calculi* [Cas92, CP03, CDE+06]. Clock calculi ensure that the complex time-related operations present in a program (e.g., buffering) still result in consistent exchanges with the environment, even in the presence of feedback loops. However, while clock calculi can deal with intricate communication patterns inbetween time steps, existing reactive languages have largely turned a blind eye to what happens *within* a time step, especially when compared to the precise and versatile accounts of computation obtained via linear logic. Dually, the missing account of *time*, understood from the synchronous perspective, in linear logic is especially galling since recent works [BDS16, DBHS, BDKS19] have shown that it can accomodate infinite data structures in a natural way.

Probabilistic programming is another programming paradigm going beyond purely functional computation. It has been intensively developed in recent years, with the final purpose of encoding in the same settings various probabilistic models appearing in artifical 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.

In this area, Kozen has developed a measure-theoretic semantics, using *Markov kernels* to describe the *operational semantics* of probabilistic programs [Koz81]. Then, building on the work of Giry [Gir82] and Moggi [Mog89], Jones and Plotkin [JP89] introduced a *denotational semantics* based on probabilistic power domains, which led to rich research programs. Recently, Kozen and Silva used this approach based on domains to model networks with probabilistic features [SKF+17]. In parallel, Danos and Ehrhard introduced probabilistic coherent spaces, a *denotational semantics*, where *formal series* denote programs [DE11]. The scientific coordinator Tasson and her collaborators Ehrhard and Pagani have proved that this model is precise enough to statically characterize the computational behavior of programs handling discrete probabilities [EPT14]. A major obstacle had then to be overcome to model continuous probabilities and *higher-order* functional programming languages (programs that can handle functions as arguments) [Pan99]. To our knowledge, there are only few models of probabilistic programming handling higher-order and continuous probabilities: quasi-Borel spaces [SYW+16, HKSY17, SKV+18, VKS19], the measurable cones designed by the scientific coordinator and her collaborators [EPT18a] and the ordered Banach spaces [DK20]. These success relied on properties inspired by linear logic and functional programming.

In the meantime, and independently from works on the semantics of functional probabilistic programming, reactive-language designers have been adding probabilistic constructs to their languages. For instance, Baudart and his collaborators have extended the Zélus language [BBCP19] to the probabilistic reactive language ProbZélus [BMA+19]. 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, especially when used online. On the contrary, simulations combined with statistical inference can be faithful to reality yet tractable. This is an extremely recent area of research, with promising applications to the design of embedded systems [BMP+20]. Moreover, the semantical study of such extensions will have to involve modern statistics and the theory of stochastic processes, which, to our knowledge, have hardly ever been connected to core model-theoretical concepts of reactive and functional programming.

B. Research Project

Linearity and synchrony both achieve a fine-grained study of the interaction between a program and its environment. Moreover, they use very similar technical tools, e.g., operational semantics, denotational models stated in category-theoretical terms, or modal formulæ. Yet, nobody has considered them together, or tried to integrate them. Given the recent evolution of the two subjects, we believe that the time is ripe for this endeavor.

Outline. The emergent team we want to construct aims at bridging the two communities of reactive programming and functional programming. We will build our collaboration on the construction of a common knowledge base (**Theme 1**). We will then be in a position of tackling our purposes: understand the semantics of probabilistic reactive languages and guide the design of related programming constructs (**Theme 2**).

Theme 1: Time, Effects, and Resources in Reactive Programming and Linear Logic

In brief:	Cross-fertilize semantics of reactive programming and linear logic.
Goals:	**G1** - Infinite objects in linear logic
	G2 - Time and resources in linear logic and reactive programming
	G3 - Effects in reactive programming, from a linear-logic viewpoint
Fallout:	Finer-grained semantics of reactive programming, enrichments of linear logic.
Collaborators:	T. Ehrhard, A. Guatto, M. Pagani, M. Pouzet, A. Saurin, C. Tasson.

Goals. Recent advances in and around linear logic suggest potential connections with the semantics of reactive programming. We expect the influences to go in both directions: refining the semantics of reactive programs with ideas from linear logic, but also enriching linear logic with stylized facilities for reactive programming.

G1 - Reactive programming deals with infinite data structures such as streams. Thus, as a first bridge with linear logic, we are interested in casting such data structures in a proof-theoretic framework. We want to start from the recent work of Alexis Saurin and his collaborators on the understanding of infinite objects and fixpoints in linear logic [BDS16, DBHS, BDKS19]. This syntactic work will be complemented by a model-theoretic study informed by existing approaches to recursive types [Ehr16] as well as inductive and coinductive types [San02, FS13, BDS15] in linear logic: while finitary proofs for logic with least and greatest fixed points can be given a semantics, no denotational semantics is known for non well-founded proofs.

G2 - On the one hand, linear logic has led to a thorough description of the duplication and sharing of data that occurs when a program and its environment interact. Models of the synchronous discipline, on the other hand, give a precise account of the temporal continuity of this interaction. We want to combine both aspects inside a single setting, in order to refine our understanding of higher-order computation. It is especially exciting that these two aspects do not appear to be orthogonal: restricting the number of duplications allowed per time step might force a program into a pattern of temporal interaction with its environment. This kind of space-time tradeoff is essential to many important program optimizations which, up to now, have never been studied from the semantics viewpoint [LS91, Fea92a, Fea92b].

G3 - The synchronous discipline finds a very natural model in the topos of trees, a simple categorical setting for well-behaved recursion [BMSS12b, Gua18b]. It can be construed as a time-oriented layer grafted on top of a purely functional basis. However, emerging application domains suggest adopting richer notions of computation in this base layer. Such notions can often be presented through linear logic models [Has02, Ehr16]. This suggests, as a first step, trying to replace the category of sets and functions lying at the heart of the topos of trees with a model of linear logic. In a second step, we want to design generic constructions giving rise to a model of reactive programming starting from models of linear logic.

Theme 2: Mathematics of Probabilistic Reactive Programming

In brief:	Build a mathematical understanding of probabilistic reactive programming.
Goals:	**G4** - Analyze higher-order functional probabilistic programming
	G5 - Use this analysis for certification
	G6 - Describe semantics for probabilistic reactive programming language design
Fallout:	Mathematical understanding of the design of probabilistic reactive languages.
Collaborators	G. Baudart, T. Ehrhard, C. Faggian, L. Mandel, M. Pagani, M. Pouzet, Y. Régis-Gianas, C. Tasson.

Goals. Probabilistic programming raises new challenges in programming languages. They are used to design new statistic models from primitive blocks such as sampling, observation, conditioning and inference. Because of samplings, programs handle values that are generated at run-time. How to implement inference operators is a key question which can be seen as the exploration of traces of execution combined with the scheduling of the queries to the randomized input [GCSR04]. Certification of probabilistic programs and of their analysis is a crucial challenge that can be reached only with theoretical advances in the comprehension of their semantics.

The operational theory of a language can be formalized as a notion of rewriting; we need theoretical tools to support the study of operational properties in probabilistic computation. The notion of Probabilistic Abstract Reduction Systems (PARS) has been introduced by Bournez and Kirchner in [BK02], yet work in this direction is only beginning [DLFVY17, DM17, KC17, ADLY18, Fag19].

G4 - The recent development in semantics [HKSY17, EPT18a, BFK+18, Ehr20] have produced models unveiling different aspects of probabilistic programming. For example, one of our most recent contribution shows an unexpected analogy between the semantical notion of stability [Ber78, Gir86], corresponding to (a light form of) sequential computation, with the mathematical property of absolute monotonicity in the models of probabilistic programming [EPT18b]. We plan to further develop this link, introducing derivatives in probabilistic programming and trying to extend our result of full-abstraction from a countable setting, like probabilistic PCF [ETP14, ET19], to uncountable structures, such as the set of real numbers [EPT18b, Ehr20]. Another tasks of this goal is to focus on operational semantics, developing the tools and frameworks for comparing different evaluation strategies of probabilistic programs. Call-by-name, call-by-value, call-by-push-value etc. have a different flavour when you do not rewrite a single expression but a probabilistic distribution of possible expressions. Standard rewriting properties like confluence, termination, standardisation need a quantitative setting in order to be able to compare and design evaluation strategies for probabilistic programs. Such a theory is essential to allows for program transformations, optimizations, distributed implementations. Our goal is to develop this setting, building on recent progress – in the formalism of probabilistic lambda calculus [FR19, Lev19].

G5 - We want to apply the semantics construction that we have designed and will continue to develop for the certification of probabilistic programs, that is proving that the implementations of probabilistic models or algorithms correct with respect to their specification. We will use the Coq proof assistant. The ALEA library of Christine Paulin is based on the monadic approach [Koz81] and we want to renew it with other approaches to semantics. This goal is timely as the library COQUELICOT has describe most of the tools will need to deal with reals. Moreover, we will benefit from the development of the library EASYCRYPT proposing another approach of probabilistic programs based on the probabilistic Hoare Logic.

G6 - Baudart and his coauthors have introduced the first synchronous probabilistic programming language [BMA+19] with facilities for probabilistic models and *inference* (which allows to compute distributions conditionned by observation from statistical data). In this paper, they have proposed an inference engine compatible with the reactive setting, and they have proved its soundness by transformation to a first order language in which they were able to describe the probabilistic reactive semantics. They have also given examples of applications to path detection and robotics controler [BMP+20].

In order to move from first order to higher-order languages, we propose to start from the semantics of higher-order probabilistic languages developped by the PI and her collaborators [EPT18a]. It is noteworthy that infinite data structures at play in reactive programming (streams, infinite trees…) have a very natural interpretation in those probabilistic models. Moreover, we will be able to exploit the linear logic structure of this semantics described in [Ehr20]. Thus, we plan to study these available infinite data structures to give account to time in streams models in an approach reminiscent of the stream semantics of synchronous languages [Gua18b] and using the bridges developed in **Theme 1**.

Defining a precise semantics will give us a theoretical framework to inform language design, prove future compiler optimizations, reason about program equivalence, and verify the correctness of applications.

C. Organization

Results. We will create a web page dedicated to the project, that will publicize the meetings, workshops, demos and results of the project. As the project is theoretical, the fallouts will be mainly publications in both theoretical computer science and programming language top tier international conferences and journals.

Bimonthly meetings. The central tools of the team will be the bimonthly workshop where the common socle of knowledge will be established through presentations of the works of the two communities. As two members of the team are abroad (the IBM research lab is based in the US), we will use the visioconference material of the university to enable distant working sessions. The first year will be mainly devoted to tutorial by the participants of the team. In the following years, we will alternate invited talks of external researchers to feed our work, presentation of results by the participants and working sessions on specific aspects of the project.

All the material of the talks will be gathered on the website of the project and will constitute a reference guideline to the subject of the project ReaLiSe.
The project coordinator, assisted by its permanent participants, will supervise the project by:

Stimulating exchanges between the various participants of the project and push towards achieving the goals of the project. This will specifically be done at the occasion of regular meetings of the project. Since the members of the project come from slightly different communities (combinatorics, categorical algebra, logic, semantics and programming), we will organize tutorial sessions to lay the foundations for interactions. Meetings will be mainly dedicated to brainstorming and collaborative work in order to favor the emergence of original scientific ideas and of collaborations. A particular attention will also be paid by task coordinators to analyze potential difficulties that might occur in achieving the various tasks of the project as well as new perspectives.

Supervising reviews of the literature as well as transmission of expertise between the members of the project themselves and between the project members and foreign researchers outside the project. To that end, we will invite foreign researchers to give talks to our meetings and to stay for short terms visits.

Coordinating recruitment process for positions opened in the project (Post-Doc fellows, PhD student and Master interns). This will include attendance of the PhD student to young researchers summer schools.

Ensuring the dissemination of the results of the project by a coordinated effort towards publication of these results as well as presentations in seminars and working groups and workshops. We will also organize a workshop at the end of the project, with the participation of international researchers. We will centralize the results of the project in a website and broadcast them through official open access channels of our institutions (CNRS, INRIA and Universities).

Paying attention to the future of the project, in particular by planning discussions on the future of the project during year 3 of the project.

C.1 Collaborations

The project will crucially require exchanges with well identified collaborators, e.g. Ugo Dal Lago (Bologne), Pierre Clairambault (ÉNS Lyon), Fredrik Dahlqvist (London), Ohad Kammar (Edinburgh), Sam Staton (Oxford), Lionel Vaux (Marseille).

C.2 Budget

The duration of ReaLiSe will be 4 years. As the scientific coordinator, Tasson will devote 70% of her research time to the project. She will conduct the research project she has designed by working on and coordinating every task. However, to reach her goals, ReaLiSe requires a close collaboration with her team.

One PhD student will be hired for **Theme 2**, she will design *the semantical tools to study probabilistic reactive programming*. One postdoctoral position will be funded for benefiting from her expertise in probabilistic programming. Hopefully, one other PhD student will be financed through École doctorale, École Normale, or Polytechnique, recruited for instance in the master courses taught by seven of the participants. Participants will gain autonomy from this project that will result in a new group in the team of the coordinator. There will be regular national meetings, one cofunded international workshop and several international researchers invitations.

The budget of ReaLiSeof 280k€ is divided in 250k€ funded by Paris, 20k€ funded by FSMP and 10k€ funded by the IRIF lab. It will be mainly used for hiring one PhD student (110k€) and 18 months of postdoctoral position (90k€) and for funding a teaching discharge of six months for the scientific coordinator over the first year of the project (20k€). The other planned expenses will be dedicated to the organization of the team working groups (10k€), to participation to national meetings (5k€), to invitation of five international researchers for one month (25.5k€), paying master students stipends (4.5k€) and to dissemination (10k€) with missions and co-organization of one international workshop and for computers (5k€).

D. References

[ADLY18] Martin Avanzini, Ugo Dal Lago, and Akihisa Yamada. On Probabilistic Term Rewriting. In *Symposium on Functional and Logic Programming, FLOP*, 2018.

[BBCP19] Albert Benveniste, Timothy Bourke, Benoît Caillaud, and Marc Pouzet. Zélus: a synchronous language with ODEs. http://zelus.di.ens.fr, 2019.

[BCE+03] Albert Benveniste, Paul Caspi, Stephen A. Edwards, Nicolas Halbwachs, Paul Le Guernic, and Robert de Simone. The Synchronous Languages 12 Years Later. *Proceedings of the IEEE*, 2003.

[BDKS19] D. Baelde, A. Doumane, D. Kuperberg, and A. Saurin. Bouncing threads for infinitary and circular proofs. unpublished draft, 2019.

[BDS15] David Baelde, Amina Doumane, and Alexis Saurin. Least and greatest fixed points in ludics. In Stephan Kreutzer, editor, *24th EACSL Annual Conference on Computer Science Logic, CSL 2015, September 7-10, 2015, Berlin, Germany*, volume 41 of *LIPIcs*, pages 549–566. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2015.

[BDS16] D. Baelde, A. Doumane, and A. Saurin. Infinitary proof theory: the multiplicative additive case. In *CSL 2016*, 2016.

[Ber78] Gérard Berry. Stable models of typed lambda-calculi. In *Proceedings of the 5th International Colloquium on Automata, Languages and Programming*, number 62 in Lec. Notes Comput. Sci. Springer-Verlag, 1978.

[BFK+18] Giorgio Bacci, Robert Furber, Dexter Kozen, Radu Mardare, Prakash Panangaden, and Dana Scott. Boolean-valued semantics for the stochastic λ-calculus. In *LICS*, pages 669–678. ACM, 2018.

[BG92] Gérard Berry and Georges Gonthier. The ESTEREL synchronous programming language: design, semantics, implementation. *Science of computer programming*, 19(2):87–152, 1992.

[BK02] Olivier Bournez and Claude Kirchner. Probabilistic rewrite strategies. applications to ELAN. In *RTA*, volume 2378 of *Lecture Notes in Computer Science*. Springer, 2002.

[BMA+19] Guillaume Baudart, Louis Mandel, Eric Atkinson, Benjamin Sherman, Marc Pouzet, and Michael Carbin. Reactive probabilistic programming, 2019.

[BMP+20] Guillaume Baudart, Louis Mandel, Marc Pouzet, Eric Atkinson, Benjamin Sherman, and Michael Carbin. Programmation d'applications réactives probabilistes. *31ème Journées Francophones des Langages Applicatifs*, pages 63–71, 2020.

[BMSS12a] Lars Birkedal, Rasmus Ejlers Møgelberg, Jan Schwinghammer, and Kristian Støvring. First steps in synthetic guarded domain theory: step-indexing in the topos of trees. *Logical Methods in Computer Science*, 8(4), 2012.

[BMSS12b] Lars Birkedal, Rasmus Ejlers Møgelberg, Jan Schwinghammer, and Kristian Støvring. First steps in synthetic guarded domain theory: step-indexing in the topos of trees. *Logical Methods in Computer Science*, 8(4), 2012.

[Cas92] Paul Caspi. Clocks in dataflow languages. *Theoretical Computer Science*, 94(1):125–140, 1992.

[CDE+06] Albert Cohen, Marc Duranton, Christine Eisenbeis, Claire Pagetti, Florence Plateau, and Marc Pouzet. N-synchronous Kahn networks: a relaxed model of synchrony for real-time systems. In *Principles of Programming Languages (POPL'06)*, 2006.

[CP96] Paul Caspi and Marc Pouzet. Synchronous Kahn Networks. In *International Conference on Functional Programming (ICFP'96)*. ACM, 1996.

[CP03] Jean-Louis Colaço and Marc Pouzet. Clocks as First Class Abstract Types. In *International Conference on Embedded Software (EMSOFT'03)*. ACM, 2003.

[CPHP87] Paul Caspi, Daniel Pilaud, Nicolas Halbwachs, and John Plaice. LUSTRE: A declarative language for programming synchronous systems. In *Principles of Programming Languages (POPL'87)*, 1987.

[DBHS] A. Doumane, D. Baelde, L. Hirschi, and A. Saurin. Towards completeness via proof search in the linear time μ-calculus: The case of Büchi inclusions. In *LICS 2016*.

[DE11] V. Danos and T. Ehrhard. Probabilistic coherence spaces as a model of higher-order probabilistic computation. *Inform. Comput.*, 2011.

[DK20] Fredrik Dahlqvist and Dexter Kozen. Semantics of higher-order probabilistic programs with conditioning. *PACMPL*, 4(POPL):57:1–57:29, 2020.

[DLFVY17] Ugo Dal Lago, Claudia Faggian, Benoît Valiron, and Akira Yoshimizu. The geometry of parallelism: classical, probabilistic, and quantum effects. In *Symposium on Principles of Programming Languages, POPL*, 2017.

[DM17] Alejandro Díaz-Caro and Guido Martinez. Confluence in probabilistic rewriting. In *LSFP2017, Workshop on Logical and Semantic Frameworks with Applications*, 2017.

[EH97] Conal Elliott and Paul Hudak. Functional Reactive Animation. In *International Conference on Functional Programming (ICFP'97)*. ACM, 1997.

[Ehr16] Thomas Ehrhard. Call-by-push-value from a linear logic point of view. In *ESOP*, volume 9632 of *Lecture Notes in Computer Science*, pages 202–228. Springer, 2016.

[Ehr20] Thomas Ehrhard. On the linear structure of cones, 2020.

[EPT14] Thomas Ehrhard, Michele Pagani, and Christine Tasson. Probabilistic coherence spaces are fully abstract for probabilistic PCF. In *POPL*, pages 309–320. ACM, 2014.

[EPT18a] Thomas Ehrhard, Michele Pagani, and Christine Tasson. Measurable cones and stable, measurable functions: a model for probabilistic higher-order programming. In *POPL*, pages 59:1–59:28. ACM, 2018.

[EPT18b] Thomas Ehrhard, Michele Pagani, and Christine Tasson. Measurable cones and stable, measurable functions: a model for probabilistic higher-order programming. *PACMPL*, 2(POPL):59:1–59:28, 2018.

[Est19] Esterel Technologies & ANSYS. SCADE Suite. http://www.esterel-technologies.com/products/scade-suite/, 2019.

[ET19] Thomas Ehrhard and Christine Tasson. Probabilistic Call By Push Value. *Logical Methods in Computer Science*, 15(1), 2019.

[ETP14] Thomas Ehrhard, Christine Tasson, and Michele Pagani. Probabilistic coherence spaces are fully abstract for probabilistic PCF. In Suresh Jagannathan and Peter Sewell, editors, *POPL*, pages 309–320. ACM, 2014.

[Fag19] Claudia Faggian. Probabilistic rewriting: Normalization, termination, and unique normal forms. In *4th International Conference on Formal Structures for Computation and Deduction, FSCD 2019*, volume 131 of *LIPIcs*, pages 19:1–19:25. Schloss Dagstuhl, 2019.

[Fea92a] Paul Feautrier. Some efficient solutions to the affine scheduling problem. I. One-dimensional time. *International journal of parallel programming*, 21(5):313–347, 1992.

[Fea92b] Paul Feautrier. Some efficient solutions to the affine scheduling problem. Part II. Multidimensional time. *International journal of parallel programming*, 21(6):389–420, 1992.

[FR19] Claudia Faggian and Simona Ronchi Della Rocca. Lambda calculus and probabilistic computation. In *34th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2019, Vancouver, BC, Canada, June 24-27, 2019*, pages 1–13. IEEE, 2019.

[FS13] Jérôme Fortier and Luigi Santocanale. Cuts for circular proofs: semantics and cut-elimination. In *CSL*, volume 23 of *LIPIcs*, pages 248–262. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2013.

[GCSR04] Andrew Gelman, John B. Carlin, Hal S. Stern, and Donald B. Rubin. *Bayesian Data Analysis*. Chapman and Hall/CRC, 2nd ed. edition, 2004.

[Gir82] Michèle Giry. *A categorical approach to probability theory*, pages 68–85. Springer Berlin Heidelberg, Berlin, Heidelberg, 1982.

[Gir86] Jean-Yves Girard. The system F of variable types, fifteen years later. *Theor. Comput. Sci.*, 45:159–192, 1986.

[Gir87] Jean-Yves Girard. Linear logic. *Theor. Comput. Sci.*, 50:1–102, 1987.

[Gua18a] Adrien Guatto. A Generalized Modality for Recursion. In *Logic in Computer Science (LICS'18)*, 2018.

[Gua18b] Adrien Guatto. A generalized modality for recursion. In *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science*, LICS '18, pages 482–491, New York, NY, USA, 2018. ACM.

[Has02] Masahito Hasegawa. Linearly Used Effects: Monadic and CPS Transformations into the Linear Lambda Calculus. In Zhenjiang Hu and Mario Rodríguez-Artalejo, editors, *Functional and Logic Programming, 6th International Symposium, FLOPS 2002, Aizu, Japan, September 15-17, 2002, Proceedings*, volume 2441 of *Lecture Notes in Computer Science*, pages 167–182. Springer, 2002.

[HKSY17] Chris Heunen, Ohad Kammar, Sam Staton, and Hongseok Yang. A convenient category for higher-order probability theory. In *LICS*, pages 1–12. IEEE Computer Society, 2017.

[JP89] Claire Jones and Gordon Plotkin. A probabilistic powerdomains of evaluation. In *Proceedings of the 4th Annual IEEE Symposium on Logic in Computer Science LICS-1989* [lic89].

[Kah74] Gilles Kahn. The semantics of a simple language for parallel programming. In *Information Processing Congress (IFIP'74)*. IFIP, 1974.

[KC17] Maja H. Kirkeby and Henning Christiansen. Confluence and convergence in probabilistically terminating reduction systems. In *LSFA, Workshop on Logical and Semantic Frameworks with Applications*, 2017.

[Koz81] Dexter Kozen. Semantics for probabilistic programs. *Journal of Computer and System Sciences*, 22, 1981.

[Lev19] Thomas Leventis. A deterministic rewrite system for the probabilistic λ-calculus. *Mathematical Structures in Computer Science*, 29(10):1479–1512, 2019.

[lic89] Pacific Grove, CA , USA, June 5–8, 1989. IEEE Computer Society Press.

[LS91] Charles E. Leiserson and James B. Saxe. Retiming Synchronous Circuitry. *Algorithmica*, 1991.

[Mat19] Mathworks, The. Simulink - Simulation and Model-Based Design. https://www.mathworks.com/products/simulink.html, 2019.

[Mog89] Eugenio Moggi. Computational lambda-calculus and monads. In *Proceedings of the 4th Annual IEEE Symposium on Logic in Computer Science LICS-1989* [lic89].

[Pan99] Prakash Panangaden. The category of markov kernels. *Electronic Notes in Theoretical Computer Science*, 22:171 – 187, 1999. PROBMIV'98, First International Workshop on Probabilistic Methods in Verification.

[San02] Luigi Santocanale. A calculus of circular proofs and its categorical semantics. In Mogens Nielsen and Uffe Engberg, editors, *Foundations of Software Science and Computation Structures, 5th International Conference, FOSSACS 2002. Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2002 Grenoble, France, April 8-12, 2002, Proceedings*, volume 2303 of *Lecture Notes in Computer Science*, pages 357–371. Springer, 2002.

[SKF+17] Steffen Smolka, Praveen Kumar, Nate Foster, Dexter Kozen, and Alexandra Silva. Cantor meets scott: semantic foundations for probabilistic networks. In *POPL*, pages 557–571. ACM, 2017.

[SKV+18] Adam Scibior, Ohad Kammar, Matthijs Vákár, Sam Staton, Hongseok Yang, Yufei Cai, Klaus Ostermann, Sean K. Moss, Chris Heunen, and Zoubin Ghahramani. Denotational validation of higher-order bayesian inference. *PACMPL*, 2(POPL):60:1–60:29, 2018.

[SS71] D. Scott and C. Strachey. Toward A Mathematical Semantics for Computer Languages. In *Proceedings of the Symposium on Computers and Automata*, volume XXI, pages 19–46, 1971.

[SYW+16] Sam Staton, Hongseok Yang, Frank D. Wood, Chris Heunen, and Ohad Kammar. Semantics for probabilistic programming: higher-order functions, continuous distributions, and soft constraints. In *LICS*, pages 525–534. ACM, 2016.

[VKS19] Matthijs Vákár, Ohad Kammar, and Sam Staton. A domain theory for statistical probabilistic programming. *PACMPL*, 3(POPL):36:1–36:29, 2019.