A Bird's Eye View on the Quantitative Semantics of Linear Logic

Michele Pagani
Laboratoire d'Informatique de Paris Nord
Institut Galilèe – Universitè de Paris 13
Villetaneuse, France
`michele.pagani@lipn.univ-paris13.fr`

Since the inception of Linear Logic [6], there has been an increasing interest in applying linear algebra to the formal methods approach to computation. Data types are interpreted as vector spaces, the addition expressing a kind of superposition of atomic states and the scalars a quantitative estimation of such a superposition. Programs using their inputs exactly once correspond then to linear functions, while typical programs, using their inputs several (or none) times are represented by power series.

These are the original intuitions of Girard's *quantitative semantics*, and date back to [7]. Depending on the field of scalars, these semantics can model quantitative behaviors of programs, such as runtime estimations or resource analysis. Moreover, vectors are able to model overlapping of "inconsistent" information, a feature which becomes crucial when one wants to interpret non-deterministic computations or quantum data types. Indeed, recently we succeeded in developing very solid and precise models of probabilistic and quantum functional languages [2, 5, 11], filling rather big gaps in the literature.

The goal of this presentation is to gently introduce some of the main ideas and most recent results in quantitative semantics.

1 Programs as power series

Take a functional program M of type $\mathtt{1} \to \mathtt{1}$, where $\mathtt{1}$ is the unit type whose only value is `skip`. The basic idea of quantitative semantics is to interpret M as a power series (centered at 0):

$$[\![M]\!] = \sum_{n=0}^{\infty} p_n x^n \,.$$

The unknown x corresponds to the input of M and the exponent n refers to the number of times M will call x in order to give an output. The whole series gathers all the possible number of calls for x. The coefficient p_n is the part specific to the program M, giving a weight to the possibility that M actually uses the input n times. Indeed, if M is a deterministic program (e.g. a regular λ-term), then p_n will be zero everywhere except for at most one monomial. The degree of this monomial tells us how many times M needs to use the input value `skip` in order to give the output `skip`. However, this simple situation changes as we move to more complex data types or languages.

The series has only one unknown because the input type has dimension one. The dimension of a ground type is the number of its values. Take then the boolean type `Bool` with two values `tt` and `ff`, and consider now M of type $\mathtt{Bool} \to \mathtt{1}$. Its denotation is a power series with two unknowns:

$$[\![M]\!] = \sum_{n=0}^{\infty}\sum_{h=0}^{\infty} p_{n,h} x_{\mathtt{tt}}^{n} x_{\mathtt{ff}}^{h} \,,$$

Submitted to:
SOS 2007

where a single monomial $x_{\texttt{tt}}^{n}x_{\texttt{ff}}^{h}$ expresses the computations calling the input $n+h$ times, this input behaving n times as `tt` and h times as `ff`. Such semantics have a built-in form of non-determinism, allowing for a non-uniform behavior of the various calls for the input.

Finite multisets give a convenient notation for multivariable series:

$$[\![M]\!] = \sum_{m\in\mathscr{M}_{\mathrm{f}}(\{\texttt{tt},\texttt{ff}\})} p_m x_{\texttt{tt}}^{m(\texttt{tt})} x_{\texttt{ff}}^{m(\texttt{ff})} = \sum_{m\in\mathscr{M}_{\mathrm{f}}(\{\texttt{tt},\texttt{ff}\})} p_m x^m\,,$$

where $\mathscr{M}_{\mathrm{f}}(\{\texttt{tt},\texttt{ff}\})$ is the set of the finite multisets over $\{\texttt{tt},\texttt{ff}\}$, and $m(\texttt{tt})$, $m(\texttt{ff})$ denote the number of occurrences in m of, respectively, `tt` and `ff`. Also, the x in the last series is a variable of dimension two, representing the two original unknowns $x_{\texttt{tt}}$ and $x_{\texttt{ff}}$.

If we consider output types with more than one possible value, then the denotation describes a family of power series and not a sole one. If M has type $\texttt{Bool}\to\texttt{Bool}$, we have two multivariable power series:

$$[\![M]\!]_{\texttt{tt}} = \sum_{m\in\mathscr{M}_{\mathrm{f}}(\{\texttt{tt},\texttt{ff}\})} p_{m,\texttt{tt}} x^m, \text{ and} \qquad [\![M]\!]_{\texttt{ff}} = \sum_{m\in\mathscr{M}_{\mathrm{f}}(\{\texttt{tt},\texttt{ff}\})} p_{m,\texttt{ff}} x^m,$$

with two different families of coefficients. Indeed, once fixed input and output types, the only information needed to express such power series are the monomial coefficients. Hence, the denotation can be presented as a matrix, whose lines are indexed by finite multisets (describing monomials), whose columns are indexed by the possible output values (describing a specific series) and whose entries are the coefficients:

$$[\![M]\!] = \begin{array}{r c} & \begin{array}{cc} \underset{\downarrow}{\texttt{tt}} & \underset{\downarrow}{\texttt{ff}} \end{array} \\ \begin{array}{r} []\rightarrow \\ [\texttt{tt}]\rightarrow \\ [\texttt{ff}]\rightarrow \\ [\texttt{tt},\texttt{ff}]\rightarrow \\ \vdots \end{array} & \left(\begin{array}{cc} p & q \\ p' & q' \\ p'' & q'' \\ p''' & q''' \\ \vdots & \vdots \end{array}\right) \end{array} \tag{1}$$

For example, the coefficient $[\![M]\!]_{[\texttt{tt},\texttt{tt},\texttt{ff}],\texttt{ff}}$ is the coefficient of the monomial $x_{\texttt{tt}}^{2}x_{\texttt{ff}}$ in the series describing the computations of M returning `ff`.

Linear logic proofs are represented by linear functions (i.e. families of power series of degree one). This is the ideal setting to enlighten one among the most astonishing features of linear logic: the correspondence between the cut-elimination behavior of logical rules and the standard constructions of linear algebra, alluded by the linear logic jargon (tensor $\otimes$, direct sum $\oplus$, dual space, etc...). For example, the splitting of the classical sequent rules into multiplicatives and additives can be motivated with respect to two different behaviors under cut-elimination: branching and inter-communication. In the quantitative models, this splitting corresponds to two different ways of aggregating linear functions: direct products and tensors, both defined by universal properties.

There are in the literature a multitude of quantitative semantics. I list here some choices which must be taken in order to implement a concrete model. The list is not at all exhaustive, but I hope that it will give an idea of the different features of these models.

Scalars. In Girard's normal functors model [7], scalars are possibly infinite sets. Ehrhard's Köthe sequences spaces [3] and finiteness spaces [4] recast Girard's intuitions on actual vector spaces, taking scalars from standard fields: Köthe sequences spaces consider the field of real numbers $\mathbb{R}$ as well as that of complex numbers $\mathbb{C}$, whilst the finiteness spaces construction work over any field. In the weighted relational models [9], scalars can be taken from any continuous commutative semi-ring. In the model developed by Selinger, Valiron and myself [11] the coefficients are completely positive maps of finite dimension.

Convergence. To choose scalars we must also consider a notion of convergence of a series of scalars. This is a crucial issue of the model, necessary to compose power series (hence to have a category) and to see them as well-defined functions from inputs to outputs. A shortcut is by postulating that every series converges everywhere. This can be done by endowing scalars with a complete order, having the bottom element 0 (the neutral element of the sum) and the top element ∞ (which is absorbing with respect to the sum), and then postulating that the "morally" diverging series will value ∞. This choice is taken in the models in [9] and [11]. The price (not so high) to pay is that scalars must be always non-negative and so data types are modules rather than vector spaces.

Ehrhard's models [3] and [4] adopt less obvious topologies (in particular, Köthe sequences spaces use the standard topology of real and complex numbers), allowing for divergent series, but restricting the hom-sets to *continuous* power series, which always compose and converge absolutely in the vector space associated with the input type. The other side of the coin is that this restriction makes the hom-sets not cpo-enriched, hence failing to model fix-point combinators and the untyped λ-calculus.

Powers. Girard's translation of the functional type of programs $A \to B$ into the linear logic formula $!A \multimap B$ is the bridge between linear functions and power series. In particular, the promotion $!A$ is the operation lifting a vector x of type A into the space $!A$ of "powers" of type A. The question is: what is a "power" of x? Here is another point where quantitative models may differ considerably.

In [9] and [11], the exponential $!A$ is defined as the infinite bi-product of the symmetric n-fold tensor powers of A, following the formula:

$$!A = \bigoplus_{n=0}^{\infty} A^n, \tag{2}$$

the intuition being that the n-th layer A^n of such a bi-product contains the monomials of degree n. The fact that we are considering symmetric tensors amounts to say that the order in which unknowns appear is irrelevant.

Equation (2) does not work in Ehrhard's models, namely infinite products and infinite co-products are different in finiteness and Köthe sequence spaces. However, Melliès et al. showed that the exponentials of finiteness spaces can be obtained via a slightly different formula [10, 16]. In all models mentioned so far, the exponential comonoid is the free commutative comonoid, however the freeness is not necessary to model linear logic exponentials. For sure, we are ignoring many other notions of exponentials in these semantics, which might have an interest for modeling operational properties. Blute et al. proposed to consider, for example, the exponentials generated by the antisymmetric tensor [1].

2 Two examples of applications

Probabilistic computing. Probabilistic coherence spaces [8, 2] yield a quite intuitive example of quantitative semantics. Scalars are non-negative real numbers, hence vectors can express probabilistic distributions of data. Indeed, in collaboration with Ehrhard and Tasson, we proved that this semantics is fully abstract with respect to the contextual equivalence of call-by-name PCF extended with a random primitive [5].

The proof uses innovative tools which might be useful to study probabilistic programming from a semantical viewpoint. In a language with random functions, two programs should be considered different not only when they give different results, but also when they give the same result but with different probabilities. Showing this difference can be much harder than in a deterministic language. Indeed, it requires a sharp control over coefficients expressing probabilities. Probabilistic coherence spaces denote programs with power series, this allows us to use standard tools of Calculus for handling probabilities.

Quantum computing. An important problem in the semantics of quantum languages is how to combine quantum computing with higher-order functions, or in other words, how to design a functional quantum programming language. A syntactic answer to this question was arguably given with the design of the quantum λ-calculus [17, 14]. This language has a well-defined syntax and operational semantics. However, the question of how to give a denotational semantics to the language turned out to be difficult, and has remained open for many years [13, 15]. One reason that designing such a semantics is difficult is that quantum computation is inherently defined on *finite dimensional* Hilbert spaces, whereas the semantics of higher-order functional programming languages, including such features as infinite data types and recursion, is inherently infinitary.

In a joint project with Selinger and Valiron, we give a quantitative semantics of a higher-order quantum language with full recursion and an infinite data type [11].

The starting point is Selinger's category CPM of *completely positive maps* [12]. This category is suitable for implementing first-order quantum languages, but it has no higher-order duplication and, consequently, cannot model higher-order languages. Basically, we extend CPM in order to make meaningful Equation (2), allowing for linear logic exponentials. The result is a model interpreting a higher-order quantum program as an infinite dimensional block matrix whose blocks are finite dimensional completely positive maps.

This explains and illustrates the distinction between the quantum and classical parts of the language. The quantum part is described by completely positive maps (finite dimension), whereas the classical control is given by the structure of Lafont category (i.e., linear logic). The model demonstrates that the two "universes" work well together, but also, surprisingly, that they do not mix too much, even at higher-order types (we always have an *infinite* matrix of *finite* dimensional CPMs). The control flow is completely handled by the biproduct completion, and not by the CPM structure.

Our main result is the adequacy of the model with respect to the operational semantics.

References

[1] R. F. Blute, Prakash Panangaden & R. A. G. Seely (1994): *Fock Space: A Model of Linear Exponential Types*.

[2] Vincent Danos & Thomas Ehrhard (2011): *Probabilistic coherence spaces as a model of higher-order probabilistic computation*. Inf. Comput. 209(6), pp. 966–991.

[3] Thomas Ehrhard (2002): *On Köthe sequence spaces and linear logic*. Math. Struct. Comput. Sci. 12, pp. 579–623.

[4] Thomas Ehrhard (2005): *Finiteness spaces*. Math. Struct. Comput. Sci. 15(4), pp. 615–646.

[5] Thomas Ehrhard, Michele Pagani & Christine Tasson (2014): *Probabilistic Coherence Spaces are Fully Abstract for Probabilistic PCF*. In P. Sewell, editor: The 41th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL14, San Diego, USA, ACM.

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

[7] Jean-Yves Girard (1988): *Normal functors, power series and lambda-calculus*. Ann. Pure Appl. Logic 37(2), pp. 129–177.

[8] Jean-Yves Girard (2004): *Between Logic and Quantic: a Tract*. In Thomas Ehrhard, Jean-Yves Girard, Paul Ruet & Philip Scott, editors: Linear Logic in Computer Science, London Math. Soc. Lect. Notes Ser. 316, CUP.

[9] J. Laird, G. Manzonetto, G. McCusker & M. Pagani (2013): *Weighted relational models of typed lambda-calculi*. In: 28th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS 2013), 25-28 June 2013, New Orleans, USA, Proceedings, pp. 301–310.

[10] Paul-André Melliès, Nicolas Tabareau & Christine Tasson (2009): *An Explicit Formula for the Free Exponential Modality of Linear Logic*. In: Automata, Languages and Programming, LNCS 5556, Springer, pp. 247–260.

[11] Michele Pagani, Peter Selinger & Benoit Valiron (2014): *Applying Quantitative Semantics to Higher-Order Quantum Computing*. In P. Sewell, editor: The 41th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL14, San Diego, USA, ACM.

[12] Peter Selinger (2004): *Towards a quantum programming language*. Mathematical Structures in Computer Science 14(4), pp. 527–586.

[13] Peter Selinger (2004): *Towards a semantics for higher-order quantum computation*. In: QPL'04, TUCS General Publication No 33, pp. 127–143.

[14] Peter Selinger & Benoit Valiron (2006): *A Lambda Calculus for Quantum Computation with Classical Control*. Math. Struct. Comput. Sci. 16(3), pp. 527–552.

[15] Peter Selinger & Benoît Valiron (2009): *Quantum Lambda Calculus*. In Simon Gay & Ian Mackie, editors: Semantic Techniques in Quantum Computation, chapter 9, Cambridge University Press, pp. 135–172.

[16] Christine Tasson (2009): *Sèmantiques et syntaxes vectorielles de la logique linèaire*. Ph.D. thesis, Université Paris 7.

[17] Benoît Valiron (2008): *Semantics for a higher-order functional programming language for quantum computation*. Ph.D. thesis, University of Ottawa.