PhD Proposal:

Solvability in Intuitionistic and Classical Call-by-Push-Value

General Context

The computational model behind modern **functional languages** and **proof assistants** is a formal system known as the λ-calculus, which provides a concise theoretical setting to conceptually express most of the properties relating to them. Indeed, every programming language/proof assistant implements a particular deterministic **evaluation strategy** which specifies, among other things, when parameters are evaluated during function calls. For example, in **call-by-value** (CbV), the argument is evaluated before being passed to the function, while in **call-by-name** (CbN) the argument is substituted directly into the function body, so that the argument may never be evaluated (if the argument is not used in the function body), or may be re-evaluated several times (each time it is used). For decades, CbN and CbV have been studied independently, developing distinct *ad hoc* techniques, until the remarkable observation that they are two different instances of a more general framework specified by Girard's **Linear Logic** (LL) [17]. Their logical duality ("CbN is dual to CbV") was understood later [10, 9] and their rewriting semantics were finally unified by the **call-by-push-value** (CBPV) paradigm introduced by P. B. Levy [21, 20], which is able to generalize different functional languages, i.e. different evaluation strategies.

Summary

This PhD proposal concerns the call-by-push-value paradigm, a formalism capable of encoding several functional evaluation strategies such as CbN and CbV in a single formalism.

The *first* goal of this thesis is to study the fundamental property of **solvability** [2] in the original framework of (intuitionistic) CBPV. This requires the study and comparison of different notions of solvability available in the literature, which are not always well-related, especially in the case of evaluation strategies ' The *second* goal of the thesis is to extend CBPV to classical logic, as well as its correspondent notion of solvability. Indeed, while pure functional programming (and thus its computational model the λ-calculus) is in deep correspondence with **intuitionistic logic** by means of the famous **Curry-Howard correspondence**, programming languages with continuations (modeled for example by the $\lambda\mu$-calculus [24]) correspond to **classical logic**.

Recent work has shown how it is possible to **unify** the theories of CbN and CbV [19, 14, 4]. It is then expected in both cases (intuitionistic and classical) that the obtained results of solvability in CBPV perfectly capture the solvability notions in (intuitionistic and classical) CbN and CbV.

Details of the Proposal

We now explain the key concepts of our proposal behind the summary and the goals mentioned above.

Call-by-Push-Value. The call-by-push-value paradigm, introduced by P. B. Levy [21, 20], distinguishes between values and computations under the slogan "a value is, a computation does". It

subsumes the λ-calculus by adding some primitives allowing to capture both call-by-name (CbN) and call-by-value (CbV) semantics. Essentially, CBPV introduces unary primitives `thunk` and `force` — the former freezes the execution of a term (i.e. it is not allowed to compute under a `thunk`) while the latter fires again a frozen term. Resorting to the paradigm slogan, `thunk` turns a computation into a value, while `force` does the opposite. Thus, CbN and CbV are captured by conveniently labelling a λ-term using `force` and `thunk` to pause/resume the evaluation of a subterm depending on whether it is an argument (CbN) or a function (CbV). Levy's `thunk` and `force` primitives can be seen under a linear logic perspective as two new constructors *bang* and *dereliction* [13]. Two interesting evaluation notions for CBPV can then be defined, depending on whether the reduction may take place under a bang constructor. These notions are known in the theory of programming languages as *weak* and *strong* reduction. So that the theory behind CBPV is in deep correspondence with fundamental principles of linear logic and various functional programming languages. Moreover, CBPV provides a unique formalism capturing distinct evaluation strategies, thus allowing to study operational and denotational semantics of different evaluation strategies in a unified framework.

Solvability. Solvability captures the fact that terms can operationally interact with the environment in order to produce a given output. Formally, in the CbN λ-calculus, a closed (i.e. without free variables) λ-term t is *solvable* if there are terms $u_1, \dots, u_n$ ($n \geq 0$) such that $t\ u_1 \cdots u_n$ reduces to a completely defined result (the identity program). Closed solvable terms represent meaningful programs as they are able to produce any desired result when applied to a suitable sequence of arguments. This notion can be easily extended to open terms (i.e. with free variables) by performing their closure.

In the literature there are several *operational* and *logical* characterizations of solvability for the CbN λ-calculus, that is:

$$t \text{ is solvable } \overset{\text{(operational)}}{\iff} \text{ the head reduction of } t \text{ terminates } \overset{\text{(logical)}}{\iff} t \text{ admits suitable intersection types}$$

The operational characterization of solvability has been proved in an untyped setting using the so-called *standardization theorem*. The logical characterization can be proved by building an appropriate *intersection type assignment system* characterizing head normalizable terms.

In the framework of CbV, the definition of solvability is analogous to the one in CbN, but the notion is trickier [23, 1, 15]. Moreover, it is not possible to provide an operational characterization of solvability in Plotkin's original formulation of the CbV λ-calculus [25], because reduction is somehow too restricted [6]. Operational characterizations of CbV solvability have been recently found in some extensions of Plotkin's CbV λ-calculus so that to characterize operationally CbV solvability it is necessary to go beyond Plotkin's original syntax [6]. A logical characterization of CbV solvability has been recently proposed in [22].

Intersection Types. Intersection types are based on a type constructor $\sigma \cap \tau$, which can be assigned to terms having both the type σ and the type τ. They usually enjoy associativity ($(\sigma \cap \tau) \cap \delta = \sigma \cap (\tau \cap \delta)$), commutativity ($\sigma \cap \tau = \tau \cap \sigma$), and **idempotency** ($\sigma \cap \sigma = \sigma$). Intersection type systems [8] have been introduced to increase the (limited) typability power of simple type assignment systems, but quite early they turned out to be a very powerful tool for characterizing semantic properties of λ-calculus, like solvability and strong normalization, and for describing models of λ-calculus in various settings. A flavour that became quite convenient in the last decade is that of **non-idempotent**

intersection types, where the intersection $\sigma \cap \sigma$ is not equivalent to σ. They first appeared in [16] but it is the seminal work of de Carvalho [7, 11], who found fundamental uses of non-idempotency to characterise quantitative properties of λ-calculus, stressing their importance. Roughly, distinguishing $\sigma \cap \sigma$ from σ gives rise to resource aware semantics of different λ-calculi, that is why they are also called **quantitative types**. A survey can be found in [5]. Non-idempotent intersections have two main features that we are going to exploit in the framework of this work:

1. Bounds on evaluation lengths: they go beyond simply qualitative characterizations of termination, as typing derivations provide quantitative bounds on the length of evaluation (i.e. on the number of β-steps) and on the size of the obtained normal forms. Therefore, they provide a tool to reason about the intentional insights on programs.

2. Linear logic interpretation: non-idempotent types are deeply linked to linear logic. The relational model [3, 12] of linear logic (often considered as a sort of canonical model of linear logic) is based on non-idempotent types, which can be seen as a syntactic presentation of the relational model of the λ-calculus induced by the interpretation into linear logic.

Computational Classical Logic. The Curry-Howard Isomorphism is a well-known relationship between programming languages and logical systems: while Curry first introduced the analogy between Hilbert-style deductions and combinatory logic, Howard highlighted the one between simply typed lambda calculus and natural deduction. Both analogies use intuitionistic logic. The extension of the Curry-Howard Isomorphism to classical logic took more than two decades, when Griffin [18] observed that Felleisen's C operator can be typed with the double-negation elimination. A major step in this field was done by Parigot [24], who proposed the $\lambda\mu$-calculus as a simple term notation for classical natural deduction proofs. The $\lambda\mu$-calculus is an extension of the simply typed λ-calculus that encodes usual control operators as the Felleisen's C operator mentioned so far. Other calculi were proposed since then, e.g. Curien-Herbelin's $\lambda\mu\tilde{\mu}$-calculus [9] based on classical sequent calculus.

The CBPV calculus introduced by P. B. Levy [21, 20] is intuitionistic, in the sense that it only models pure functional programming. An extension of the original CBPV to a classical setting is part of this PhD proposal.

Tentative timeline of the proposed research activities. During the first semester, the student will familiarize with the different notions involved in the thesis, e.g., solvability and call-by-push-value. A characterization of solvability in the setting of intuitionistic CBPV is expected by the end of the first year. During the second part of the thesis Victor Arrial will work on solvability extended to languages with continuations. The last six months of the three years will be consecrated to the writing of the thesis.

Technical Tools and Collaborations. A good knowledge of functional programming is required. Mathematical tools such as rewriting, λ-calculus, linear logic, and type systems will be needed. Many collaborations in the domain are carried out with S. Ronchi Della Rocca (Universite de Turin, Italie), A. Bucciarelli (IRIF, Universite Paris-Diderot), G. Guerrieri (Univ. Bath), B. Accattoli (LIX Ecole Polytechnique), and A. Viso (INRIA, France).

References

[1] B. Accattoli and L. Paolini. Call-by-value solvability, revisited. In *Functional and Logic Programming (FLOPS)*, pages 4–16, May 2012.

[2] H. Barendregt. *The Lambda Calculus: Its Syntax and Semantics*, volume 103 of *Studies in logic and the foundation of mathematics*. North-Holland, Amsterdam, revised edition, 1984.

[3] A. Bucciarelli and T. Ehrhard. On phase semantics and denotational semantics: the exponentials. *Ann. Pure Appl. Log.*, 109(3):205–241, 2001.

[4] A. Bucciarelli, D. Kesner, A. Ríos, and A. Viso. The bang calculus revisited. In *Functional and Logic Programming (FLOPS)*, pages 13–32, September 2020.

[5] A. Bucciarelli, D. Kesner, and D. Ventura. Non-idempotent intersection types for the lambda-calculus. *Logic Journal of the IGPL*, pages 431–464, 2017.

[6] A. Carraro and G. Guerrieri. A semantical and operational account of call-by-value solvability. In *Foundations of Software Science and Computation Structures (FOSSACS)*, pages 103–118, April 2014.

[7] D. d. Carvalho. *Sémantiques de la logique linéaire et temps de calcul*. PhD thesis, 2007.

[8] M. Coppo and M. Dezani-Ciancaglini. An extension of the basic functionality theory for the λ-calculus. *Notre Dame Journal Formal Logic*, pages 685—-693, 1980.

[9] P. Curien and H. Herbelin. The duality of computation. In *Int. Conf. on Functional Programming (ICFP)*, pages 233–243, September 2000.

[10] V. Danos, J. Joinet, and H. Schellinx. Lkq and lkt: Sequent calculi for second order logic based upon dual linear decompositions of classical implication. In *Advances in Linear Logic*, pages 222–211. 1995.

[11] D. de Carvalho. Execution time of λ-terms via denotational semantics and intersection types. *Mathematical Structures in Computer Science*, pages 1169–1203, 2018.

[12] D. de Carvalho, M. Pagani, and L. Tortora de Falco. A semantic measure of the execution time in linear logic. *Theorical Computer Science, Special issue Girard's Festschrift*, pages 1884–1902, 2011.

[13] T. Ehrhard. Call-by-push-value from a linear logic point of view. In *Programming Languages and Systems (ESOP)*, pages 202–228, April 2016.

[14] C. Faggian and G. Guerrieri. Factorization in call-by-name and call-by-value calculi via linear logic. In *Int. Conf. on Foundations of Software Science and Computation Structures (FOSSACS)*, pages 205–225, March-April 2021.

[15] Á. García-Pérez and P. Nogueira. No solvable lambda-value term left behind. *Logical Methods in Computer Science*, pages 125–159, 2016.

[16] P. Gardner. Discovering needed reductions using type theory. In *Theoretical Aspects of Computer Software, International Conference (TACS)*, pages 555–574, April 1994.

[17] J. Girard. Linear logic. *Int. Conf on Theoretical Computer Science (TCS)*, pages 1–102, 1987.

[18] T. Griffin. A formulae-as-types notion of control. In *Principles of Programming Languages (POPL)*, pages 47–58, January 1990.

[19] G. Guerrieri and G. Manzonetto. The bang calculus and the two girard's translations. In *Joint International Workshop on Linearity & Trends in Linear Logic and Applications*, pages 15–30, July 2018.

[20] P. B. Levy. Call-by-push-value: Decomposing call-by-value and call-by-name. *Higher-Order and Symbolic Computation*, pages 377–414, 2006.

[21] P. B. Levy. Call-by-push-value: A subsuming paradigm. In *Typed Lambda Calculi and Applications (TLCA)*, pages 228–242, April 1999.

[22] G. Manzonetto, A. Kerinec, and S. Ronchi Della Rocca. Call-by-value, again! In *International Conference on Formal Structures for Computation and Deduction*, LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2021. To appear.

[23] L. Paolini and S. R. D. Rocca. Call-by-value solvability. *RAIRO Theoretical Informatics and Applications*, pages 507–534, 1999.

[24] M. Parigot. Lambda-mu-calculus: An algorithmic interpretation of classical natural deduction. In *Logic Programming and Automated Reasoning, International Conference (LPAR)*, pages 190–201, July 1992.

[25] G. D. Plotkin. Call-by-name, call-by-value and the lambda-calculus. *Int. Conf. on Theoretical Computer Science (TCS)*, pages 125–159, 1975.

Encadrement (in French)

Giulio Manzonetto (Directeur), `giulio.manzonetto@lipn.univ-paris13.fr`

Laboratoire d'accueil: LIPN - Laboratoire d'Informatique de Paris Nord (UMR 7030 du CNRS). Université Paris 13, Villetaneuse.

Le Laboratoire d'Informatique de Paris Nord (LIPN - UMR 7030) joue un rôle majeur dans la recherche en informatique fondamentale en Île-de-France. En particulier, le laboratoire accueille, dans l'équipe LoVe (Logique et Vérification), un groupe de chercheurs et enseignant-chercheurs ayant une expertise bien établie et reconnue de longue date en logique linéaire, avec des applications à plusieurs domaines de l'informatique fondamentale: théorie de la démonstration, lambda-calcul et programmation fonctionnelle, sémantique dénotationnelle, complexité implicite. Ces domaines font partie des axes portants de l'équipe LoVe et du LIPN.

À présent, 2 professeurs, 5 maîtres de conférences, 4 CR CNRS et plusieurs doctorants travaillent activement sur ces thématiques. En particulier, le doctorant nouvellement recruté pourra bénéficier de l'expertise acquise par Kerinec sur la characterisation logique et sémantique de la solvabilité. L'équipe LoVe conduit ou contribue à plusieurs projets de recherche nationaux autour du sujet, et mène régulièrement des collaborations internationales.

La thése pourra être co-encadrée par Delia Kesner (Irif) qui a déjà supervisionné M. Arrial pendant son stage de Master 2 (MPRI).

Giulio Manzonetto
Maître de conférences
IUT de Villetaneuse
Laboratoire LIPN
Université Paris-Nord