Preface

This volume contains a selection of papers presented at DCM 2018, the Twelfth International Workshop on Developments in Computational Models, and ITRS 2018, the Ninth Workshop on Intersection Types and Related Systems. The workshops took place on July 8, 2018, in Oxford, UK as satellite events of FLoC 2018 in association with the ACM/IEEE Symposium on Logic in Computer Science (LICS) and the International Conference on Formal Structures for Computation and Deduction (FSCD).
The DCM workshop aims to bring together researchers who are currently developing new computational models or new features for traditional computational models, in order to foster their interaction, to provide a forum for presenting new ideas and work in progress, and to enable newcomers to learn about current activities in this area. Topics of interest include all abstract models of computation and their applications to the development of programming languages and systems.

The aim of the ITRS 2018 workshop series is to bring together researchers working on both the theory and practical applications of systems based on intersection types and related approaches. Over the last twenty years the scope of the research on intersection types and related systems has broadened in many directions, ranging from restricted (and more manageable) forms, to applications in areas such program analysis and higher-order model checking, to more recent research focusing on a behavioural/intensional approach to types.

The programs of the two workshops included two joint DCM/ITRS invited talks by Delia Kesner and Damiano Mazza, and two additional invited talks by Ugo Dal Lago (DCM invited speaker) and Pawel Parys (ITRS invited speaker). In addition, there were six contributed papers presented at DCM and six contributed papers presented at ITRS. For this volume containing the final proceedings, the program committees of DCM and ITRS 2018 select a total of five contributed papers: two contributed papers from DCM (A Category Theoretic Interpretation of Gandy’s Principles for Mechanisms by Joseph Razavi and Andrea Schalk, and A syntactic model of mutation and aliasing by Paola Giannini, Marco Servetto and Elena Zucca) and three contributed papers from ITRS (Natural Deduction and Normalization Proofs for the Intersection Type Discipline, by Federico Aschieri, Towards a Semantic Measure of the Execution Time in Call-by-Value lambda-Calculus, by Giulio Guerrieri, and Intersection Subtyping with Constructors by Olivier Laurent).

The volume also contains also an invited paper from ITRS (Intersection Types for Unboundedness Problems by Pawel Parys) and the abstracts from the two DCM/ITRS joint invited talks (Quantitative Types: From Foundations to Applications by Delia Kesner and Polyadic Approximations and Intersection Types by Damiano Mazza).

We wish to thank the organisers of FLoC 2018 for their support. We are indebted to the program committee members and the external referees for their careful and efficient work in the reviewing process. Finally we are grateful to the authors for submitting their work and the invited speakers for sharing their knowledge. A special thanks to EPTCS for supporting the publication of this joint issue.

Sandra Alves and Michele Pagani
DCM and ITRS Program Chairs

Program Committee of ITRS 2018:

Ugo de’ Liguoro (Università di Torino, Italy), Boris Duedder (University of Copenhagen, Denmark), Joshua Dunfield (Queen's University, Canada), Charles Grellois (Université Aix-Marseille, France), Naoki Kobayashi (University of Tokyo, Japan), Michele Pagani - chair (IRIF, Université Paris Diderot, France), Joe Wells (Heriot-Watt University, Scotland).

Program Committee of DCM 2018:
Polyadic Approximations and Intersection Types

In his paper introducing linear logic, Girard briefly mentioned how full linear logic could be seen, in an informal sense, as the limit of its purely linear fragment. Several ways of expanding and formalizing this idea have been developed through the years, including what I call polyadic approximations. In this talk, I will show how these are deeply related to intersection type systems, via a general construction that is sufficiently broad to allow recovering every well known intersection type system and their normalization properties, as well as introducing new type systems in wildly disparate contexts, with applications ranging from deadlock-freedom in the pi-calculus to a type-theoretic proof of the Cook-Levin theorem.

(Part of this work was developed jointly with Luc Pellissier and Pierre Vial).

Quantitative Types: From Foundations to Applications

Delia Kesner (IRIF, CNRS and Université Paris Diderot)

Among the well-known techniques used to reason about relevant aspects of programs we find the large world of type systems, whose development was consolidated by the famous Curry-Howard correspondence, providing a robust correspondence between proofs (in some logic) and well-typed programs (in some programming language). This discovery has already
provided important conceptual foundations for typed programming languages like OCaml and proof assistants like Coq.

The Curry-Howard correspondence has already contributed to our understanding of many aspects of programming languages by establishing a rich connection with intuitionistic and classical logic, however, there are still some quantitative crucial aspects of computation that need to be logically understood, like the use of time and space resources.

In this talk we give a theoretical understanding of the use of resources in programming languages by means of quantitative type systems. We introduce different alternatives, suitable for a wide range of powerful models of computation, such as for example pattern matching, control operators and infinitary computations.

From qualitative type systems ... Several notions of type assignment systems for \( \lambda \) -calculus have been defined in these last years. The most natural and oldest among them is given by the simple type assignment system, which has been further extended with polymorphic functions, i.e. functions that can be applied to various kinds of arguments, even of incomparable type. The resulting system is used as the core of different typing systems used nowadays in programming languages like ML and OCaml.

Even if polymorphic types are powerful and convenient in programming practice, they have several drawbacks. For example, it is not possible to assign a type to a term of the form \( \lambda x.xx \), which can be understood as a meaningful program specified by a terminating term.

Intersection types, pioneered by Coppo and Dezani [11, 12], introduce a new constructor for types, written \( \cap \), which allows to type the term \( \lambda x.xx \) with a type of the form \( (\sigma \to \sigma) \cap \sigma \to \sigma \). The intuition behind this is that a term \( t \) has type \( \tau_1 \cap \tau_2 \) if \( t \) has both types \( \tau_1 \) and \( \tau_2 \). The symbol \( \cap \) needs to be understood as a mathematical intersection, so in principle, intersection type theory was developed by using idempotent \( (\sigma \cap \sigma = \sigma) \), commutative \( (\sigma \cap \tau = \tau \cap \sigma) \), and associative \( ((\sigma \cap \tau) \cap \delta = \sigma \cap (\tau \cap \delta)) \) laws.

Intersection types are so powerful that they can be seen as a behavioural tool to reason about operational and semantical properties of different languages. They can be used in particular to characterise different properties of the \( \lambda \) -calculus. For example, a term/program \( t \) is strongly normalising/terminating if and only if \( t \) can be assigned an intersection type in an appropriate intersection type assignment system. Similarly, intersection types are able to describe and analyse models of \( \lambda \) -calculus [2], characterise solvability [31], head normalisation [31], linear-head normalisation [23], and weak-normalisation [31, 29] among other properties.

... to quantitative type systems This technology turns out to be a powerful tool to reason about qualitative properties of programs, but not for quantitative ones. Indeed, we know that a term is typable if and only if it verifies some property, but we cannot count the number of resources which are necessary for this term in order to verify the property. Here is where non-idempotent types, pioneered by [19, 25], come into play, making a clear distinction between \( \sigma \cap \sigma \)
and σ, because using the resource σ twice or once is not the same. This change of point of view can be related to the essential spirit of Linear Logic [20], which removes the contraction and weakening structural rules in order to provide an explicit control of the use of logical resources, i.e. to give a full account of the number of times that propositions are used to derive conclusions. Typing of Church numerals provide a clear example of the peculiarity of non idempotent intersection type systems, with respect to the usual, idempotent, ones. Let 
\[ n = \lambda x \lambda y. x(x(...(x\cdots x y y)\cdots)) \] 
be the \( n \)-th Church numeral, and let us consider the typing of \( 2 = \lambda x \lambda y. x(y) \) in the idempotent and non-idempotent systems respectively. The idempotent type system allows to type \{\em all\} Church numerals with the \em same\ type \(((\alpha \to \alpha) \to \alpha) \to \alpha\), here assigned also to the Church numeral \(2\). However, the non-idempotent type system would use a quantitative type information like \(((\alpha \to \alpha) \cap (\alpha \to \alpha)) \to \alpha\), in order to enforce the fact that the variable \(x\) is used twice in the Church numeral \(2\). Similarly, the Church numeral \(3\) will receive another type of the form \(((\alpha \to \alpha) \cap (\alpha \to \alpha) \cap (\alpha \to \alpha)) \to \alpha\), etc. Non-idempotent intersection types have been used to give denotational semantics of linear logic [20]. They give rise to different \em resource aware\ languages, pioneered by Kfoury [25] and Boudol, Curien and Lavatelli [6]. Relational models of \(\lambda\)-calculi based on non-idempotent types have been investigated in [9, 16]. D. de Carvalho [9] used non-idempotent intersection type systems to obtain exact bounds on evaluation lengths for the call-by-name lambda-calculus. This approach is also applied to linear logic proof-nets [10]. Non-idempotency is used to reason about the longest reduction sequence of strongly normalising terms in both the call-by-name lambda-calculus [3,13, 4] and in different (call-by-name) lambda-calculi with explicit substitutions [4, 23]. Extensions to classical logic also consider non-idempotent union types [24]. The case of call-by-value is addressed in [16, 8], while call-by-push value is studied in [17, 18] and call-by-need in [21, 22, 1]. Non-idempotent types also appear in linearisation of the lambda-calculus [25], type inference [26, 32], different characterisations of solvability [34] and verification of higher-order programs [33]. While the inhabitation problem for intersection types is known to be undecidable in the idempotent case [35], decidability was recently proved [7] through a sound and complete algorithm in the non-idempotent case. This list is not exhaustive.

In this talk we discuss quantitative properties that can be observed and formally characterised on a broad range of programming language paradigms.

On one side, we want to go beyond toy examples, like the \(\lambda\)

- calculus, to focus on the challenges posed by the \em advanced features\ that are found in modern higher-order programming languages, proof assistants and theorem provers such as \em pattern matching, infinite data, and/or control operators.\

On the other side, we do not want to restrict our study to a particular programming language, so that we can cover \em different reduction strategies\ used to implement these higher-order programming languages, such as for example \em call-by-value\ (OCaml), \em call-by-need\ (Haskell), \em standard strategies\ (call-by-name languages).\

Interesting connections exist, and would need to be understood better, between the use of quantitative techniques in program analysis and the emerging quantitative approach in well-established areas of computer science such as verification [27, 33], logic [28, 20] and automata [5, 15].
References


