Quantitative Inhabitation in Call-by-Value

Victor Arrial

Université Paris Cité - IRIF - CNRS

Abstract

We show the decidability of the inhabitation problem in a quantitative Call-by-Value setting.

Inhabitation. *Type systems* are formalisms made of rules assigning a *type* to the constructs of a programming language, usually represented by a *term calculus*. Types enforce some particular specification (*e.g.* termination, memory safety, deadlock freeness, etc), so that they guarantee the construction of well-behaved terms, in the sense that "well-typed programs cannot go wrong" [15]. Typing in a given type system $\mathcal{X}$ is written $\triangleright_{\mathcal{X}} \Gamma \vdash t : \sigma$, where t is a term, σ is the type assigned to t, and Γ is an *environment* assigning types to the (free) variables of t. The *inhabitation* problem naturally arises for any given type assignment system: given an environment Γ, and a type σ, decide whether there exists a term such that $\triangleright_{\mathcal{X}} \Gamma \vdash t : \sigma$. Inhabitation corresponds to decide the existence of a program (term t) that satisfies the given specification (type σ) under additional assumptions (environment Γ). Decidability of the inhabitation problem naturally provides tools for type-based *program synthesis* [14, 3], whose task is to construct—from scratch—a program that satisfies some high-level formal specification (the one guaranteed by the type assignment).

Quantitative Typing Systems. *Intersection type assignment systems* [8, 9] were introduced for the λ-calculus to increase the typability power of simple types by introducing a new *intersection* type constructor $\wedge$, which is, in principle, associative, commutative and *idempotent* ($\sigma \wedge \sigma = \sigma$). Intersection types allow terms having different types simultaneously, *e.g.* a term t has type $\sigma \wedge \tau$ whenever t has both the type σ and the type τ. In these (idempotent) systems typability and inhabitation are both undecidable [17]. However, intersection types constitute a powerful tool to reason about *qualitative* properties of programs, for example, there are intersection type systems characterizing different notions of normalization [16, 10], in the sense that a term t is typable in a given system if and only if t is normalizing for some particular notion. By removing idempotence [13, 11], a term of type $\sigma \wedge \sigma \wedge \tau$ can be seen as a resource that, during execution, can be used once as a data of type τ and twice as a data of type σ. The resulting *non-idempotent* type systems do not only provide qualitative characterization of operational properties, but also *quantitative* ones, in the sense that a term t is still typable if and only if is normalizing, and in addition, any type derivation of t gives a *bound* to the execution time for t (the number of steps to reach a normal form) [12, 1]. In such a setting, typability is still undecidable, nevertheless inhabitation has proven to be *decidable* in the Call-by-Name case [6, 7]. So, an algorithm solving the inhabitation problem for a quantitative type system provides a decidable tool for type-based *quantitative program synthesis*, which aims to construct—from scratch—a program that satisfies some quantitative specification.

Call-by-Value. Call-by-Value evaluation is the most common parameter passing mechanism for programming languages: parameters are evaluated before being passed. We use the $\lambda_{\mathtt{vsub}}$ calculus introduced by Accattoli and Paolini [2] which exploits explicit substitutions for both delaying CBV redex restrictions as well as acting at a distance, and which thus presents good operational properties. This calculus can be equipped with a non-idempotent intersection type system [4] (presented in Fig. 1) which characterizes head normalization. We use multisets to denote intersections.

$$\frac{}{x : \mathcal{M} \vdash x : \mathcal{M}}\ \mathsf{ax} \qquad \frac{(\ \Gamma_i ; x : \mathcal{M}_i \vdash t : \sigma_i)_{i \in I}}{+_{i \in I} \Gamma_i \vdash \lambda x.t : [\mathcal{M}_i \Rightarrow \sigma_i]_{i \in I}}\ \mathsf{abs}$$

$$\frac{\Gamma_1 \vdash t : [\mathcal{M} \Rightarrow \sigma] \qquad \Gamma_2 \vdash u : \mathcal{M}}{\Gamma_1 + \Gamma_2 \vdash tu : \sigma}\ \mathsf{app} \qquad \frac{\Gamma_1 ; x : \mathcal{M} \vdash t : \sigma \qquad \Gamma_2 \vdash u : \mathcal{M}}{\Gamma_1 + \Gamma_2 \vdash t[x \backslash u] : \sigma}\ \mathsf{es}$$

Figure 1: Call-by-Value Type System $\mathcal{V}$

Contributions. We show that the inhabitation problem for the Call-by-Value λ-calculus with respect to the quantitative type system $\mathcal{V}$ [4] is decidable. We do not simply give an algorithm searching for a term that can be typed with a given environment Γ and type σ, but we solve a more ambitious goal: we look for *all* and *only* such typable terms. This provides either a strong and powerful tool for (quantitative) program synthesis.

Canonical Derivations as Finite Basis: The set of all solutions of any instance of the inhabitation problem is either infinite or empty. Building an algorithm providing all solutions for a given instance is therefore worthless. However, following the technique used in the Call-by-Name λ-calculus case [6, 7] and generalizing it to explicit substitutions [5] allows us to introduce the notion of *canonical derivation* which later forms a finite basis for the solution set of each instance. It is a basis as it *exactly generates* each solution set through two simple operations: redex expansion and term plugging. We show how to compute the canonical representative of any solution as well as how it is recovered from its canonical representative, thus providing proofs of *correction* and *completeness* for each basis.

Basis Search Algorithm: Equipped with such tools, we provide an algorithm computing the basis for any given instance. It highly exploits a central property of any basis of terms called *head subtype property* which indicates that some of its types are contained in the context and thus helps guiding the search. This (non-deterministic) algorithm is shown to be *correct* and *complete* as it finds all and only basis terms. We show that it can in fact be *deterministically simulated* in a finite time, which provides for free a proof of the finiteness of each basis. It therefore constitutes a program synthesis mechanism for an expressive programming language.

References

[1] Beniamino Accattoli, Stéphane Graham-Lengrand, and Delia Kesner. Tight typings and split bounds, fully developed. J. Funct. Program., 30:e14, 2020.

[2] Beniamino Accattoli and Luca Paolini. Call-by-value solvability, revisited. In Tom Schrijvers and Peter Thiemann, editors, FLOPS, volume 7294 of LNCS, pages 4–16. Springer, 2012.

[3] Jan Bessai, Tzu-Chun Chen, Andrej Dudenhefner, Boris Düdder, Ugo de'Liguoro, and Jakob Rehof. Mixin composition synthesis based on intersection types. CoRR, abs/1712.06906, 2017.

[4] Antonio Bucciarelli, Delia Kesner, Alejandro Ríos, and Andrés Viso. The bang calculus revisited. In Keisuke Nakano and Konstantinos Sagonas, editors, Functional and Logic Programming - 15th International Symposium, FLOPS 2020, Akita, Japan, September 14-16, 2020, Proceedings, volume 12073 of Lecture Notes in Computer Science, pages 13–32. Springer, 2020.

[5] Antonio Bucciarelli, Delia Kesner, and Simona Ronchi Della Rocca. Solvability = Typability + Inhabitation. Logical Methods in Computer Science, Volume 17, Issue 1, January 2021.

[6] Antonio Bucciarelli, Delia Kesner, and Simona Ronchi Della Rocca. The inhabitation problem for non-idempotent intersection types. In TCS, volume 8705 of LNCS, pages 341–354. Springer, 2014.

[7] Antonio Bucciarelli, Delia Kesner, and Simona Ronchi Della Rocca. Inhabitation for non-idempotent intersection types. Logical Methods in Computer Science, 14(3), 2018.

[8] M. Coppo and M. Dezani-Ciancaglini. A new type assignment for λ-terms. Archiv für mathematische Logik und Grundlagenforschung, 19(1):139–156, 1978.

[9] M. Coppo and M. Dezani-Ciancaglini. An extension of the basic functionality theory for the λ-calculus. Notre Dame J. Form. Log., 21(4):685–693, 1980.

[10] Mario Coppo, Mariangiola Dezani-Ciancaglini, and Betti Venneri. Functional characters of solvable terms. Math. Log. Q., 27(2-6):45–58, 1981.

[11] Daniel de Carvalho. Sémantiques de la logique linéaire et temps de calcul. PhD thesis, Université Aix-Marseille II, 2007.

[12] Daniel de Carvalho. Execution time of λ-terms via denotational semantics and intersection types. Math. Struct. Comput. Sci., 28(7):1169–1203, 2018.

[13] Philippa Gardner. Discovering needed reductions using type theory. In Masami Hagiya and John C. Mitchell, editors, Theoretical Aspects of Computer Software, pages 555–574, Berlin, Heidelberg, 1994. Springer Berlin Heidelberg.

[14] Zohar Manna and Richard J. Waldinger. A deductive approach to program synthesis. ACM Trans. Program. Lang. Syst., 2(1):90–121, 1980.

[15] Robin Milner. A theory of type polymorphism in programming. Journal of Computer and System Sciences, 17(3):348–375.

[16] Garrel Pottinger. A type assignment for the strongly normalizable λ-terms. In To H.B.Curry: Essays on Combinatory Logic, Lambda-calculus and Formalism, pages 561–577. Academic Press, 1980.

[17] Pawel Urzyczyn. The emptiness problem for intersection types. Journal of Symbolic Logic, 64(3):1195–1215, 1999.