Discrete Polymorphism with Gradual Typing

Pedro Ângelo

Faculdade de Ciências & LIACC, Universidade do Porto, Porto, Portugal
pjfangelo@hotmail.com

1 Introduction

With the aim of seamlessly integrating static and dynamic typing, and hence harnessing the advantages of both typing disciplines, gradual typing [3,4,8] allows to fine-tune the distribution of static and dynamic type checking in a program. To accomplish this, gradual typing introduces the dynamic type, which stands for unknown type information, and the consistency relation, required for the comparison of types with dynamic components. By annotating lambda-abstractions with the dynamic type, one can delay type checking of that expression until run-time. By deciding to add or remove dynamic type annotations, the quantity of static and dynamic type checking also varies. Types that are compared with the consistency relation need to be checked at run-time, to ensure program correctness. Run-time checks are performed via casts, which check the actual type of expressions against its expected type. To illustrate, consider the expression $(\lambda x : Dyn \ . \ x + 1) \ true$. It clearly has a type error, however, by adding the dynamic type, we are delaying the type checking to runtime, so this error will only be uncovered when the program is running.

The successful application [8] of gradual typing to the parametric polymorphic Hindley-Milner (HM) type system [7, 10, 16] marks an important breakthrough, showing that it is possible to apply it to statically typed functional programming languages, which typically include polymorphism, such as Haskell or ML.

Intersection types, originally defined in [5], extend the simply typed lambda-calculus [9], by adding to the language of types an intersection operator $\cap$ and allowing the same terms to be typed with different types. Thus, intersection types provide a form of polymorphism, called discrete polymorphism, in which it is possible to explicitly indicate every single instance of a type. Type systems based on these types are able to type more programs than the HM type system, some are able to type all the strongly normalizing terms, and also allow for increased expressiveness when describing instances of polymorphic types. Since the original publication, several other contributions have been published, which focus on different aspects of intersection types, such as refined and improved type systems [2, 6, 15, 17]; type inference and fragments [11, 13, 14] and surveys [1]. A reccuring example of an expression typed with intersection types is $\lambda x \ . \ x \ x$. By assigning type $\alpha \to \beta \cap \alpha$ to the variable x, the expression is typed with $(\alpha \to \beta \cap \alpha) \to \beta$.

Although the type inference problem for intersection types is not decidable in general, it becomes decidable for finite rank fragments of the general system [13, 14]. In particular, rank 2 intersection types have a type inference problem which is near in complexity to type inference for parametric polymorphic programming languages such as ML, while typing more programs than ML [11, 12].

2 Research Plan

My research nowadays consists in developing functional languages and static verification mechanisms that integrate intersection types with gradual typing.

Problem being addressed and it's relevance Given both the advantages of gradual typing and intersection types, the potential of harnessing the advantages of both verification techniques was what compelled me to pursue my current research topic. The integration of these two verification techniques is technically challenging due to their fundamentally different properties. In gradual typing, the type that will be assigned to a variable is the type in the annotation, while in intersection types, it can be one instance of a polymorphic type. Further, since the original casts introduced by [4] are not prepared to deal with intersection types, a new compilation and operational semantics must be defined to accommodate the different properties of these two techniques. There are few intersection type implementations, such as [17], so I believe my work will help the development and implementation of intersection type-based languages.

My solution Currently, my solution to integrate the two techniques in a type system and type inference mechanisms follows standard practice in the area. In the case of the type system, the solution consists of adding new type rules which deal directly with intersection types, to the gradual type system. Developing a type inference mechanism is more challenging. In this case, my research focus on extending previous algorithms for finite rank intersection types to deal with gradual types. This extension has some serious challenges and our first algorithm for rank 2 types defines a non-deterministic procedure to infer a set of most general types. As future work, I also want to develop a compilation phase which inserts appropriate casts and operational semantics based on the blame calculus, able to verify gradual intersection types at run-time.

Research Approach After getting up to date on the state of the art, my goal is to develop a type system with the intended properties I seek. Then, it follows a proof of soundness and the verification of several correctness criteria of the new type system with respect to a new operational semantics for a core-language explicitly typed with gradual intersection types. This type system and the underlined operational semantics for the language are going to be implemented in Haskell with the aim of a future integration on Core-Haskell. Finally, my goal is to devise a type inference mechanism for a finite rank restriction of the type system and verify its correctness and, eventually, its completeness.

Expected Contribution The ultimate goal of this work is to develop and implement a functional programming language which enables both intersection types and gradual typing while also automatically inferring types for programs. Several intermediate technical results that will stem from my research and will aid in reaching the ultimate goal are the specification of annotated intersection type systems, combining gradual typing with intersection types into a correct type system, the definition of an operational semantics which will allow the use of intersection types in a gradual framework, and finally a type inference algorithm for this new hybrid type system. All these technical results are expected to be proved correct using formal proofs and/or automated proofs.

3 Progress to date and current state of research

To date, I have defined a first draft of a gradual system with intersection types [19], although much work still has to be done. Currently, I'm developing a gradual intersection type system, which already has all its associated gradual criteria proofs [18]. I also defined a type inference algorithm for a rank 2 gradual intersection type system which was accepted for presentation at the symposium Trends in Functional Programming 2019 [20]. Next, I will start working on the compilation phase and the definition of an operational semantics with casts for dynamic type checking and relate this type aware semantics with the original type system.

References

1. van Bakel, S.: Intersection type disciplines in lambda calculus and applicative term rewriting systems. Amsterdam: Mathematisch Centrum (1993)
2. Barendregt, H., Coppo, M., Dezani-Ciancaglini, M.: A filter lambda model and the completeness of type assignment. Journal of Symbolic Logic **48**(4), 931–940 (1983)
3. Cimini, M., Siek, J.G.: The gradualizer: A methodology and algorithm for generating gradual type systems. In: Proceedings of the 43rd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages. pp. 443–455. POPL '16 (2016)
4. Cimini, M., Siek, J.G.: Automatically generating the dynamic semantics of gradually typed languages. In: Proceedings of the 44th ACM SIGPLAN Symposium on Principles of Programming Languages. pp. 789–803. POPL 2017 (2017)
5. Coppo, M., Dezani-Ciancaglini, M.: An extension of the basic functionality theory for the λ-calculus. Notre Dame Journal of Formal Logic **21**(4), 685–693 (1980)
6. Coppo, M., Dezani-Ciancaglini, M., Venneri, B.: Functional characters of solvable terms. Mathematical Logic Quarterly **27**(2-6), 45–58 (1981)
7. Damas, L., Milner, R.: Principal type-schemes for functional programs. In: Proceedings of the 9th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages. pp. 207–212. POPL '82 (1982)
8. Garcia, R., Cimini, M.: Principal type schemes for gradual programs. In: Proceedings of the 42nd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages. pp. 303–315. POPL '15 (2015)

9. Hindley, J.R.: Basic Simple Type Theory. Cambridge Tracts in Theoretical Computer Science, Cambridge University Press (1997)
10. Hindley, R.: The principal type-scheme of an object in combinatory logic. Transactions of the American Mathematical Society **146**, 29–60 (1969)
11. Jim, T.: Rank 2 type systems and recursive definitions. Tech. rep., Cambridge, MA, USA (1995)
12. Jim, T.: What are principal typings and what are they good for? In: Proceedings of the 23rd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages. pp. 42–53. POPL '96 (1996)
13. Kfoury, A.J., Wells, J.B.: Principality and decidable type inference for finite-rank intersection types. In: Proceedings of the 26th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages. pp. 161–174. POPL '99 (1999)
14. Kfoury, A., Wells, J.: Principality and type inference for intersection types using expansion variables. Theoretical Computer Science **311**(1), 1 – 70 (2004)
15. Liquori, L., Rocca, S.R.D.: Intersection-types à la church. Information and Computation **205**(9), 1371 – 1386 (2007)
16. Milner, R.: A theory of type polymorphism in programming. Journal of Computer and System Sciences **17**(3), 348 – 375 (1978)
17. Reynolds, J.C.: Design of the Programming Language Forsythe, pp. 173–233. Birkhäuser Boston, Boston, MA (1997)
18. Siek, J.G., Vitousek, M.M., Cimini, M., Boyland, J.T.: Refined Criteria for Gradual Typing. In: 1st Summit on Advances in Programming Languages (SNAPL 2015). Leibniz International Proceedings in Informatics (LIPIcs), vol. 32, pp. 274–293. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany (2015)
19. Ângelo, P., Florido, M.: Gradual intersection types. In: Ninth Workshop on Intersection Types and Related Systems, ITRS 2018, Oxford, U.K., 8 July 2018 (2018)
20. Ângelo, P., Florido, M.: Type inference for rank 2 gradual intersection types. In: Trends in Functional Programming, TFP 2019, Vancouver, B.C., Canada, 14 June 2019 (2019)