Type-Checking Linear Dependent Types

Arthur Azevedo de Amorim[1], Emilio Jesús Gallego Arias[1], Marco Gaboardi[2], and Justin Hsu[1]

[1] University of Pennsylvania
[2] University of Dundee

Linear indexed type systems have been used to ensure safety properties of programs with respect to different kinds of resources; examples include usage analysis [10], implicit complexity [3], and more. Linear indexed types use a type-level *index language* to describe resources and *linear types* to reason about the program's resource usage in a compositional way.

A limitation of current analysis techniques for such systems is that resource usage is inferred independently of the control flow of a program—e.g., the typing rule for branching usually approximates resources by taking the maximal usage of one of the branches. To make this analysis more precise, some authors have proposed extending adding dependent types, considering both resource usage and the *size information* of a program's input. This significantly enriches the resulting analysis by allowing resource usage to depend on runtime information. Linear dependent type systems have been used in several domains, such as implicit complexity [1] and others.

Of course, there is a price to be paid for the increase in expressiveness: type checking and type inference inevitably become more complex. In linear indexed type systems, these tasks are often done in two stages: a standard Hindley-Milner-like pass, followed by a constraint-solving procedure. In some cases, the generated constraints can be solved automatically with custom algorithms [6] or off-the-shelf SMT solvers [4]. However, the constraints are specific to the index language, and richer index languages often lead to more complex constraints.

In this work we consider the type-checking problem for a particular system with linear dependent types, *DFuzz*. *DFuzz* was born out of *Fuzz* [9], a language where types are used to reason about sensitivity of programs, which measures the distance between outputs on nearby inputs. *Fuzz* uses real numbers as indices for the linear types, which provide an upper bound on the sensitivity of the program. As shown by [4], type-checking *Fuzz* programs can be done efficiently by using an SMT solver to discharge the numeric proof obligations arising from the type system. The same approach works for type inference, which infers the minimal sensitivity of a function.

DFuzz [5] was introduced to overcome a fundamental limitation of *Fuzz*: sensitivity information cannot depend on runtime information, such as the size of a data structure. This is done by enriching *Fuzz* with a limited form of dependent types, whose index language combines information about the *size* of data structures and the *sensitivity* of functions. These changes have a significant impact on the difficulty of type checking, since type checking constraints in *DFuzz* may involve general polynomials rather than just constants.

One solution could be to extend the algorithm proposed in [4] to work with the new index language by generating additional constraints when dealing with the new constructs. This would be similar in spirit to the work of [2] for type inference for dℓPCF, a linear dependent type system for complexity analysis. Unfortunately, such an approach does not work as well for *DFuzz*, since it relies on the presence of arbitrary computable functions in the index language, whereas *DFuzz*'s index language is far simpler. Instead, since the type system of *DFuzz* also supports subtyping, we consider a different approach inspired by techniques from the literature on subtyping (e.g. [7]) and on constraint-based type-inference approaches (e.g. [8]).

The main idea is to type-check a program by inferring some set of sensitivities for it, and then testing whether the resulting type is a subtype of the desired type. To obtain completeness, one must ensure that the inferred sensitivities are the “best” possible. Unfortunately, the *DFuzz* index language is not rich enough for expressing such sensitivities. For instance, some cases require taking the maximum of two sensitivity expressions, which may not lie inside the basic sensitivity language. We solve this problem by extending the index language with a handful of index constructs to ease sensitivity-inference; we call this new system *EDFuzz*. We present a sensitivity-inference algorithm for *EDFuzz*, which we show sound and complete. Furthermore, *EDFuzz* has similar meta-theoretic properties as *DFuzz*.

We are left with the problem of solving the constraints generated by our algorithm. First, we show how to compile the constraints generated by the algorithmic systems to first-order constraints, allowing us to use standard solvers. Unfortunately, the resulting set of constraints is too powerful, and we also show that type checking for *DFuzz* is undecidable. We discuss how to approximate complete type-checking with a constraint relaxation procedure that is enough to handle the examples proposed in [5].

References

[1] Ugo Dal Lago and Marco Gaboardi. Linear dependent types and relative completeness. In *IEEE Symposium on Logic in Computer Science (LICS), Toronto, Ontario*, pages 133–142. IEEE, 2011.

[2] Ugo Dal Lago, Barbara Petit, et al. The geometry of types. In *ACM SIGPLAN–SIGACT Symposium on Principles of Programming Languages (POPL), Rome, Italy*, pages 167–178, 2013.

[3] Ugo Dal Lago and Ulrich Schöpp. Functional programming in sublinear space. In *ACM Transactions on Programming Languages and Systems*, pages 205–225. Springer, 2010.

[4] Loris D'Antoni, Marco Gaboardi, Emilio Jesús Gallego Arias, Andreas Haeberlen, and Benjamin C. Pierce. Sensitivity analysis using type-based constraints. In *Workshop on Functional Programming Concepts in Domain-specific Languages (FPCDSL)*, FPCDSL '13, pages 43–50, New York, NY, USA, 2013. ACM.

[5] Marco Gaboardi, Andreas Haeberlen, Justin Hsu, Arjun Narayan, and Benjamin C. Pierce. Linear dependent types for differential privacy. In *ACM SIGPLAN–SIGACT Symposium on Principles of Programming Languages (POPL), Rome, Italy*, POPL '13, pages 357–370, New York, NY, USA, 2013. ACM.

[6] Ugo Dal Lago and Ulrich Schöpp. Type inference for sublinear space functional programming. In Kazunori Ueda, editor, *Asian Symposium on Programming Languages and Systems (APLAS), Shanghai, China*, volume 6461 of *Lecture Notes in Computer Science*, pages 376–391. Springer, 2010.

[7] Benjamin C. Pierce and Martin Steffen. Higher-order subtyping. In *IFIP Working Conference on Programming Concepts, Methods and Calculi (PROCOMET)*, pages 511–530, 1994. Full version in *Theoretical Computer Science*, vol. 176, no. 1–2, pp. 235–282, 1997 (corrigendum in TCS vol. 184 (1997), p. 247).

[8] François Pottier and Didier Rémy. The essence of ML type inference. In Benjamin C. Pierce, editor, *Advanced Topics in Types and Programming Languages*, chapter 10, pages 389–489. MIT Press, 2005.

[9] Jason Reed and Benjamin C. Pierce. Distance makes the types grow stronger: A calculus for differential privacy. In *ACM SIGPLAN International Conference on Functional Programming (ICFP), Baltimore, Maryland*, ICFP '10, pages 157–168, New York, NY, USA, 2010.

[10] Philip Wadler. Is there a use for linear logic? In *Symposium on Partial Evaluation and Semantics-Based Program Manipulation (PEPM), New Haven, Connecticut*, volume 26, pages 255–273. ACM, 1991.